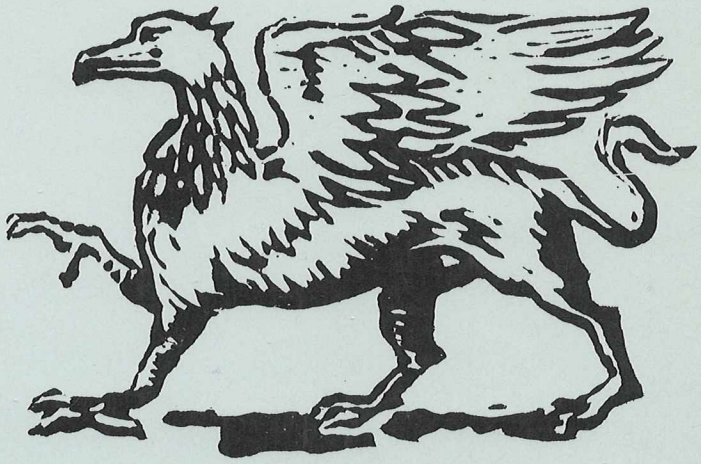
Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Three



The People's Computer Company Series

Dr. Dobb's Journal

of Computer Calisthenics & Orthodontia,

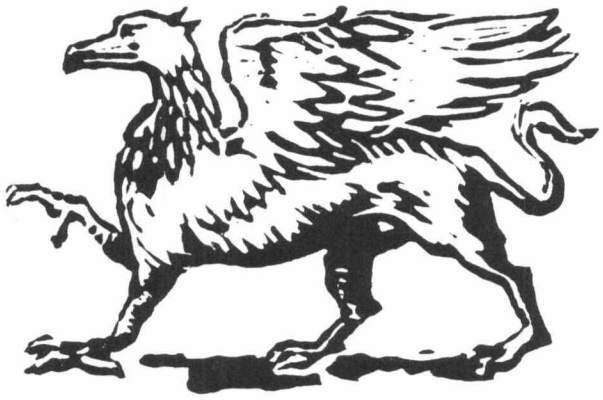
Volume Three

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Three



The People's Computer Company Series

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

Acknowledgments

Profuse thanks to all the authors and readers who contribute articles and share ideas with Dr. Dobb's. *Dr. Dobb's Journal*, distributed worldwide, owes its existence to you. Thanks also to the employees and other supporters of People's Computer Company who have seen *Dr. Dobb's* through its inception, growing pains and success.

Full-page computer art by David Dameron.

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

ISBN 0-8104-5490-4

Copyright © 1980 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

	3	4	5	6	7	8	9	PRINTING
81	82	83	84	85	86	87	88	YEAR

Editor's Preface

The third volume of *Dr. Dobb's Journal* represented a year of growth and maturity in the field of small computers, a maturity evident among manufacturers and users. Although *Dr. Dobb's* has always been primarily a software journal, Volume III reflects this increased sophistication with a substantial increase in the number of software articles. It branched out to include more microprocessors, most notably the 6502, used in the Apple, PET, and KIM 1. Software was somewhat different from that submitted in 1977; the earlier emphasis on systems yielded to the exploration of programming languages and to theoretical articles on computing in general. New implementations of languages for microcomputers SAM-76, UCSD PASCAL, BASEX, and FORTH were given extensive treatment.

Yet, there were still specialized applications, utilities, and unique discoveries made by readers eager to share their findings. Graphics programs for various systems, a Z-80 debugger, a KIM-1 editor, a renumbering program for the Apple, several 6502 cassette interface routines, and much, much more were published.

Many of the articles stressed technique rather than specific implementation, which speaks for the sophistication of the readers. As the year progressed, readers were able to adapt techniques and examples to their particular hardware system requirements. Specific applications articles stressed the algorithmic aspect more than in the past. (For example, see Richard Emerson's "Shorter Printer Driver for PolyMorphic BASIC." The article gives a specific application, and gives the reader introduction to the concept of circular buffers. J.M. Terry offers an equally informative discussion of the basics of list processing; Mike Gabrielson gives a clear and entertaining look at "Binary Tree Manipulation.")

Reader debates ranged from controversies over hex versus octal notation to the relative merits of languages. Follow, for example, the lively exchanges between the proponents of FORTH and STRUBAL to get the flavor of discussion.

By the end of 1978, it was apparent that small computers could have as great an effect on human society as printing has had. A tremendous base of knowledge for computers had been established. Industry gurus were losing sleep trying to predict where it was all going to lead.

And where *will* it lead?

Among the effects the microcomputer will have on our world will be a profound increase in the volume and quantity of communication. When small computers are linked by means of satellites to other computers as well as to large data bases, the world will see an information revolution before which the invention of printing will pale. That day has been brought closer by the readers and contributors of *Dr. Dobb's*.

Tom Williams
Editor, Volume III

Contents

VOLUME 3, NUMBER 1, 1978

13 The Heath H-8 "Evaluation": I Think We Blew It! Jim C. Warren, Jr., Editor
14 Memory Test for 6502 Carl Moser
15 Consarn That H-8! Craig A. Pearce
16 Challenging Challenger's ROMs Stephen P. Smith
18 Is Your Assembly Language Program Well Structured? Webb Simmons
20 ISIS: Anatomy of a Real-World Operating System Perry C. Hutchison
28 **The SAM76 Language** Ancelme Roichel
48 An 8080 Disassembler & Support Software for SOL Don Van Dyke

VOLUME 3, NUMBER 2, 1978

60 ET TU, KLATU? Stanford News Service
61 Decoding Efficiency & Speed: Pros & Cons of Table Look-up H.T. Gordon
64 MITS 4.0/4.1 BASIC Searcher/Patcher James Monagan
66 **A Simple Technique for Static Relocation of Absolute Machine Code** Gary A. Kildall
70 Relocation is Not Just Moving Programs Richard L. Roth
78 **Absolutely Dazzling Dazzleware** Gordon French & Gregory Yob
86 **Dazzler Driver Subroutine and a "Chase"** Phillip Simpson
98 Z-80 RAM Tester John MacDougall
99 Mini Editor for 8080 PILOT Tim Scully
101 The Last Rehash of Beau Hex? letters in the unending(?) hex/octal debate

VOLUME 3, NUMBER 3, 1978

106 **Renumbering and Appending BASIC Programs on the Apple-II Computer** Steve Wozniak
112 STRUBAL: A Structured BASIC Compiler Robert Grappel
113 A Z-80 Tracer Arthur W. Cline
115 A KIM Binary Calculator John Eaton
118 Status of the UCSD PASCAL Project Kenneth Bowles
122 **ACT – An 8080 Macroprocessor – Part I** Alex Cecil
148 SAM 76 – Additional Comments Ancelme Roichel

VOLUME 3, NUMBER 4, 1978

152 **List Processing Unravelled: Pointers, Chains, Rings and Trees** J.M. Terry
157 The Oznaki "Life" Harvey A. Cohen
159 A 1K Debugger for the Z-80 Les Hancock
166 Exit This Reviewer Jef Raskin
169 A Line-Oriented PILOT Editor David J. Morgan
174 **ACT–An 8080 Macroprocessor – Part II** Alex Cecil

VOLUME 3, NUMBER 5, 1978

198 **Programming Language Translation Techniques** W. M. McKeeman
211 Yet Another Body Ken Anderson
215 FORTH for Microcomputers John James
222 Resident Memory Test Systems Terry Ritter
228 EDITHA: A Revision of the Fylstra KIM-1 Editor Program H.T. Gordon
234 **Proposed Standard for the S-100 Bus** George Morrow and Howard Fullmer

VOLUME 3, NUMBER 6, 1978

244 **Intelligent Terminal Implementation on the S-100 Bus** Tim Pugh
257 Autodump Richard Emerson
263 **High-Speed Cassette I/O for the KIM-1** Theodore E. Bridge
267 The Zilog Z-8 U.R. Source
270 LAZARUS Andrew Hertzfeld
273 A Tiny BASIC Extensions Package Leor Zolman

VOLUME 3, NUMBER 7, 1978

290 **A Compact 8080 Disassembler & Dumper** Richard Greenlaw
305 The Intel 8086 Mike Gabrielson
308 A 1K Utilities Package for the Z-80 Les Hancock
313 A Shorter Printer Driver for PolyMorphic's BASIC Richard B. Emerson
317 Fast Cassette Interface for the 6502 Carl Moser
320 A BASIC Renumbering Program John B. Trenholme
328 Custom File Routines for ALS-8 Larry A. Marek
332 Common PILOT: Progress Report Larry Kheriaty and George Gerhold

VOLUME 3, NUMBER 8, 1978

338 **EM8080: An 8080 Emulator for 6800 Systems** Gary Gaugler
354 The Last of the First Aubrey Dahl
358 Lisp for the 6800 Fritz van der Wateren
360 FORTH Dump Programs John James
364 Making a New Case for the H-9 Terminal Howard L. Nurse
367 **A Programmable IC Tester for KIM** Joel Swank
372 Dumping Northstar Disk Files Titus Purdin
376 Vocal Memory Dump Phil Mork

VOLUME 3, NUMBER 9, 1978

386 LEDIP: A KIM/6502 Line Editor Kiumi Akingbehin
395 **Quick and Dirty Routines for BASIC** Dennis Allison
398 The Digicast™ Project Jim Warren
400 **POW: An 8080 'Processor of Words'** Herman Watson
412 Data-Boy™ Speech Processor James Anderson
416 Whatis: An 8080 Expression Evaluator Mike Gabrielson

VOLUME 3, NUMBER 10, 1978

429 **A LISP interpreter for the 8080** Darrel Van Buer
437 Curve-Fitting Using a Focal Interpreter on the KIM-1 Theodore E. Bridge
440 Binary Tree Manipulation on the 8080 Mike Gabrielson
443 Programming Languages and Standards David Chapman
444 SAM-76 Language Update Ancelme Roichel
447 **The Heath H-8 System** Tom Williams
451 BASEX: A Fast, Compact Interactive Compiler Paul K. Warme
457 Tiny GRAFIX for Tiny BASIC Joel Swank
459 **The STD Bus** Keith Britton
465 Copying a Diskette on Intel Double Density DOS Paul M. Julich

474 **Index to Volumes One, Two, and Three**

About People's Computer Company

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

As one of PCC's founders, Bob Albrecht explained in the first issue of the PCC newspaper, "Computers are mostly used against people instead of for people, used to control people instead of to free them. It's time to change all that. We need a People's Computer Company."

Since then, in its role as a non-profit, educational corporation, PCC has published a newspaper, three magazines, and several books on home/personal computing. The Personal Computer NETwork, (PCNET) joined People's Computer Company in March, 1979. PCNET is a Bay Area, California group dedicated since 1977 to the development of personal computer-based telecommunications. Computertown, U.S.A.!, a computer literacy project joined People's Computer Company in August, 1979. In the past PCC also operated a neighborhood computer center, and a mail-order bookstore.

Today, the PCC magazines – **Recreational Computing**, **Dr. Dobb's Journal**, and **Computer Music Journal** – have a combined circulation of about 28,000 (paid subscribers plus newsstand sales).

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

In PCC's view, familiarity with computers breeds confidence and personal power. "The computer-liberated kid grows up to be an adult unafraid to challenge his bills – or anything else in the world's great electronic system," says Albrecht.

People's Computer Company publishes three magazines for three distinct audiences. The oldest, **Recreational Computing**, is for the computer novice to the intermediate hobbyist. It started out as a newspaper called **People's Computer Company**, evolved into **People's Computers** magazine, and took its present name with the January-February 1979 issue. The magazine is published bimonthly.

"It's in the home—in a relaxed, recreational environment—that the real computer revolution is going to take place over the next few years," said Albrecht.

As the first publication in the personal computing field, **Recreational Computing** has spawned imitators. "But we feel we have a very special niche," says publisher Willard Holden. "**Recreational Computing** takes risks that other publications don't. We are willing to publish articles that may be off-beat and outrageous if we think they will get people excited about the possibilities of computers."

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

Dr. Dobb's Journal is a reference journal read worldwide. Ten percent of the circulation is outside the United States. The magazine contains sophisticated discussions of programming techniques and hardware modifications. About half of each issue is devoted to free software. **DDJ** appears ten times a year.

Computer Music Journal, the third PCC publication, is at the forefront of new developments in the computer music world. During the past 20 years, there has been extensive research in applying computers to music. The results of this research are becoming more visible as digital techniques transform music production, music analysis, and music recording.

Computer Music Journal, published since 1977, is a respected source for information in this field. The **Journal** is distributed quarterly to universities, research institutes, and individual subscribers in 27 countries.

People's Computer Company occupies the second floor of a small office building in downtown Menlo Park. The 30 staff members, more than half of whom are part-time, have professional interests in education, art, musical composition, computer science and journalism.

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

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 21 | Box E, Menlo Park, CA 94025 | Volume 3, Issue 1

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Heath's H-8

The Heath H-8 "Evaluation": I Think We Blew It! – *Jim C. Warren, Jr., Editor* 3

Consarn That H-8! – *Craig A. Pearce* 5
a tongue-in-cheek scolding of DDJ

6502 Stuff

Memory Test for 6502 – *Carl Moser* 4

Challenging Challenger's ROMs – *Stephen P. Smith* 6

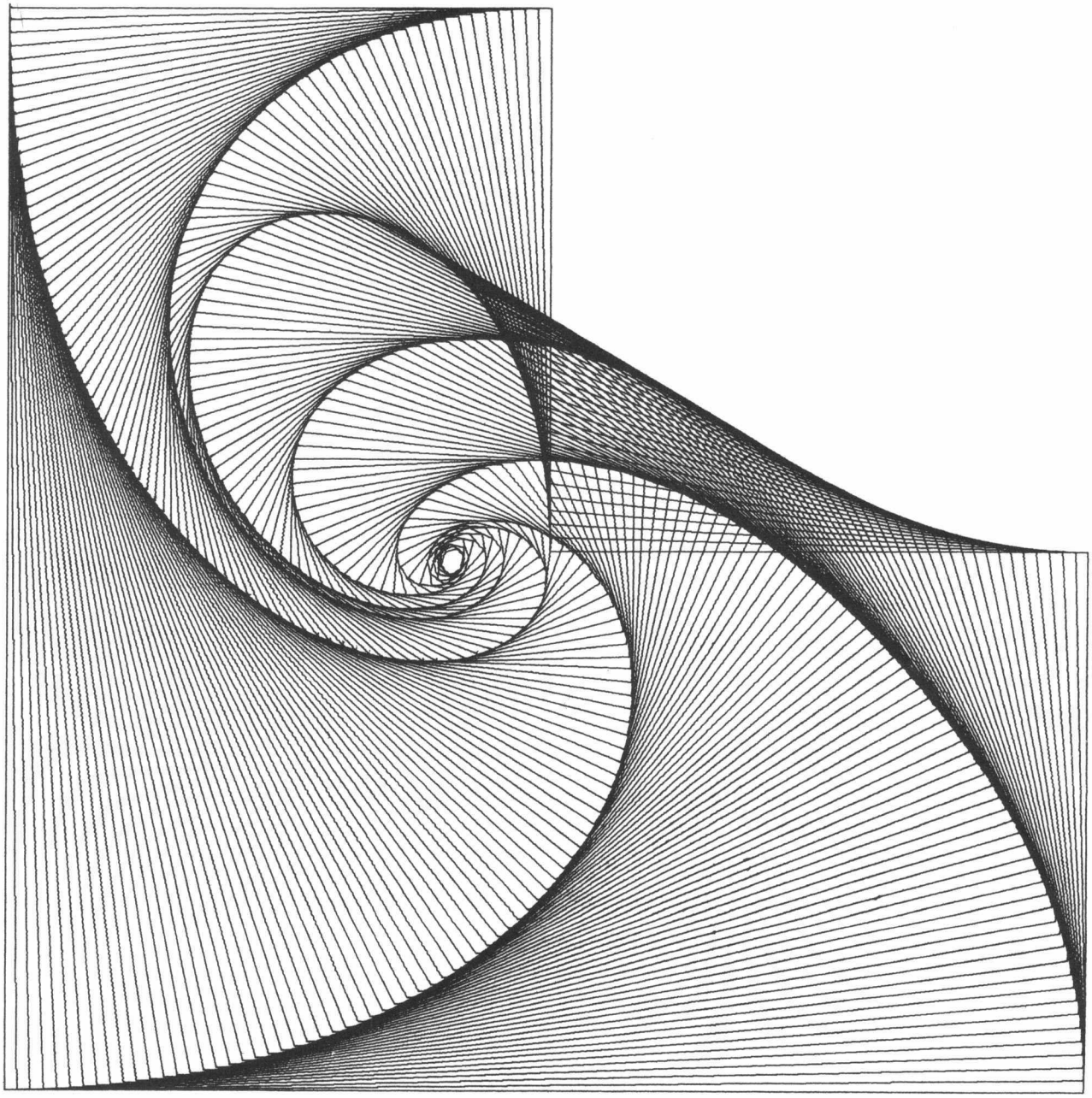
Software Design & Implementations

Is Your Assembly Language Program Well Structured? – *Webb Simmons* 8

ISIS: Anatomy of a Real-World Operating System – *Perry C. Hutchison* 10
an "internal logic manual" for Intel's ISIS

The SAM76 Language – *Ancelme Roichel* 18
complete documentation of a powerful string & macro processor

An 8080 Disassembler & Support Software for SOL – *Don Van Dyke* 38



© David Dameron 1979

THE HEATH H-8 "EVALUATION": I THINK WE BLEW IT!

Jim C. Warren, Jr., Editor, *DDJ*

In issue 19 of *Dr. Dobb's*, we published a rather negative evaluation (?) of Heath's H-8, along with a detailed response from Heath. The evaluation was written by a person we know personally whose technical judgment has proven to be fair and accurate in the past. At his request, we withheld his identity from publication–an editorial policy that is quite common among periodicals . . . at least, regarding "Letters to the Editor."

In retrospect, it is beginning to appear that we were wrong on several counts.

(1). If we were going to publish an article, giving it the strength of being a *DDJ* "evaluation", then we should have identified the author. While it may be appropriate to withhold a writer's identity when publishing a *letter* to the editor, we have concluded that authors of *articles*–and certainly of product evaluations–should be identified, and we will do so in the future.

(2). We have been receiving additional feedback from other H-8 consumers, and have found it to be explicitly contradictory to a number of complaints in the original article we published. We have been publishing and will continue to publish such follow-up information.

(3). We currently have, in-house, an H-8 kit shipped to us by Heath for evaluation. Tom Williams, our Associate Editor, is currently in the process of constructing it and has already found a number of points of disagreement with the original negative evaluation. As soon as he finishes the construction and testing of the kit, we will be publishing *his* evaluation of the kit, as an *evaluation* performed by a *Dr. Dobb's* Editor, in-house.

We would like to apologize to Heath and to our readers for having published what is beginning to appear to have been an inaccurate and poorly researched "evaluation" of the H-8. It is no more appropriate to publish an inaccurate negative article concerning a product, than it is to publish an inaccurate article that praises a product. Articles are written by individuals, and evaluations require judgment on the part of the individual doing the writing. In this case, it appears that the author of the article erred in significant aspects of his judgment, and that we erred in trusting his judgment. For this, we apologize.

RS-232 ADAPTER FOR KIM

News Release *Received: 77 Dec 20*

The ADApter converts KIM's 20 ma. current loop port to an RS-232 port. ADA does not affect the baud rate and uses standard power supplies. The unit comes complete with instructions, is assembled and tested, and measures 3" x 3½" x 1". KIM is isolated from the RS-232 device by optoisolators.

ADA sells for $24.50 with drilled, plated through solder pads for all connections, or for $29.50 with barrier strips and screw terminals. Connecticut Microcomputer, 150 Pocono Rd, Brookfield, CT 06804, (203) 775-9659.

UNSUPPORTED OPCODE PITFALLS

Dear Jim, *Received: 78 Jan 9*

Pardon my tardiness, but when I got around to the article on 650X Opcodes (August 1977) it set off a sensitive nerve. In my opinion, there are no words strong enough to convey an adequate warning against investing any significant time and effort in hardware or software that is either irreplaceable or not yours to use.

Before you use unsupported "instructions" in an MPU, before you buy that music board with the secret, proprietary, irreplaceable kludge chip, before you buy that proprietary computer language, ask yourself what you will do if your marbles get taken away. Suppose the chip blows? They do, you know. Suppose your supplier goes out of business? Suppose he merely multiplies the annual support/royalty charges by ten?

My advice is to do what knowledgeable people in the electronics industry have usually done: unless it's in the nature of a throwaway, don't even think about buying it or renting it or using it unless you have a clear alternative if your supplier starts to behave unacceptably, or ceases to exist. People who disobey that injunction have been badly burned in the past, and will be badly burned in the future.

So have fun with the unsupported opcodes, marvel at the secret kludge chips, and read about the latest super-gizmo proprietary packages. But don't tie yourself too heavily to any of them.

Cordially yours,
Paul Schick

730 Witte Hall
Madison, Wisc. 53706

ERROR ON MOSTEK ERRATA

Dear Mr. Warren: *Received: 78 Jan 10*

I recently received the October 1977 issue of *Dr. Dobb's* and have found a letter entitled "MOSTEK/Zilog Flag Flap" which is in error and could mislead your readers.

MOSTEK is a fully licensed second source of the Z80 chip set (CPU, PIO, CTC, CMA and SIO) designed by Zilog. As such, MOSTEK fabricates the Z80 chip set from masks designed by Zilog with no problems.

Reader Caldwell's confusion stems from an error in our first printing of the MOSTEK Z80 Programming Manual. An ERRATA sheet has been added to successive printings of the manual to correct the errors and available to anyone who contacts us regarding this matter.

Both the MOSTEK and Zilog Z80-CPU devices set the carry flag when subtraction generates a borrow. This is also compatible with the operation of the 8080A as the 8080A instruction set is a subset of the Z80 instruction set.

MOSTEK is concerned that Caldwell's letter in your publication will mislead your readers. Therefore, we encourage you to print this letter and the enclosed ERRATA sheet in your next issue. If any of your readers have questions regarding this matter, feel free to contact me or any of our Field Applications Engineers.

Regards,
James F. Vittera
Applications Engineering

MOSTEK Corporation
P.O. Box 169
Carrollton, TX 75006

MEMORY TEST FOR 6502

Dear Mr. Warren: *Received: 78 Jan 3*

Enclosed is a source listing of a program which tests RAM memory in 6502 based systems. I wrote this test after using other memory test programs which did not perform a complete test. The problem areas were untested chip selects, and address line inputs. This program written in my assembler's source syntax (advertised in Vol.2 No. 8 of *DDJ*), performs an exhaustive test of a user specified range of memory. The I/O is arranged for 6502 TIM based systems but can be easily changed.

The program performs two tests:

Test 1: Tests all memory cells for storage retention, and shorted data and address lines. This is done by writing: 00 11 . . . FF 00 11 . . . FF continually throughout the memory range for the first pass. When this has been written it is checked to validate the data. On the next pass 01 02 . . . FF 00 01 . . . FF is written and checked. This continues for 256 (hex FF) passes until all possible combinations of bit patterns have been used.

Test 2: Tests the RAM chip select inputs. This is the same as test 1 except data 00 01 . . . F2 00 01 . . . F2 is used.

Before execution, enter the address of the start of the memory to be tested in 00C0 (lo) and 00C1 (hi), and the end of memory +1 in 00C2 (lo) and 00C3 (hi).

Execution begins at 0000 with the first test.

If an error occurs, it will be outputted in the following form:

Address	Test Pattern	Error
xxxx	yy	zz

When test 1 runs to completion, a break will occur and register A will contain E1 (signifying end of test 1). To execute test 2, simply continue execution (type G to TIM monitor).

If errors occur, they will be of the same form as described above. When test 1 has run to completion, a break will occur and register A will contain E2 signifying the end. To continue execution at test 1, simply type G to the TIM monitor. The address range entered in 00C0–00C3 is not altered by the memory test program.

If errors occurred in test 2 but not in test 1, you can safely assume a chip select malfunction (defective or not connected). Usually a number of errors will occur in test 1 when the fault is a single defective address input, data input, or data output.

If a continuous sequence of addresses with errors occur, the problem is likely to be an open data input, or stuck at '1' or '0' data output.

If every 2nd, 4th, 8th, 16th or some power of 2 address sequence with errors occur, check for defective address inputs.

If you are checking say 2102's (1 x 1K) and are specifying a range of, say 4K of memory, and an error common to the whole range occurs, the problem is likely to be in the power leads, defective data or address buffers, stuck at '0' address inputs, stuck at '0' data inputs, or stuck at '0' data outputs.

In all of the above, you may have to examine the various memory error patterns for some similarity in order to isolate the defective component. This is especially true of the 1 x 1K 2102 memory chip where each chip is devoted to a particular data lead ($D_0 - D_7$).

Finally, since this program has a copyright assigned, I waive this right for publication in *DDJ* only.

Sincerely,
Carl Moser

3239 Linda Dr.
Winston-Salem, NC 27106

```
                        0001            .OS
                        0002 :
                        0003 :*****COPYRIGHT 1977 BY CARL MOSER*****
                        0004 :*****     ALL RIGHTS RESERVED.     *****
                        0005 :
                        0006 :        MOS 6502 MEMORY TEST
                        0007 :            PROGRAM
                        0008 :
                        0009 :USER ENTERS MEMORY TO BE TESTED STARTING
                        0010 :ADDRESS IN START. AND END ADDRESS IN END.
                        0011 :
                        0012 :
                        0013 :THESE STORAGE VARIABLES ARE EXTERNAL IN ZERO PAGE.
                        0014 START      .DE $C0  USER ENTERS START OF MEM. TO BE TESTED
                        0015 END        .DE $C2  USER ENTERS END OF MEM. TO BE TESTED +1
                        0016 ADDRS      .DE $C4
                        0017 TEST.TYPE  .DE $C6
                        0018 TEST.PATRN .DE $C7
                        0019 :
                        0020 :
                        0021 :ROM I/O CALLS
                        0022 TRYT       .DE $72B1  OUTPUT A PACKED BYTE (TWO DIGITS)
                        0023 SPACE?     .DE $7374  OUTPUT TWO SPACES
                        0024 CRLF       .DE $72BA  OUTPUT A CARRIAGE RETURN. LINE FEED
                        0025 :
                        0026 :
                        0027            .BA $0
                        0028 :
0000  A200     0029 MEM.TEST   LDX #00
0002  8EC600   0030            STX TEST.TYPE  TEST 1
0005  201800   0031            JSR TEST.PGM
0008  A9E1     0032            LDA #E1
000A  02       0033            BRK
000B  EA       0034            NOP
000C  EA       0035            NOP
000D  EEC600   0036            INC TEST.TYPE  TEST 2
0010  201800   0037            JSR TEST.PGM
0013  A9E2     0038            LDA #E2
0015  00       0039            BRK
0016  EA       0040            NOP
0017  EA       0041            NOP
0018  4C0000   0042            JMP MEM.TEST
                0043 :
001B  20BA72   0044 TEST.PGM   JSR CRLF
001E  A000     0045            LDY #00  PATTERN REGISTER
0020  A200     0046            LDX #00
0022  8EC700   0047            STX TEST.PATRN
0025  4C2E00   0048            JMP NX.PASS
                0049 :
0028  EEC700   0050 NX.PATRN   INC TEST.PATRN
002B  D001     0051            BNE NX.PASS
002D  60       0052            RTS
002E  ADC700   0053 NX.PASS    LDY TEST.PATRN
0031  208500   0054            JSR INI.ADDRS
0034  98       0055 LOOP1      TYA
0035  81C4     0056            STA (ADDRS.X)  STORE PATTERN
0037  C1C4     0057            CMP (ADDRS.X)  CHECK
0039  F003     0058            BEQ NO.ERR1
003B  208500   0059            JSR ERROR  ADDRS. R(A). (ADDRS.X)
003E  206E00   0060 NO.ERR1    JSR INC.ADDRSC
0041  F006     0061            BEQ CK.PATRN
0043  206100   0062            JSR INC.RY
0046  4C3400   0063            JMP LOOP1
                0064 :
0049  ADC700   0065 CK.PATRN   LDY TEST.PATRN
004C  208500   0066            JSR INI.ADDRS  INITIALIZE ADDRS
004F  98       0067 LOOP2      TYA
0050  C1C4     0068            CMP (ADDRS.X)
0052  F003     0069            BEQ NO.ERR2
0054  208500   0070            JSR ERROR  ADDRS. R(A). (ADDRS.X)
0057  206100   0071 NO.ERR2    JSR INC.RY
005A  206E00   0072            JSR INC.ADDRSC
005D  D0F0     0073            BNE LOOP2
005F  F0C7     0074            BEQ NX.PATRN
                0075 :
                0076 :
0061  C8       0077 INC.RY     INY
0062  ADC600   0078            LDA TEST.TYPE
0065  F006     0079            BEQ EXIT1
0067  C0F3     0080            CPY #F3  RESET R(Y) TO CHECK CHIP SELECTS
0069  9002     0081            BCC EXIT1
006B  A000     0082            LDY #00
006D  60       0083 EXIT1      RTS
                0084 :
                0085 :
006E  EEC400   0086 INC.ADDRSC INC ADDRS
0071  D003     0087            BNE SKIP.HI
0073  EEC500   0088            INC ADDRS+$01
0076  ADC200   0089 SKIP.HI    LDA END
0079  CDC400   0090            CMP ADDRS
007C  D006     0091            BNE EXIT2
007E  ADC300   0092            LDA END+$01
0081  CDC500   0093            CMP ADDRS+$01
0084  60       0094 EXIT2      RTS
                0095 :
                0096 :
                0097 :OUTPUT THE ERROR: ADDRESS. PATTERN. ERROR
0085  48       0098 ERROR      PHA
0086  ADC500   0099            LDA ADDRS+$01
0089  20B172   0100            JSR TRYT  OUTPUT ADDRS HI
008C  ADC400   0101            LDA ADDRS
008F  20B172   0102            JSR TRYT  OUTPUT ADDRS LO
0092  207473   0103            JSR SPACE?
0095  68       0104            PLA
0096  20B172   0105            JSR TRYT  OUTPUT PATTERN
0099  207473   0106            JSR SPACE?
009C  A1C4     0107            LDA (ADDRS.X)
009E  20B172   0108            JSR TRYT  OUTPUT ERROR IN MEMORY
00A1  20BA72   0109            JSR CRLF
00A4  60       0110            RTS
                0111 :
                0112 :
                0113 :INITIALIZE ADDRS WITH START
00A5  ADC000   0114 INI.ADDRS  LDA START
00A8  8DC400   0115            STA ADDRS
00AB  ADC100   0116            LDA START+$01
00AE  8DC500   0117            STA ADDRS+$01
00B1  60       0118            RTS
                0119 :
                0120 :
```

CONSARN THAT H-8!

Dear Sirs: *Received: 78 Jan 12*

I truly enjoyed the article by . . . hmmm . . . let's see . . . no name given, eh? Well, anyway, the article by (must be) Dr. Dobbs (or maybe Dr. Duds) himself on the great review of the Heath H8 microcomputer. Fantastic! What a swell job of reviewing. The author really got to the point. You tell 'em Doc.

The H8 HAS to be the worst 8080A microcomputer out. Mine, running with 24K at the moment, actually runs correctly when switched on! No self respecting home computer should do that. It even had the audacity to come up running immediately after I completed building it! What will all the tinkerers do with a system that actually *runs*?

Unlike the original S-100 bus system I first built, the H8 can't destroy memory contents when a reset is performed. The damn thing keeps preserving what is in RAM. Furthermore, if you can believe it, this poor machine doesn't even let the novice write his own bootstrap program to use the cassette interface . . . one is already in ROM ready to use for loading or dumping. Some spoil sports these Heath people are, aren't they?

That angle rack mounting is really bad. Here, the boards are rigidly secured on top, bottom and side, making this excuse for a computer simple to transport, rather than having the fun of stuffing newspaper or packing 'pop-corn' in so the boards don't snap off like some S-100 Systems. Since the mounting bracket on each board also acts as a heat sink, the fact that they are attached to the computer main frame can dissipate the heat even better, and thus, you don't have the fun of leaving the top off the computer like on other systems, nor the sheer pleasure of the fan constantly running in your ears, since none is needed.

I was able to fix the problem of not enough stuff on the boards that Doc brings up too. (After all, what fun is having a circuit board made simple, with fewer parts to go wrong, when a jumbled up board, taking three times as long to build is much more impressive looking . . . even if it doesn't work.) I just glued extra ICs on the board indiscriminately to make things more complex!

At least ol' Doc, the author of this H8 review, had enough sense to leave off the metal clip supplied with the H8 (to be placed by the keys mounted on the circuit board) so that his board bends as the keys are hit. I, like a fool, followed the Heath directions and put the clip on, which totally avoided the problem of the bending board Doc points out in his article. As soon as I finish this letter, I intend to immediately run up and pull the clip off my machine so I can bend my front panel board too!

Oh yes, and then there is that God awful OCTAL! My Lord, what a sin! It takes someone almost 10 minutes to get used to a new numbering system. I wish I were as lucky as Doc, and have 8 thumbs on each hand so I could use Hexadecimal too.

The fact that you can monitor, both dynamically, as well as statically any register, memory location or I/O port via the front panel too is also a hardship. I'd much rather have a string of LED lights to view and interpret. Or better yet, *no* front panel at all, like the SOL or APPLE II. Where's the *challenge* in seeing what you're doing?

I guess what's really horrible, as Doc points out, is the fact that there is no compatibility with existing S-100 cards. No one is going to make H8 compatible cards (excusing the already announced Godbout 12K RAM card, or some other manufacturers who have also requested to manufacture H8 compatible cards). I mean, outside of them, there's only Heath.

By all means, Heath has really gone too far with the H8. Swelled heads and too big for their britches, as Doc says. They actually had the nerve to offer the typical hobbyist and home user a microcomputer that actually works, has a functional and useful front panel, compatible software, memory boards that hold their contents rather than dropping bits, I/O boards that operate correctly the first time they are built, solid construction and a design that isn't a rehash of all the other hardware on the market today. Damn innovaters . . . next thing you know, someone may offer documentation that isn't 10 mimeographed sheets (like a lot of other systems) and give full support like the H8 documentation.

Do your readers know that the Heath people are actually *giving* you software in the form of Basic, console debug, text editor and assembler *with* the computer to get the user going faster? Where's that good old "We'll get 'em on the accessories!" rule that has become a standard in the industry? The people at Heath even returned my phone calls on some questions I had about the system; with right answers even! Now what kind of computer manufacturer is that, that actually RETURNS the users' calls, with correct answers to boot? No one else that I can think of.

And, I can see Doc's dislike of the fact that only the Heath part numbers are given on the schematics. Why make it easy for the non-technician to order parts through Heath when needed? Besides, I guess Doc has the same problem I have of not knowing how to use the cross reference table in the manual (each has one), which gives the user the standard, industrial part number. Hell, you have to know how to match up numbers to use a table like that!

Finally, I have personally gained some special knowledge in writing styles. I am the author of the Microscene column in *65-Notes*, a newsletter put out by the *HP-65 Users Club* (not affiliated with Hewlett-Packard), which is read by some 1900 members. When I ran a poor review of a micro, I was actually dumb enough to sign my name. I've learned now. This may well be my last time I'll sign any of my work that might bring me criticism.

Let's hope that Heath can do better next time, in producing a piece of hardware that constantly requires revisions like the other micro manufacturers, (with their various patches and fixes). And, by all means, Mr. Warren, Jr., keep a high regard for the author of this fine review, ol Doc Duds. He's really the best.

Oh, one more thing, Mr. Warren . . . if you should find this little note of some interest, please print it in the letters column, rather than give it to "Doc Duds". At least those who subscribe to your magazine can read.

Sincerely,
Mr. Craig A. Pearce

2529 S. Home Avenue
Berwyn, Illinois 60402

Conclusion of "Memory Test for the 6502"

```
LABEL FILE:  [ / = EXTERNAL ]

/START=00C0          /END=00C2          /ADDRS=00C4
/TEST+TYPE=00C6      /TEST+PATR N=00C7  /TRYT=7291
/SPACE2=7374         /CRLF=7298         MEM+TEST=0000
TEST+PGM=001A        NX+PATRN=0029      NX+PASS=002E
LOOP1=0034           NO+ERR1=003F       CK+PATRN=0049
LOOP2=004F           NO+ERR2=0057       INC+RY=0061
EXIT1=006D           INC+ADDRSC=006E    SKIP+HI=0076
EXIT2=0094           ERROR=0085         INI+ADDRS=00A5
END+PGM=0092
//0000
```

CHALLENGING CHALLENGER's ROMS

Dear Friends, *Received: 77 Dec 28*

I have just become the proud owner of an Ohio Scientific Challenger with Basic in 8K of ROM and a monitor in ½K of PROM. While I am delighted with its performance, I'm not happy about the lack of documentation provided for the read only memory. I'd like to use the utility routines which must be buried there.

I tried hand decoding memory dumps, but that's a laborious task. Then it occured to me to let Basic do the work for me. Using the PEEK function, I wrote a disassembler to convert the stored bytes into MOS Technology cross assembler mnemonics and decimal operands. The monitor decoded easily, revealing a basket of I/O goodies. The Basic ROMs are yielding more slowly. They apparently contain a number of tables which must be picked around carefully.

I've enclosed a copy of my program so that other 6502 users can reap the same benefits. It's written, of course, in OSI's Microsoft Basic, but should be adaptable for PET and Apple. I haven't time to pursue refinements such as hex conversion for addresses, recognition of ASCII characters, or automatic skips around tables. If another reader makes these or other additions, I'd appreciate them sending me a copy of the program.

Sincerely yours,
Stephen P. Smith

P.O. Box 841
Parksley, VA 23421

GOODIES FROM THE GOVERNMENT??

Dear Dr. Dobbs, *Received: 77 Dec 29*

Your readers might be interested in a government sponsored software library. The National Aeronautics and Space Administration (NASA) puts out a quarterly journal "Computer Program Abstracts" which can be ordered from the Superintendent of Documents, U.S. Government Printing Office, Washington DC, 20402 for $3.30/yr.

Be aware that most of the programs are of little use to the average home computer freak. Not too many need a 4,062 line FORTRAN IV program for $480 to "calculate flow fields in supersonic inlets". But useful items do surface; like a 79 line BASIC program for "organizing and managing a collection of articles dealing with a specific keyword or subject"; program and documentation for $25. The programs are written for everything from IBM 370's to HP-25's. Costs range from $15 to $2,500 with separate documentation running at 5 to 10 percent of the program price. If the documentation meets the usual government specifications, it will be very complete.

To sum up, the prices for this software are such that if you find what you need here and you have a market for its services, the government software would be a bargain.

Ronald N. Orr
1114 Harper Ave
Redondo Beach, CA 90278

```
LIST

 10 PRINT "DISASSEMBLER FOR MOS TECHNOLOGY 650X"
 20 PRINT "BY STEPHEN P. SMITH, PARKSLEY, VA
 21 PRINT "VERSION 1.0, DECEMBER 13,1977
 22 PRINT
 30 DIM N$(56),A(256),L$(3),T$(5)
 40 DATA ADC,AND,ASL,BCC,BCS,BEQ,BIT,BMI,BNE,BPL,BRK,BVC,BVS,CLC,CLD
 50 DATA CLI,CLV,CMP,CPX,CPY,DEC,DEX,DEY,EOR,INC,INX,INY,JMP,JSR,LDA
 60 DATA LDX,LDY,LSR,NOP,ORA,PHA,PHP,PLA,PLP,ROL,RTI,RTS,SBC,SEC,SED
 70 DATA SEI,STA,STX,STY,TAX,TAY,TSX,TXA,TXS,TYA,ILLEGAL
 80 FOR I=1 TO 56
 90 READ N$(I)
100 NEXT I
110 L$(0)=" "
120 L$(1)="#"
130 L$(2)="("
140 L$(3)="  A"
150 T$(0)=" "
160 T$(1)=",X)"
170 T$(2)="),Y"
180 T$(3)=",X"
190 T$(4)=",Y"
200 T$(5)=")"
201 REM
202 REM EACH POSSIBLE WORD IS ASSIGNED A FIVE DIGIT CODE
203 REM THE FIRST TWO DIGITS GIVE THE APPLICABLE MNEMONIC
204 REM THE THIRD GIVES THE TOTAL NUMBER OF BYTES IN THE INSTRUCTION
205 REM THE FORTH INDICATES WHICH OF 4 LEADING FIGURES IS TO BE PRINTED
207 REM THE FIFTH  INDICATES WHICH OF 6 TRAILING FIGURES IS USED
208 REM
210 DATA 11100,35221,56100,56100,56100,35200, 3200,56100
220 DATA 37100,35210, 3130,56100,56100,35300, 3300,56100
230 DATA 10200,35222,56100,56100,56100,35203,56100,56100
240 DATA 14100,35304, 3203,56100,56100,35303, 3303,56100
250 DATA 29300, 2221,56100,56100, 7200, 2200,56100,56100
260 DATA 39100, 2210,56100,56100, 7300, 2300,56100,56100
270 DATA  8200, 2222,56100,56100,56100, 2203,56100,56100
280 DATA 44100, 2304,56100,56100,56100, 2303,56100,56100
290 DATA 41100,24221,56100,56100,56100,24200,33200,56100
300 DATA 36100,24210,33130,56100,28300,24300,56100,56100
310 DATA 12200,24222,56100,56100,56100,24203,33300,56100
320 DATA 16100,24304,33203,56100,56100,24303,33300,56100
330 DATA 42100, 1221,56100,56100,56100, 1200,56100,56100
340 DATA 38100, 1210,56100,56100,28325, 1300,56100,56100
350 DATA 13200, 1222,56100,56100,56100, 1203,56100,56100
360 DATA 46100, 1304,56100,56100,56100, 1303,56100,56100
370 DATA 56100,47221,56100,56100,49200,47200,48200,56100
380 DATA 23100,47210,53100,56100,49303,47300,48300,56100
390 DATA  4200,47222,56100,56100,49200,47203,48204,56100
400 DATA 55100,47304,54100,56100,56100,47303,56100,56100
410 DATA 32210,30221,31210,56100,32200,30200,31200,56100
420 DATA 51100,30210,50100,56100,32300,30300,31300,56100
430 DATA  5200,30222,56100,56100,32203,30203,31204,56100
440 DATA 17100,30304,52100,56100,32303,30303,31304,56100
450 DATA 20210,18221,56100,56100,20200,18200,21200,56100
460 DATA 27100,18210,22100,56100,20300,18300,21300,56100
470 DATA  9200,18222,56100,56100,56100,18203,21203,56100
480 DATA 15100,18304,56100,56100,56100,18303,21303,56100
490 DATA 19210,43221,56100,56100,19200,43200,25200,56100
500 DATA 26100,43210,34100,56100,19300,43300,25300,56100
510 DATA 06200,43222,56100,56100,56100,43203,25203,56100
520 DATA 45100,43304,56100,56100,56100,43302,25303,56100
530 FOR I=1 TO 256
540 READ A(I)
550 NEXT I
560 PRINT "ENTER STARTING AND ENDING ADDRESSES"
570 INPUT S,E
580 REM BEGIN DISASSEMBLY
600 I=PEEK(S)
605 REM LOCATE AND DISMEMBER THE 5 DIGIT CODE
610 C=A(I+1)
620 M=INT(C/1000)
630 IF M=56 THEN 1000
631 IF M<1 THEN 1000
640 B=INT(C/100)-M*10
650 P=INT(C/10)-M*100-B*10
660 Q=C-M*1000-B*100-P*10
670 IF B=1 THEN 800
675 REM SECOND BYTE OF INSTRUCTION
680 D1=PEEK(S+1)
690 IF B=2 THEN 800
695 REM THIRD BYTE OF INSTRUCTION
700 D2=PEEK(S+2)
710 D1=D1+D2*256
720 REM SPECIAL TREATMENT IF BRANCH
800 IF M<4 THEN 900
810 IF M>13 THEN 900
820 IF D1<127 THEN 900
830 D1=-(254-D1)
900 PRINT S;TAB(10);N$(M);TAB(16);L$(P);
902 IF D1=-123456 THEN 908
904 PRINT D1;T$(Q);
908 D1=-123456
909 PRINT
910 S=S+B
911 REM NEXT INSTRUCTION
920 IF S<E THEN 600
930 PRINT TAB(30),"**********"
940 STOP
1000 PRINT "AT";S;" ";I;" IS NOT A VALID OPCODE"
1010 PRINT "TERMINATING DISASSEMBLY"
1020 END
OK
RUN
DISASSEMBLER FOR MOS TECHNOLOGY 650X
BY STEPHEN P. SMITH, PARKSLEY, VA
VERSION 1.0, DECEMBER 13,1977

ENTER STARTING AND ENDING ADDRESSES
? 65024,65047
 65024     LDA      64512
 65027     LSR      A
 65028     BCC     -4
 65030     LDA      64513
 65033     AND    # 127
 65035     PHA
 65036     LDA      64512
 65039     LSR      A
 65040     LSR      A
 65041     BCC     -5
 65043     PLA
 65044     STA      64513
                                          **********

BREAK IN  940
OK
```

CONSUMER DO's AND DON'T's FROM A DEALER

Dear Dr. Dobb's, *Received: 78 Jan 9*

I read with interest the article, "A Detailed Report on Product Delivery"in the last issue (Volume 2 Issue 9) of *DDJ*. I thought that a few comments and suggestions from a computer dealer (we operate a retail computer outlet as part of our business in Northern Florida) may be of interest to you and your readers.

1) If at all possible, purchase products directly from a local computer store in your area. Not only does this permit you to see the merchandise working before you buy it, but many times you can get it the very same day you decide you want it. Additionally, if you have problems, you will have somewhere to go. Unfortunately, many manufacturers give you the instant "well, you'd better send it back" routine (ie, 2–8 weeks delay) when you phone them.
2) If at all possible, DO NOT send the full amount in advance for an item (ESPECIALLY for a new item)! In my opinion, the MOST that a firm should ask in advance is 25%. COD can be somewhat of a fail-safe device, but unfortunately, not all firms will ship on a COD basis. When a firm refuses to ship COD, it makes me wonder . . . however, a non-COD policy does not mean that a firm is not honest. COD seems to be the best way to go when purchasing a newly introduced item from a new company.
3) If it sounds too good to be true in the ad, it probably is (too good to be true). BEWARE!! Just because there is a photo of an item in an advertisement, that in no way means that that item is being produced or has ever even existed! The big ad does not make a firm reputable.
4) When ordering by mail, request that the firm immediately confirm receipt of your order by mail (you might even include a postage-paid postcard for the company to return to you) and also request an expected shipping date. This eliminates the "we never received your order routine" that could crop up on a long overdue shipment. You might also want to invest in a long distance phone call and inquire as to delivery times BEFORE you place an order and send money.
5) If a firm is promising and not delivering (especially if they have your money) do not hesitate to bug and bother them with letters and phone calls. Many times it is the person who yells the longest and loudest that gets what he's after. Do not hesitate to tell the overdue or non-producing firm that you are writing to several computer-related publications (ie, *Dr. Dobb's*, *Byte*, *Kilobaud*, *Interface Age*, etc.) and advising them of your plight. It won't make you a friend of the company you are dealing with, but then do you really want friends who take your money, promise, and not deliver?

One final comment: I find unacceptable the statements in the above mentioned article concerning the supposition that it takes too much paperwork to notify customers of order confirmations and delays. As far as I am concerned, it is an OBLIGATION of a firm to notify customers of any delay beyond a stated delivery date. That's not a public relations add-on, it is just good business.

Sincerely,
Bobby Williams

2062 Liberty Street
Jacksonville, FL 32206

S-100 PRO & CON CENSUS

Dr. Dobb's, *Received: 78 Jan 4*

Read with interest 'The Heath H-8: Pro & Con'. The Con author's first evaluation (?) point was that the H-8 is not S-100 compatible. My first negative point would be that it is not Z-80 based. As far as the bus structure not being S-100, well that is still an open question.

The Pro author's comments about the S-100 are just as emotional as the Con author's criticism of the H-8. And all along I thought this hobby was basically logical (1's and 0's and that sort of thing). Someone even told me that programming forces one to disciplined thinking.

My main rebuttal to the Pro author is that "you will find that over 95% of them (S-100 boards) are memory cards of some configuration." My count was in decimal–not octal or hex, so the following could be inaccurate.

From R.E. Purser's list of S-100 Bus Compatible Boards/ Systems, which has been reprinted in *Interface Age* and *Creative Computing*:

Being late in the 'eve, my count is:

RAM memory boards	56
PROM memory boards	33
Total	89
Serial & Parallel I/O Boards	32
Total of both	121

95% of 200 (a rounded figure) is 190 memory boards. Now my count is not a "critical look"–just a midnight counting.

Regarding compatibility of the S-100, the Pro author is right, there isn't much beyond the basically defined pins. Since no one knows how much incompatibility exists, I decided to find out.

The results of my labor should appear in a future issue of *Kilobaud*. I would have given it to y'all, but I had to recover my postage costs.

Open question: If the majority of magazine issues are sold to subscribers, why are we the last to receive them (i.e. after the newstands)? Mailing in bulk by UPS to stores is no excuse. Give them to UPS a couple weeks *after* the subscription copies have been mailed.

Pax,
Bill Fuller
1018F SE 8th St
Grand Prairie, TX 75051

Bill, your complaint is just, but we simply don't have space to store the bulk issues. They do go out somewhat after the subscriptions are mailed but even at that we have no way of knowing how long the subscriber copies will spend in the Post Office's aging bins before they're finally delivered. – TRW

SOUTHERN CALIFORNIA DEALERS ASSOCIATION

News Release *Received: 78 Jan 3*

The Southern California Computer Dealers Association has been formed, as a non-profit corporation, covering the Santa Barbara to San Diego area. Membership is limited to bona-fide stores which sell microcomputers for personal, business, industrial, or educational use.

Meetings will offer a forum for the exchange of views and information about the industry. The association is setting up committees to investigate further activities and objectives such as publicity for microcomputers, professional standards, and dealer-vendor relationships.

For further information, contact Carl Burlin at (714) 956-7968, or Glenn Dollar at (213) 360-2171.

Is your assembly language program well structured?

by Webb Simmons
1559 Alcala Place
San Diego, CA 92111

Reprinted with permission from:
COMPUTE Newsletter, Vol 3,#10, pp 18-19
National Semiconductor Corp.

There are a few simple rules that can be applied to your assembly language program which will show whether your program satisfies the more important criteria to qualify it as well structured in the sense of the modern buzz words on the matter. This presentation is intended to be *practical*, whatever that means, rather than *theoretical*; therefore, I can make, and get away with, all sorts of idiotic statements.

The tools you will need are an extra program listing you can mark up and two colored pens or pencils, say red and green. These pens will be used to connect program branches, jumps, skips, and other transfers from the transfer instructions themselves to the points of reference, the destinations of the transfers. Here we must make a nice distinction between the direction of the transfers so thus the two colors, red and green.

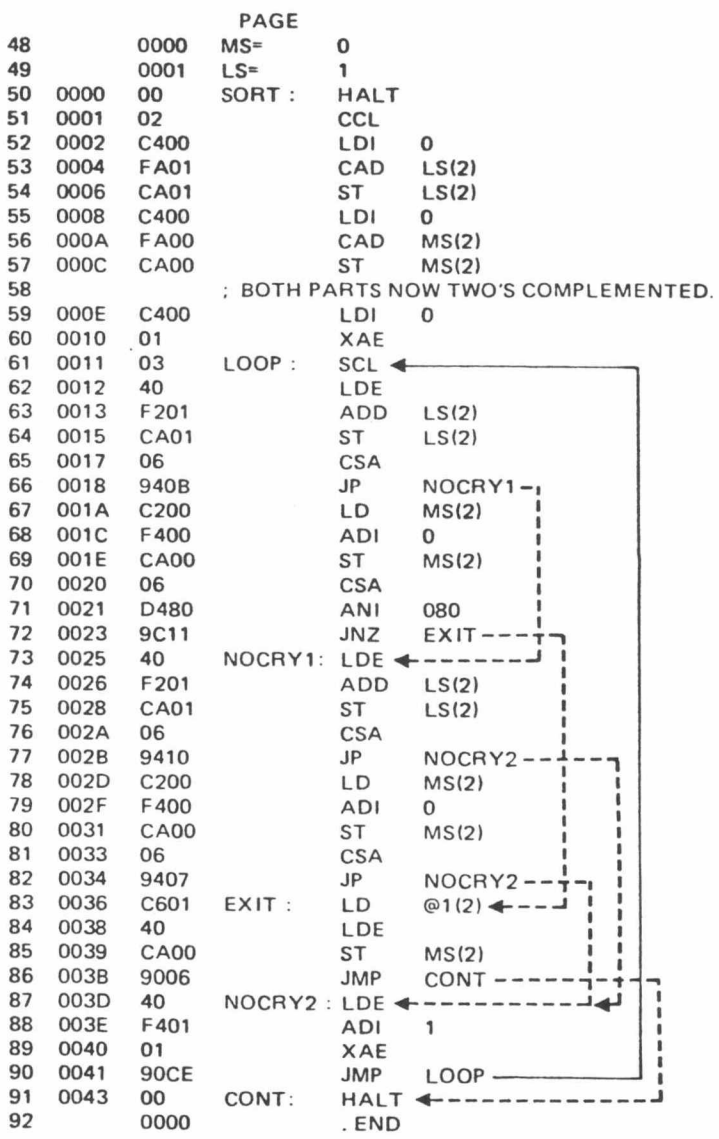
A forward reference is defined as a transfer from a lower counter location to a higher numbered location. And, of course, a backward reference is just the opposite, a jump or branch from a high program counter location to a lower numbered location. But before we start marking, a word must be said about subroutine jumps. Any subroutine call or subroutine jump that *always* returns to the point at which called (to the next executable instruction after the call) need not be considered to be a branch or transfer for the purposes of our colored markings. Fundamentally speaking, a *normal* subroutine call is merely a glorified, super macro instruction more so than a formal transfer of control in the sense that is important from a well structured standpoint. A subroutine with an *abnormal* return upon detected error conditions, etc., is not a normal subroutine so the jumps to it and its returns must be considered as transfers to be marked.

The first step is to use the red pen to mark all backward references along the side of the program listing from the jump instruction itself to the place jumped to. It does not matter but I prefer to make all of my markings on the left side of the listing. You should be able to draw red connecting lines from each backward jump to its destination with no need for one red line to cross another. Regardless of which side of the listing you choose to draw on, you must consistently use but *one side only*, left side or right side but not both sides. If you cannot draw the red connecting lines for backwards references *without* crossing one red line with another, you are already down the tubes, your program is not well structured.

After passing the first test of no crossed red lines we are ready to mark the forward references with green connecting lines. A green line may cross a red line only if it is outward bound. If your lines are on the left, a green line can cross a red line when going to the left but a green line cannot cross a red line when going to the right. Green lines can cross other green lines freely, there are no restrictions on green versus green. Under these conditions you must connect all forward transfer instructions with green lines to their destinations. If a green line must cross a red line while inbound, your program is not well structured.

What do you do if your program fails these tests? Does your program produce the correct results? Is it efficient in memory space utilization? Does it execute rapidly? These factors are interesting, but no matter; you must shoot yourself!

Since we don't have multi-colored printing, we are using a solid line in place of a red line and a broken line in place of a green line.



```
                    PAGE
48          0000    MS=     0
49          0001    LS=     1
50    0000  00      SORT :  HALT
51    0001  02              CCL
52    0002  C400            LDI   0
53    0004  FA01            CAD   LS(2)
54    0006  CA01            ST    LS(2)
55    0008  C400            LDI   0
56    000A  FA00            CAD   MS(2)
57    000C  CA00            ST    MS(2)
58                ; BOTH PARTS NOW TWO'S COMPLEMENTED.
59    000E  C400            LDI   0
60    0010  01              XAE
61    0011  03      LOOP :  SCL
62    0012  40              LDE
63    0013  F201            ADD   LS(2)
64    0015  CA01            ST    LS(2)
65    0017  06              CSA
66    0018  940B            JP    NOCRY1
67    001A  C200            LD    MS(2)
68    001C  F400            ADI   0
69    001E  CA00            ST    MS(2)
70    0020  06              CSA
71    0021  D480            ANI   080
72    0023  9C11            JNZ   EXIT
73    0025  40      NOCRY1: LDE
74    0026  F201            ADD   LS(2)
75    0028  CA01            ST    LS(2)
76    002A  06              CSA
77    002B  9410            JP    NOCRY2
78    002D  C200            LD    MS(2)
79    002F  F400            ADI   0
80    0031  CA00            ST    MS(2)
81    0033  06              CSA
82    0034  9407            JP    NOCRY2
83    0036  C601    EXIT :  LD    @1(2)
84    0038  40              LDE
85    0039  CA00            ST    MS(2)
86    003B  9006            JMP   CONT
87    003D  40      NOCRY2 : LDE
88    003E  F401            ADI   1
89    0040  01              XAE
90    0041  90CE            JMP   LOOP
91    0043  00      CONT:   HALT
92          0000            . END
```

Conclusion: This program is well structured. The only crossing of the single red (solid) line by a green (broken) line is outbound, thus O.K.

Note: Lines placed on the right rather than my preferred left because of more space.

```
1                       ;
2                       . TITLE'CDB, 'REFLEX TEST'
3           0001        . =1
4           0000        CNTR1=00
5           0001        CNTR2=01
6           0002        CNTR3=02
7           0003        CNTR4=03
8           0004        DIG4=04
9           0005        DIG3=05
10          0006        DIG2=06
11          0007        DIG1=07
12          0008        DIG0=08
13          0009        DIGA=09
14          0080        COUNT1-080
15          0010        COUNT2=010
16                      ;
17                      ; START
18                      ;
19  0001 C470           START: LDI 070 ;SET P1
20  0003 35             XPAH 1
21  0004 C477           LDI 077 ; SET P2
22  0006 36             XPAH 2
23  0007 C47A           LDI 07A ;SET P3
24  0009 37             XPAH 3
25  000A C46E           LDI 06E
26  000C 33             XPAL 3
27  000D C400           LDI 0
28  000F CA02           ST CNTR3(2)
29  0011 CA03           ST CNTR4(2)
30  0013 CA08           ST DIGO(2)
31  0015 CA09           ST DIGA(2)
32  0017 C420           LDI 020 ;SET FIRST DASH
33                      ;
34                      ; DASH
35                      ;
36  0019 01             DASH: XAE
37  001A C410           LDI COUNT2 ;SET CNTR 2
38  001C CA01           ST CNTR2(2)
39                      ;
40                      ; LOOP2
41                      ;
42  001E C480           LOOP2: LDI COUNT1 ;SET CNTR 1
43  0020 CA00           ST CNTR1(2)
44  0022 C440           LDI 040 ;FIRST DASH
45  0024 C980           ST -128(1)
46                      ;
47                      ; LOOP
48                      ;
49  0026 C11E           LOOP: LD 01E(1) ;GET KYBD
50  0028 D40F           ANI OF
51  002A 9C5D           JNZ ERROR
52  002C BA00           DLD CNTR1(2) ;DEC CNTR 1
53  002E 9CF6           JNZ LOOP
54  0030 BA01           DLD CNTR2(2) ;DEC CNTR 2
55  0032 9CEA           JNZ LOOP2
56  0034 01             XAE ;NEXT DASH
57  0035 1C             SR
58  0036 9CE1           JNZ DASH
59  0038 C900           ST (1) ;BLANK DISPLY
60                      ;
61                      ; COUNT
62                      ;
63  003A 03             COUNT: SCL ;DEC INC CNTR 3
64  003B C4AA           LDI 170 ;DELAY
65  003D 8F00           DLY 0
66  003F C202           LD CNTR3(2)
67  0041 EC00           DAI 0
68  0043 CA02           ST CNTR3(2)
69  0045 C203           LD CNTR4(2)
70  0047 EC00           DAI 0
71  0049 CA03           ST CNTR4(2)
72  004B C11E           LD 01E(1) ;GET KTBD
73  004D D40F           ANI OF ;MASK 4 MSB
74  004F 98E9           JZ COUNT
75                      ;
76                      ; CONVERT
77                      ;
78  0051 C203           CONVERT: LD CNTR4(2) ;GET MSD
79  0053 1C             SR
80  0054 1C             SR
81  0055 1C             SR
82  0056 1C             SR
83  0057 01             XAE
84  0058 C380           LD -128(3)
85  005A CA04           ST DIG4(2)
86  005C C203           LD CNTR4(2) ;GET MSD -1
87  005E D40F           ANI OF
88  0060 01             XAE
89  0061 C380           LD -128(3)
90  0063 CA05           ST DIG3(2)
91  0065 C202           LD CNTR3(2) ;GET LSD +1
92  0067 1C             SR
93  0068 1C             SR
94  0069 1C             SR
95  006A 1C             SR
96  006B 01             XAE
97  006C C380           LD -128(3)
98  006E CA06           ST DIG2(2)
99  0070 C202           LD CNTR3(2) ;GET LSD
100 0072 D40F           ANI OF
101 0074 01             XAE
102 0075 C380           LD -128(3)
103 0077 CA07           ST DIG1(2)
104                     ;
105                     ; DISPLY
106                     ;
107 0079 C404           DISPLY: LDI 04 ;SET P2 TO DIG 4
108 007B 32             XPAL 2
109 007C C420           LDI 020 ;SET TO LEFT DIGIT
110                     ;
111                     ;LOOP1
112                     ;
113 007E 01             LOOP1: XAE
114 007F C601           LD @1(2)
115 0081 C980           ST -128(1)
116 0083 01             XAE
117 0084 1C             SR
118 0085 9CF7           JNZ LOOP1
119 0087 90F0           JMP DISPLY
120                     ;
121                     ;ERROR
122                     ;
123 0089 C479           ERROR: LDI 079 ;SET ERROR MSG
124 008B CA04           ST DIG4(2)
125 008D C450           LDI 050
126 008F CA05           ST DIG3(2)
127 0091 CA06           ST DIG2(2)
128 0093 CA08           ST DIG0(2)
129 0095 C45C           LDI 05C
130 0097 CA07           ST DIG1(2)
131 0099 C400           LDI 0
132 009B CA09           ST DIGA(2)
133 009D 90DA           JMP DISPLY
134         0000        .END
```

OOPS!

Conclusions: This program is *not* well structured because the green (broken) line from line 51 to line 123 must cross one red (solid) line while inbound (near line 123).

The program would be well structured if the block from line 107 to line 119 were to be placed after line 133; however one would have to add the instruction "JMP DISPLY" after line 103 to replace the normal "FALL THROUGH" from line 103 to line 107. After moving the block the instruction at line 133 will not be needed because it will become a "fall through" and the red line will go away.

Honesty compels the observation that the program will work the same after moving the block of code as discussed–but–it will look better from a "well structured" standpoint.

ISIS: *Anatomy of a Real-World Operating System*

by Perry C. Hutchison
Computer Science Department
Iowa State University
Ames, IA 50010

Dear Jim: *Received: 77 Dec 23*

I am enclosing a sort of "internal logic manual" for Intel's ISIS system. This all started some months ago when I became the victim of a persistent "data CRC" error on the MDS which could not be reproduced on anything else. Several phone calls to Intel produced only the claim that, by the time the error message came out, the information as to what track and sector were involved was no longer available anywhere in the machine. This sounded rather preposterous, so I went a-hunting. About 2 weeks' work, spread over a considerably longer period, yielded the discoveries contained in this write-up. Since I don't suppose I am all that unique in needing to debug things in the context of ISIS, I decided to write it up for publication. It will probably be fairly interesting to 8080 (and Z-80) hackers, even if they don't have access to an MDS. The ISIS disk allocation and directory policies would make an excellent standard for floppy disk file systems.

*This article may also be useful as context for Max Agoston's RIMOS (*DDJ *9/77).*

Very truly yours,
Perry C. Hutchison

ISIS (Intel Systems Implementation Supervisor) is the floppy disk operating system developed by Intel Corporation for the Intellec MDS microcomputer system. ISIS was written entirely in PL/M, a dialect of PL/1 oriented to the 8080 microprocessor. The ISIS system is exceptionally well designed; it bears significant resemblance to Bell Labs' UNIX operating system for the PDP-11.

Intel has released neither the source code nor any internal documentation concerning ISIS. Since a knowledge of the internal workings of such systems is occasionally needed if they are to be utilized to the full extent of their capabilities, a study of the object code was undertaken. This document contains the principal findings of that study. It should be useful both as a reference for use by persons having occasion to deal with the internals of the ISIS system and as an educational example of what really goes on in a small single-user operating system. Names used are taken from the published ISIS documentation where appropriate.

The information contained herein has been derived from an examination of the object code of Version 1.2 of 32K ISIS, received in August, 1976. (The same disk also contains Version 1.1 of ASM80, 1.3 of EDIT, 1.1 of UPM, and 2.0 of ICE80.) Except where a more general applicability is specifically stated, this information should not be expected to apply to any other version of ISIS.

It is probable that, despite thorough checking, some errors will be found in this presentation. The author would appreciate being informed of them.

Environment

ISIS operates in the Intellec MDS microcomputer system, and uses the facilities of the MDS Monitor ROM (which occupies addresses F800H - FFFFH) to communicate with character-oriented peripheral devices (i.e. everything but the disk). Table I lists the entry points and functions of the principal Monitor routines. Parameters and returned values are handled as in PL/M – see the discussion below. (There are, of course, additional subroutines in the ROM, but an assembly listing of the ROM is included with the MDS and anyone interested in its internal workings can look it up. ISIS uses no Monitor entry points other than those listed in Table I.)

A detailed description of the MDS disk controller is contained in the Intel *MDS-DOS Hardware Reference Manual* and will be only briefly summarized here. The controller occupies I/O ports 78H - 7FH, and is controlled principally by a program-generated "I/O Parameter Block" (IOPB) in memory. The address of the IOPB is supplied to the controller by outputting its low and high bytes to output ports 79H and 7AH. The IOPB identifies the drive, the operation to be performed, the sector count, the track address, the starting sector address, and the starting memory address for reading or writing. The disk controller contains a DMA controller which is used for reading the IOPB as well as for transferring data.

Organization

ISIS consists of some 57 PL/M procedures, including 8 which merely provide access to the Monitor ROM. The remaining 49, plus two compiler-generated subroutines, are listed in Table II; Figures 1 and 2 depict the caller-called relationships. (Many of the routines of Fig. 1 call one or more routines of Fig. 2, but the spaghetti effect would render a combined drawing incomprehensible.) The heavy lines in Fig. 1 lead to routines which directly correspond to and implement the various ISIS system calls. (The procedures ERROR and XEQIOPB are actually Fig. 2 routines, but the direct calls to them from ISIS are shown in Fig. 1 for completeness.)

Table II should be largely self-explanatory, with the exception of a few abbreviations and notational conventions. The second column contains the Entry Point address of the procedure. The columns labeled "T" signify the Type of the associated quantity: A (Address) denotes a 16-bit value, B (Byte) an 8-bit value, and L (Logical) a Byte value used in a True-False sense. (An odd value is taken as True, an even value as False.) The column labeled "ADDR" is used for two related purposes: in the case of procedures which have parameters, it gives the memory address at which each parameter is stored; in the case of non-parametric procedures it gives the address at which the first of that procedure's local variables is stored. (Local variables of parametric procedures are stored immediately following the last parameter.) A dash in this column signifies that the procedure has no local variables and, except in the case of the compiler-generated subroutines, no parameters.

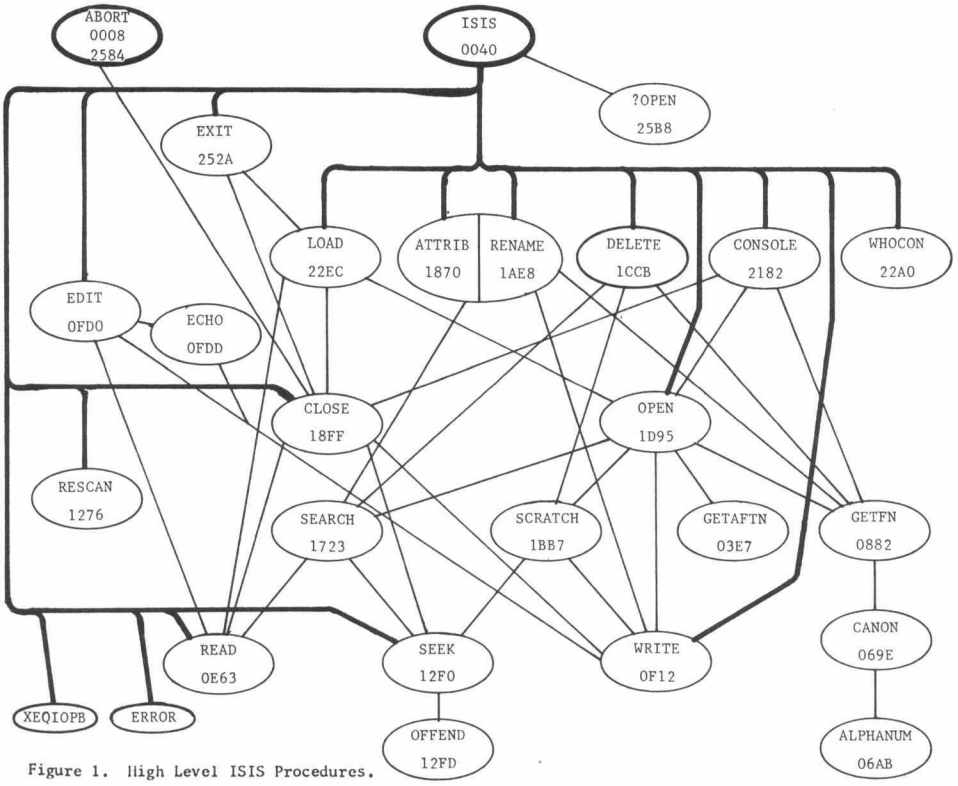


Figure 1. High Level ISIS Procedures.

The combining of ATTRIB and RENAME is a clutter-reducing notational convenience having no special significance. Both call the same three subroutines.

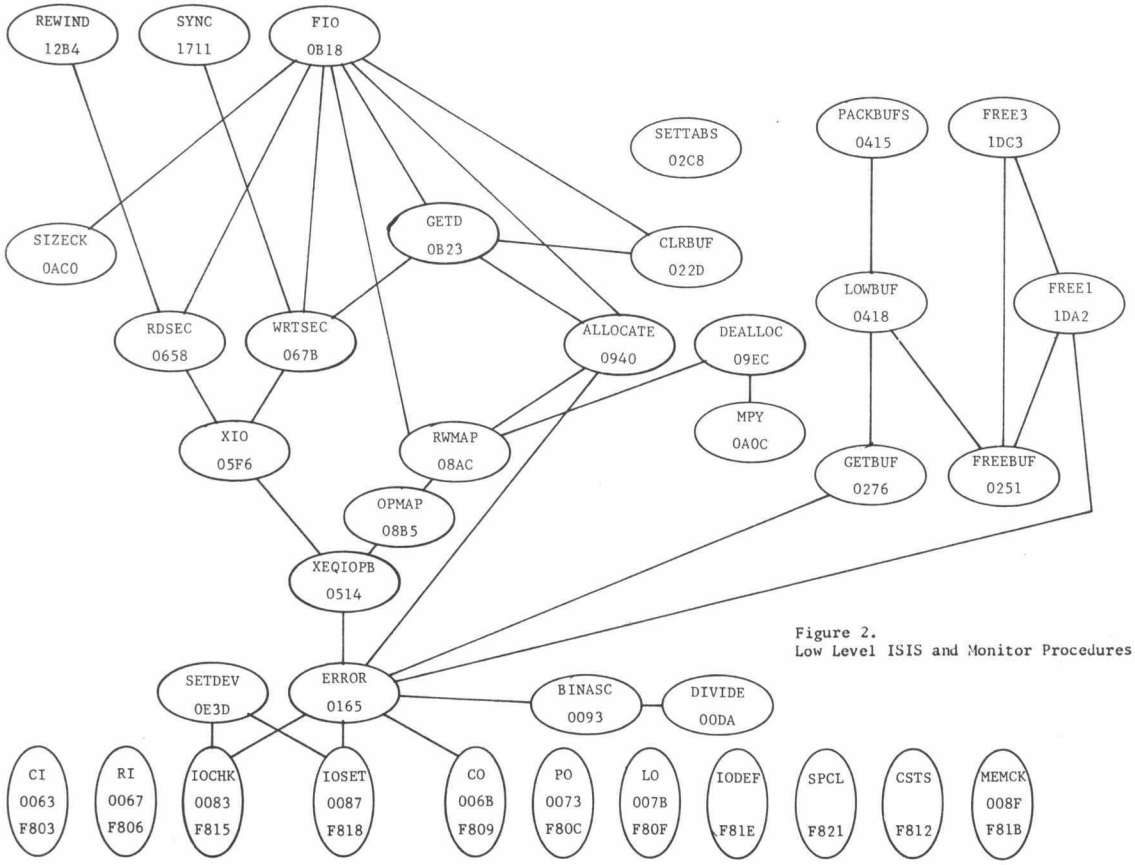


Figure 2.
Low Level ISIS and Monitor Procedures.

When one procedure is entirely contained within another, the table entry for the inner procedure immediately follows that for the outer one and the inner name is indented two spaces.

Data Structures

One of the principal motivations for this study has been the desire to obtain information of diagnostic usefulness. To that end, Table III presents a list of memory addresses whose contents may assist in diagnosing troubles. In addition, parameter values in active procedures may be useful. A traceback of procedure calls is fairly easily obtained from the stack, since entries other than return addresses are rare. The locations of the parameters may then be found from Table II.

The central in-memory data structure of ISIS is the File Structure, located in memory addresses 29E5 through 2B1B inclusive and summarized in Table IV. The File Structure consists of 18 tables, each containing 10 entries; each table has associated with it a "current pointer" which contains the address of the entry in that table which pertains to the current file (whose table index may, in turn, be found in 29E5). The first two entries (numbered 0 and 1) in each table always refer to the directories of disks 0 and 1; in a sense the directories are always open, although they do not always have buffers allocated. Entries 2 and 3 are the console files :CO: and :CI:. This leaves entries 4 through 9 for the six files which a user program may have open at any one time. Each open file corresponds to one entry in each table, but most of these entries are unused if the file is not on a disk.

The parameter AFT which many of the ISIS procedures require is simply a number in the range of 0 to 9 identifying a set of entries in the tables of the File Structure. This differs from the AFTN supplied by the user (and returned by OPEN) in that AFT = AFTN + 2.

The I.V. (Initial Value) shown in Table IV is the value which appears in the particular File Structure member for a disk file which has just been opened.

Disk Layout

The principles of the ISIS disk organization were rather well described by David Yulke in the December 1977 issue of *Kilobaud* (although he didn't mention ISIS by name), but will be included here for completeness and to provide additional details.

In ISIS, every disk sector falls into one of three groups: Data blocks, Linkage blocks, and Free blocks; this statement applies to system data as well as to user files. An ISIS file sonsists of Linkage blocks and Data blocks. Data blocks contain the bytes which compose the file, while Linkage blocks (Yulke calls them Map blocks) tell how the Data blocks are to be linked together. The first two bytes of a Linkage block contain the disk address of this file's preceding Linkage block (zero if this is the first). All disk addresses in ISIS are stored as a Block Number and a Track Number. Tracks are numbered from 0 to 76 and the blocks of each track are numbered from 1 to 26. The next two bytes contain the disk address of the following Linkage block. The remaining 124 bytes contain the disk addresses of up to 62 data blocks.

Every ISIS-format disk contains four system files, collectively referred to as the Format files. These may be opened for input and read just like any other file, but may never be written into (except by special system routines). Tables VI and VII list the Format files and describe their contents.

The system does not always bother to look the Format files up in the directory; it assumes that their locations are known. Thus the first Linkage block of the directory itself is always assumed to be Track 1 Block 1, and the Data blocks of the Allocation Bit Map are assumed to be Track 2 Blocks 2 and 3. The boot ROM expects to find the System Initialization Program starting at Track 0 Block 1 and occupying as much of Track 0 as may be needed; the System Initialization Program then assumes that the first Linkage block of the file ISIS.BIN (which contains the system proper) will be found at Track 2 Block 4.

Secret "XEQIOPB" System Call

The XEQIOPB system call is not described in the published ISIS documentation. Its "command" value is 68 decimal (44 hex); the "parameter block" must contain the following 3 words: 534BH, drive, .iopb (where drive is 0 or 1 and .iopb is the memory address of the IOPB to be executed). This is presumably intended to be used by programs like FORMAT which need to perform disk operations not needed by "normal" programs. The IOPB is to be set up as if it were to run on drive 0, regardless of which drive is specified; XEQIOPB takes care of inserting the drive identification in the proper places. Normal system handling of disk errors is provided. Note that this call bypasses all directory accessing, file mapping, and protection flags, and must therefore be used only with extreme caution. One possible use would be in a program which does a sector-for-sector copy of a disk. This would be much faster than FORMAT $A since the normal disk allocation mechanism involves considerable overhead. Such a program could even copy a non-ISIS disk, e.g., DEC RT11, CP/M, etc., if needed.

Similar calls exist in Version 1.6 of 16K ISIS (the 534BH entry in the parameter block is not examined, but the drive and the IOPB address are still taken from the second and third words) and in Version 2.2 of ISIS-II (the parameter block contains only the drive and the IOPB adress – the 534BH is omitted); however since this is an unpublished feature, any program using it must be considered system-dependent. The possibility exists that such programs will not work with some future version of ISIS.

User-Program Use of ISIS Procedures

A user program may occasionally need to perform a processing task similar or identical to that performed by one of the ISIS procedures. In such a case, it may be desirable to call the system routine rather than having to write and debug code to perform the same function. In order to do this, a few details of the implementation of PL/M must be understood.

Up to two parameters of a procedure are passed in via registers. (The first thing a parametric procedure does is to store the appropriate registers into the memory locations reserved for those parameters.) A procedure having one parameter will expect the parameter to be in the C register (if a Byte) or in the BC register pair (if an Address). If there are two or more parameters, the last will be in the E (or DE) and the next-to-last will be in the C (or BC). The CALLER must store all other parameters into the proper memory locations before the procedure is entered. (This sounds messy, and it is. Newer PL/M compilers pass extra parameters on the stack, which is much nicer.)

A procedure which returns a Byte value will return it in the A register; one which returns an Address will return the more-significant byte in the B register and the less-significant byte in the A. (Note: this does not apply to the compiler-generated MPY and DIVIDE routines – see Table II.) Except for such returned values, the contents of the registers upon return from a procedure cannot in general be depended upon.

The benefits of using a system routine must of course be weighed against the extreme system dependency which results: the routines will almost certainly be in different places in different versions of the system and there is no guarantee that, even after they have been found in a new version, their results will be the same as before.

Conclusion

Users of the ISIS operating system have heretofore been hampered by the lack of documentation on its internal operation. While such documentation is not necessary most of the time, the need for it occasionally becomes critical. This presentation should partially fill that need for Intel customers while also assisting hobbyists in understanding how a first-rate operating system and file handler is organized.

Table I. Principal MDS Monitor Subroutines

NAME-	ROM E.P.	ISIS E.P.	PARAMETER	T	RETURNS	T	PURPOSE----------
CI	F803	0063			CHARIN	B	Returns character read from Console input device.
CO	F809	006B	CHAROUT	B			Outputs character to Console output device.
CSTS	F812	--			STATUS	L	Returns TRUE if Console has a character ready.
IOCHK	F815	0083			IOBYTE	B	Returns current value of I/O control byte.
IODEF	F81E	--	PGMID ENTRY	B A			Defines extensible I/O entry point (see MDS Operator's Manual).
IOSET	F818	0087	IOBYTE	B			Sets I/O control byte.
LO	F80F	007B	CHAROUT	B			Outputs character to List device.
MEMCK	F81B	008F			MEMSIZE	A	Returns highest available memory address.
PO	F80C	0073	CHAROUT	B			Outputs character to Punch device.
RI	F806	0067			CHARIN	B	Returns character read from Read device.
SPCL	F821	--					Entry point reserved for future expansion.

CP/M USERS' GROUP

Dear Jim: *Received: 77 Dec 16*

Just a note to inform you that there exists a CP/M Users' Group which is active both in user software exchange and also in group purchasing of proprietary software. Although designated CP/M Users' Group, we also naturally welcome users of IMSAI DOS-A and M-DOS and of the Cromemco CDOS, and soon expect to see a TDL FDOS which is similarly compatible with CP/M in terms of program load point, DOS call convention and diskette allocation and directory format.

Kindest regards,
Anthony R. Gold

345 E 86 Street
New York, Ny 10028

NORTH STAR EXECUTIVE SOFTWARE

News Release *Received: 77 Sept 13*

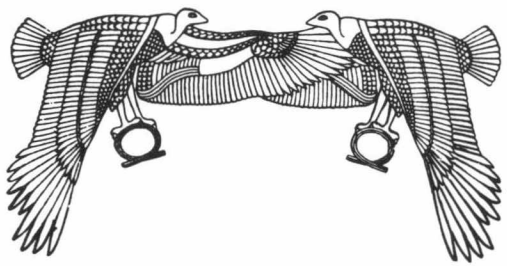
XEK, a complete system executive package for North Star users, is now available from the Byte Shop of Westminster.

The XEK package contains a disassembler capable of creating files that may be left in memory when changing from the disassembler to the executive package for re-assembly. The monitor software has the ability to accept input from cassette tapes and paper tape as either source or object files, as well as from the North Star diskette system. In addition, the assembler features a new auto-line editor for the creation of source files. This editor also extends to the modification of existing object files.

Another feature is the XEK's ability to handle up to six named files at once that may be consecutively assembled to form one object file. The assembler, monitor, and disassembler come with complete documentation, both on disk and as a manual. Total price, including first class postage, insurance and California resident's sales tax, is $48.00. For further information and ordering, contact: The Byte Shop of Westminster, 14300 Beach Boulevard, Westminster, CA 92683, (714) 894-9131.

Table II. Internal ISIS Procedures

NAME----	E.P.	PARAMETER	T	ADDR	RETURNS	T	PURPOSE----------	CALLS---
ABORT	2584			2DFF			Terminates execution and reboots system; usual entry via interrupt 1 vector at address 0008.	CLOSE XEQIOPB
ALLOCATE	0940	DRIVE	B	2CC2	TRKSEC	A	Allocates new disk block and returns its address.	ERROR RWMAP
ATTRIB	1870	.FILENAME	A	2D6A			Sets file attributes.	ERROR GETFN SEARCH SETTABS SYNC WRITE
BINASC	0093	NUMBER BASE WHERE NDIGITS	A A A B	2992 2994 2996 2998			Converts binary NUMBER to Ascii representation in requested BASE starting at WHERE.	DIVIDE
CANON	069E	.FILENAME .INTFN	A A	2B9A 2B9C	ERRNUM	B	Converts FILENAME to internal form in INTFN.	ALPHANUM
ALPHANUM	6AB			--	LETDIG	L	Returns TRUE if byte at (2B9A) is a letter or a digit.	
CLOSE	18FF	AFT	B	2D73			Closes specified file.	FREEBUF MPY READ REWIND RWMAP SEEK SETTABS SIZECK SYNC WRITE WRTSEC
CLRBUF	022D	.BUFFER	A	29DE			Clears 128-byte BUFFER to zeros.	
CONSOLE	2182	.INFILE .OUTFILE	A A	2DD0 2DD2			Changes console file assignments.	CLOSE ERROR GETFN OPEN
DEALLOC	09EC	DRIVE TRKSEC	B A	2CC9 2CCA			Releases disk block.	MPY RWMAP
DELETE	1CCB	.FILENAME	A	2D9E			Deletes a file given its name.	ERROR GETFN SCRATCH SEARCH SETTABS
DIVIDE	00DA	(BC) (DE)	A A	-- --	(BC) (DE)	A A	Compiler-generated division routine. Divides BC by DE; returns quotient in BC, remainder in DE.	
EDIT	0FD0	AFT .BUFFER COUNT .ACTUAL	B A A A	2D11 2D12 2D14 2D16			Called in place of READ when line-editing is required.	ECHO ERROR READ WRITE
ECHO	0FDD	CHAR	B	2D28			Writes CHAR on echo file.	SETTABS WRITE
ERROR	0165	FLAG CODE	B B	29A2 29A3			If FLAG = 0, returns CODE to user. If FLAG = 1, prints message. If FLAG = 2, prints message and aborts.	BINASC CO IOCHK IOSET
EXIT	252A			2DFE			Closes files and returns to system.	CLOSE LOAD RWMAP
FIO	0B18	AFT .BUFFER COUNT .ACTUAL READFLAG	B A A A L	2CCF 2CD0 2CD2 2CD4 2CD6			Transfers COUNT bytes between disk file buffer and caller's BUFFER; READFLAG is TRUE to read from file.	ALLOCATE CLRBUF GETD RDSEC RWMAP SIZECK WRTSEC
GETD	0B23			--			Allocates new data blocks as needed.	ALLOCATE CLRBUF WRTSEC
FREEBUF	0251	.BUFFER	A	29E2			Releases BUFFER to buffer pool.	
GETAFT	03E7			2B1E	AFT	B	Assigns AFT for file to be opened.	ERROR
GETBUF	0276			29E4	.BUFFER	A	Allocates a buffer and returns its address.	ERROR
GETFN	0882	.FILENAME .INTFN	A A	2BA4 2BA6			Passes params to CANON; calls ERROR if needed.	CANON ERROR



NAME----	E.P.	PARAMETER	T	ADDR	RETURNS	T	PURPOSE----------	CALLS---
ISIS	0040	COMMAND	B	2921			Principal user entry point to operating system.	ATTRIB
		BLOCK	A	2922				CLOSE
								CONSOLE
								DELETE
								EDIT
								ERROR
								EXIT
								LOAD
								OPEN
								READ
								RENAME
								RESCAN
								SEEK
								WHOCON
								WRITE
								XEQIOPB
								?OPEN
LOAD	22EC	.FILENAME	A	2DE8			Loads and optionally executes an absolute binary file.	CLOSE
		BIAS	A	2DEA				ERROR
		RETSW	A	2DEC				MEMCK
		.ENTRY	A	2DEE				OPEN
								PACKBUFS
								READ
MPY	0A0C	(BC)	A	--	(DE)	A	Compiler-generated multiplication routine. (DE) = (BC) * (DE)	
		(DE)	A	--				
OPEN	1D95	.FILENAME	A	2DA2	AFTN	B	Opens a file given its name; assigns and returns index in file table.	ALLOCATE
		ACCESS	A	2DA4				CLRBUF
		ECHOAFT	A	2DA6				ERROR
								FREE1
								FREE3
								GETAFT
								GETBUF
								GETFN
								REWIND
								RWMAP
								SCRATCH
								SEARCH
								SETTABS
								SYNC
								WRITE
								WRTSEC
FREE1	1DA2	ERRNUM	B	2DB0			Frees line-edit buffer (if any), then calls ERROR(0, ERRNUM).	ERROR
								FREEBUF
FREE3	1DC3	ERRNUM	B	2DB1			Frees link and data buffers, then calls FREE1(ERRNUM).	FREEBUF
								FREE1
PACKBUFS	0415			2B20	LOWLIM	A	Moves buffers currently in use to lowest possible locations; returns lowest vacant address.	LOWBUF

NAME----	E.P.	PARAMETER	T	ADDR	RETURNS	T	PURPOSE----------	CALLS---
LOWBUF	0418	.BUFPTR	A	2B22			Moves buffer whose address is in BUFPTR to a lower position, if one is available.	FREEBUF
								GETBUF
RDSEC	0658	TRKSEC	A	2B44			Reads requested disk block into BUFFER, from proper drive for current file.	XIO
		.BUFFER	A	2B46				
READ	0E63	AFT	B	2CDD			Reads characters from file into BUFFER until COUNT is satisfied or end-of-file is reached.	CI
		.BUFFER	A	2CDE				ERROR
		COUNT	A	2CE0				FIO
		.ACTUAL	A	2CE2				RI
								SETDEV
								SETTABS
RENAME	1AE8	.OLDNAME	A	2D8E			Changes the name of a file.	ERROR
		.NEWNAME	A	2D90				GETFN
								SEARCH
								SETTABS
								SYNC
								WRITE
RESCAN	1276	AFT	B	2D29			Sets the next-byte pointer of a line-edited file to the beginning of the current line.	ERROR
								SETTABS
REWIND	12B4			--			Initializes several file tables and reads in first link block.	RDSEC
RWMAP	08AC	DRIVE	B	2CC0			Reads or writes bit map on requested drive, using internal buffer.	OPMAP
		OPCD	B	2CC1				
OPMAP	08B5			--			Constructs IOPB'S for RWMAP and has them run.	XEQIOPB
SCRATCH	1BB7	DRIVE	B	2D95			Deletes all disk space allocated to file whose directory entry is at 2D4A.	DEALLOC
								RWMAP
								SEEK
								SETTABS
								WRITE
								XEQIOPB
SEARCH	1723	.INTFN	A	2D5C			Looks up file represented by INTFN in appropriate directory.	FREEBUF
								GETBUF
								MPY
								READ
								REWIND
								SEEK
								SETTABS

NAME----	E.P.	PARAMETER	T	ADDR	RETURNS	T	PURPOSE----------	CALLS---
SEEK	12F0	AFT	B	2D2D			Repositions the next-byte pointer of a disk file.	ALLOCATE CLRBUF ERROR OFFEND RDSEC REWIND RWMAP SETTABS SIZECK WRTSEC
		MODE	A	2D2E				
		.BLOCKNO	A	2D30				
		.BYTENO	A	2D32				
OFFEND	12FD	BLKNUMA	A	2D44	LARGER	L	Returns TRUE if requested seek would enlarge file.	
		BLKNUMB	A	2D46				
		BYTENUMA	B	2D48				
		BYTENUMB	B	2D49				
SETDEV	0E3D			--			Sets up ROM's "IOBYT" to steer I/O properly.	IOCHK IOSET
SETTABS	02C8	AFT	B	2B1D			Sets up "current pointers" into file tables.	
SIZECK	0AC0			--			Updates file length in tables.	
SYNC	1711			--			Forces delayed write of current data block.	WRTSEC
WHOCON	22A0	AFT	B	2DE1			Copies into BUFFER the name of the requested console file.	
		.BUFFER	A	2DE2				
WRITE	0F12	AFT	B	2CF7			Writes COUNT characters on file from BUFFER.	CO ERROR FIO LO PO SETDEV SETTABS
		.BUFFER	A	2CF8				
		COUNT	A	2CFA				
WRTSEC	067B	TRKSEC	A	2B48			Writes requested block from BUFFER, on proper drive for current file.	XIO
		.BUFFER	A	2B4A				
XEQIOPB	0514	DRIVE	B	2B29			Runs given IOPB on requested drive.	ERROR
		.IOPB	A	2B2A				
XIO	05F6	OPCODE	B	2B34			Constructs IOPB to perform requested operation, and has it run.	XEQIOPB
		DRIVE	B	2B35				
		TRKSEC	A	2B36				
		.BUFFER	A	2B38				
?OPEN	25B8			--			Increments user-supplied AFTN value by 2 to obtain internal value; if this does not correspond to an open file, calls ERROR(0, 2).	ERROR

Table III. ISIS Variables of Diagnostic Significance

ADDRESSES	CONTENTS----------
28C0-2920	Unused area, reserved for future expansion. Might be used for temporary debugging code, or for "patches."
2928-2931	Copy of current command parameter block.
294C-294D	User's Stack Pointer.
294E-298D	ISIS stack.
299C	Debug Switch (Logical). When True, fatal errors will invoke the MDS Monitor instead of aborting the program.
299D	"CONSOLE" command flag. Value is 2 while processing a CONSOLE system call, and zero at all other times. This is what causes all errors detected by CONSOLE to be fatal.
299E-299F	"Result Byte" and "Result Type" generated by disk channel in connection with a "hard" (non-recoverable) disk error. For details, see the MDS-DOS Hardware Manual. (These are printed as part of the "ERROR 24" message.)
29CB-29DD	Buffer Table, containing one byte for each of 19 possible buffers. The possible values of each byte are: 0 Free. 1 Space preempted by LOAD since the buffer area contains part of the user program. 2 In use as a buffer. The buffers themselves begin at address 2E00; each buffer is 128 bytes long.
29E5-2B18	File Structure (see Table IV).
2B29	Disk drive most recently accessed.
2B2A-2B2B	Address of IOPB.
2B2C	Result Byte.
2B2D	Result Type.
2B4C-2B75	Table of 2-byte device names (each name stored backwards).
2B86-2B8F	Internal-format name of last file sought by SEARCH. 2B86 contains the device identification (index in table at 2B4C), 2B87-2B8C contain the file name, 2B8D-2B8F contain the name extension.
2BA8-2CA7	Bit Map Buffer.
2CA8	Drive to which this bit map belongs.
2CA9	(Logical) True if bit map has been modified since it was last written.
2D4A-2D59	Directory Entry for last file found by SEARCH, as it then appeared. (This is not kept current with respect to growth of the file.)

ADDRESSES	CONTENTS
2DB2-2DC0	Name of current :CI: file.
2DC1-2DCF	Name of current :CO: file.
2DF6-2DF7	Address field of last logical record read by LOAD. (Except during a LOAD, this turns out to be the entry point of the last file LOADed.)
2E00-	Buffer Area.

Table IV. ISIS File Control Structure (in memory)

NAME----	BASE ADDR	T	CUR. PTR.	I.V.	DESCRIPTION----------
CLOSED	2A0E	L	29EA		True if this file is not open.
DEVICE	2A18	B	29EC		Device identification (see Table V).
ACCESS	2A22	B	29EE		Value of OPEN's ACCESS parameter.
ECHOAFT	2A2C	B	29F0		AFT of echo file; zero if non-edit.
EBUF	2A36	A	29F2		Address of edit buffer.
DBUF	2A4A	A	29F4		Address of data buffer (copy of current element at 29E8).
BYTENO	2A5E	B	29F6	128	Byte counter in data buffer.
DNUM	2A68	A	29F8		Position in directory.
LBUF	2A7C	A	29FA		Address of link-block buffer (copy of current element at 29E6).
DPTR	2A90	B	29FC	1	Word pointer in link block.
LASTBYTE	2A9A	B	29FE		Number of bytes in last data block.
ALLOC	2AA4	L	2A00	F	True if allocation has been done for this file.
DMOD	2AAE	L	2A02	F	True if current data block has been modified.
BLKCNT	2AB8	A	2A04		Number of data blocks in this file.
BLKNO	2ACC	A	2A06	0	Sequential number of current data block within this file.
LADDR	2AE0	A	2A08	(2AF4)	Track-Sector address of current link block.
L1ADDR	2AF4	A	2A0A		Track-Sector address of file's first link block.
DADDR	2B08	A	2A0C	0	Track-Sector address of current data block.

Table V. ISIS Device Identification Codes

NUMBER	DEVICE
0	:F0:
1	:F1:
2	:TI:
3	:TO:
4	:VI:
5	:VO:
6	:I1:
7	:O1:
8	:TR:
9	:HR:
10	:R1:
11	:R2:
12	:TP:
13	:HP:
14	:P1:
15	:P2:
16	:LP:
17	:L1:

Table VI. ISIS Format Files

FILE NAME	HEX DISK ADDRESSES LINK	---DATA---	CONTENTS----------
ISIS.T0	0,18	0,1 - 0,17	System Initialization Program.
ISIS.LAB	0,19	0,1A	Disk Label.
ISIS.DIR	1,1	1,2 - 1,1A	Disk Directory (see Table VII).
ISIS.MAP	2,1	2,2 - 2,3	Allocation Bit Map.

Table VII. ISIS Directory Entry

BYTES	CONTENTS----------
00	Flag: 00 = active 7F = never used FF = deleted
01-06	File Name
07-09	Name Extension
0A	Attributes: Bit 0 = Invisible Bit 1 = System Bit 2 = Write-protect Bit 7 = Format
0B	Number of bytes in last data block
0C-0D	Number of data blocks
0E-0F	Disk Address (Block, Track) of first Linkage block

The SAM76 Language

by Ancelme Roichel
copyright 1977
Pennington New Jersey, U.S.A.
All Rights Reserved

This text was formatted from a raw text file
employing procedures which make use of the

<ncra> 21

SAM76 language processor

Typographical and code assignment table

	0 1 2 3 4 5 6 7	8 9 A B C D E F
	0 0 0 0 1 1 1 1	2 2 2 2 3 3 3 3
	0 2 4 6 0 2 4 6	0 2 4 6 0 2 4 6
0 00	0 @ P ` p	0 @ P ` p
1 01	! 1 A Q a q	! 1 A Q a q
2 02	" 2 B R b r	" 2 B R b r
3 03	# 3 C S c s	# 3 C S c s
4 04	$ 4 D T d t	$ 4 D T d t
5 05	% 5 E U e u	% 5 E U e u
6 06	& 6 F V f v	& 6 F V f v
7 07	' 7 G W g w	' 7 G W g w
8 10	(8 H X h x	(8 H X h x
9 11	) 9 I Y i y	) 9 I Y i y
A 12	* : J Z j z	* : J Z j z
B 13	+ ; K [k {	+ ; K [k {
C 14	, < L \ l |	, < L \ l |
D 15	- = M] m }	- = M] m }
E 16	. > N ^ n ~	. > N ^ n ~
F 17	/ ? O _ o	/ ? O _ o

The SAM 76 Language

The SAM76 language combines into a single interpretive processor characteristics of two different string and general purpose macro generators and one (or more) infix operator mathematical systems.

SAM76 was primarily inspired by the "M6 MACRO PROCESSOR" designed by M. D. Mc Ilroy and R. Morris of the Bell Telephone Laboratories. A description authored by Andrew D. Hall is given in the Bell Lab. Computing Science Technical Report #2, and other places.

The second source of inspiration came from the syntax of "GPM - a GENERAL PURPOSE MACRO GENERATOR" described by its author - C. Strachey.

GPM syntax very significantly improves the interface between the user and the language by eliminating an awkwardness inherent in the design of M6. This awkwardness arises out of the fact that in the M6 language the specification for the action to be performed on the result of an expression evaluation is at the right hand end of the expression to be evaluated; this requirement causes the need for a unique pair of characters to be designated for the purposes of "quotation"; consequently when the writer of procedures in M6 is terminating the writing of an expression he must mentally reconstruct the sequence of alternations of "quoted", "active" and "neutral" expressions which may be nested one within the other.

In GPM, where the nature of the expression is specified at the left end, only one character need be identified to serve to provide the right hand boundary for all three types.

This means that the user, when closing out his writings, need only supply enough such identical characters (plus a few for good measure) to enable correct action to subsequently take place.

An universal lament of users of both M6 and GPM relates to the extremely unnatural method in which arithmetic expressions must be organized. The simplest arithmetic expression in normal "infix" notation becomes a tortuous involved nest of strange symbols when expressed in either M6 or GPM. A simple solution exists, is incorporated in SAM76 and forms the third element of the language.

In addition inspiration for the selection and design of a number of the resident functions came from a variety of sources (see reference list).

Acknowledgement is made of the contribution made by L. Peter Deutsch [U.C. Berkeley] who conceived of nesting an expression calling for input inside an expression which causes output [Ref. 3] thus:

{OUTPUT,{INPUT}}

Finally acknowlegement and great appreciation is expressed for:
The help received from Neil Colvin [M.I.T.] who provided me with an 8080 simulator for the PDP-10 and who coded a preliminary version of the language to run in an 8080 based microprocessor system contributed also materially to the development of this language.

Barry Lubowsky [Rider College] who allowed me to make use of a PDP-10 and who was most helpful with some system software problems.

Roger Amidon [Delta Data Systems], whose actual 8080 based "ALTAIR" and "IMSAI" microcomputers were used to test and debug the software, was also most helpful in developing a family of "people" oriented mnemonics and functions particularly well suited for the "amateur" user.

Other people who contributed to various details and who "people" tested the language are Carl Galletti [T.D.L.], Marty Nichols [Newsweek], and Tom Kirk [N.J. Bell].

References

1. Strachey, C. - "A general purpose macrogenerator", Computer Journal, Vol. 8, No. 3, Oct. 1965 p. 225; |19651000|.

2. Hall, Andrew D. - "The M6 macroprocessor", Bell Telephone Labs; Computer Science Report No. 2; 1971;

3. Kagan, Claude A. R. - "A string language Processor for small machines", Proceedings of the ACM SIGPLAN Symposium on the Pedagogical Applications of small Computers, University of Kansas, Lawrence, Kansas |19711118|.

Notice This description of the SAM76 language is substantially the same that is available from the IEEE Computer Society Repository, R 76-301 - August 1976.

The three elements of SAM76

The first element is designed around the characteristics described by STRACHEY in his GPM (General Purpose Macro Generator) language. **GPM - S76**

Strachey defined its general forms as follows:

Active %function,arguments, ... :

Neutral %VAL,function,arguments, . . . :

In order to avoid ambiguity and to provide consistency the above is changed for the purposes of SAM76 as follows:

Active %function,arguments, ... /

Neutral &function,arguments, ... /

Protected ! . . . protected string . . . /

In addition changes and extensions are made to the language originally described by Strachey. The modified form incorporated in SAM76 is to be known as S76; this is a fully operational proper subset of SAM76.

The syntax used in connection with the S76 system is to be referred to as the "S" syntax of SAM 76.

The second element is substantially as described by Andy HALL for the M6 language attributed to Mc ILROY and MORRIS: **M6 - M76**

Active #function,arguments, ... :

Neutral #function,arguments, ... ;

Except for different function mnemonics no change was made in the syntax and notation of M6 for the purposes of SAM76.

In consequence routines and texts written for M6 processors distributed by the Bell Labs outside the Bell System should be compatible with SAM76.

Compatibility with M6 is insured by enabling user to stipulate whether "resident" or "user defined" functions are to be first searched whenever an expression is to be evaluated.

Expressions written using the M76 syntax will first cause a search of the user defined functions; if no user defined function of that name exists then a search is made of the resident functions. Expressions written using the S76 syntax will first search the resident functions.

Although there is syntactic compatibility with M6, the implementation in SAM76 is designated M76 because of the generally different complement of resident functions, and the significant enhancement of the scan algorithm.

The syntactic form of expressions based on the M76 system is to be referred to as the "M" syntax.

An infix operator system characterized in general by the following example for a simple arithmetic expression: **A76**

Example [5 + 5 * (144/12)]

The syntax of the foregoing system is to be referred to as the "A" syntax. The evaluation rules and internal syntax may vary between systems for this sub system; a method of identification is provided within the framework of SAM76.

Input - Activating - Scanning and Output

Input and Output The flow of characters, be they procedures (programs) or just plain text between the SAM76 processor and either the user or other sources or destinations is under the control of two resident SAM76 functions:

Input from user (or source) to SAM76 is under control of the "input string" function whose mnemonics are "IS"; this may be used in either the "S" or "M" syntax either as an active or neutral function thus:

type	active		neutral
S76	%IS/	or	&IS/
M76	#IS:	or	#IS;

Output from the SAM76 processor to the user (or other destination) is controlled by the resident "output string" function, mnemonic of which is "OS"; active or neutral use in both the "S" and "M" syntax is:

type	active		neutral
S76	%OS/	or	&OS/
M76	#OS:	or	#OS;

Initially and whenever its task has been completed, the SAM76 processor automatically loads a "restart" expression which is scanned and provides the means for accepting input and delivering output; this expression is initially defined as the S76 expression: **The "restart" expression**

&os,%is//=

The initial or standardized condition makes use of the equal sign "=" to ACTIVATE the processor; this "activating" character may be changed by the user through execution of the appropriate SAM76 resident function.

Activating the processor

Scanning

When expressions are nested one within the other, the SAM76 processor first executes the innermost, proceeding from left to right.

The innermost in the "restart" expression is:

```
%is/
```

and therefore input is sought; the user input is terminated with an activating character, the (=) will be shown to remind the user how to terminate his input.

Once the input is terminated, it is in turn examined by the scanner for possible further execution until finally the outermost expression of the "restart" expression is encountered:

```
&os, ... /
```

where the series of periods represents the result of processing the user input and in fact will be the output that is to be given the user.

Quoting or Protection

Frequently the user may wish to incorporate in his text certain characters which have special meaning to the scanning system of SAM76. Typical of these are the characters used to indicate the start and end of expressions. These are known as "warning characters".

For standardization purposes a processor for the SAM76 language is initialized with the following assignment of "warning characters":

Abbv.	Char.	Octal	Purpose
SA	%	45	Begin S76 active expression
SN	&	43	Begin S76 neutral expression
SP	!	41	Begin S76 protected string
SE	/	52	End S76 expr. or protected string
MB	#	43	Begin M76 expression
MA	:	72	End M76 active expression
MN	;	73	End M76 neutral expression
AS	,	77	Argument separator
QC	@	100	Quote single character
BQ	<	74	Begin quoted string - Pair 1
EQ	>	76	End quoted string - Pair 1
BP	(	50	Begin protected string - Pair 2
EP	)	51	End protected string - Pair 2
BA	[	133	Begin "A - syntax" expression
EA	]	135	End "A - syntax" expression

Quoting or protecting a string

In both the "S" and "M" forms of SAM76, the equivalent of string quotes may be provided by any pair of characters designated for that purpose. Initially two such pairs are defined and consist of the following balanced pairs of enclosures:

```
< . . . . > or ( . . . . )
```

Strings of symbols found between the above illustrated enclosures are said to be "quoted" or "protected" and are completely ignored during the scanning process.

The two rules that follow are applied to the scanning process with respect to the use of "quoting characters".

2

When an unprotected character designated as a "quoting character" is encountered in the left to right scan of an expression, the other character of the pair is then defined as the character which will end the quoted string; this means that for the two initially designated pairs shown above, quoting or protection may also be achieved as follows:

```
> . . . . < or ) . . . . (
```

1

In the process of finding the character which is to terminate a particular quoted string any occurrence of the same character as that which identified the beginning of that quoted string must be balanced by a corresponding "end of quoted string" character.

This is to be differentiated from the action which takes place in connection with the "protected string" of the "S" syntactic form:

```
! . . . protected string . . . /
```

In the foregoing a tally is made of "unquoted" warning characters in order to find the slant sign "/" which matches the exclamation mark:

The tally count is increased by a count of one - % & ! #

The tally count is decreased by a count of one - / : ;

It should, however, be noted that the "warning characters" (symbols used to delineate bounds of expressions) shown in the foregoing illustrations and examples are changeable by the user and are only representative of initial or standardized conditions.

Any of the foregoing "warning characters" may be changed using the appropriate SAM76 resident function; the abbreviations listed earlier are used to identify the particular warning character it is desired to change.

Quoting or protecting a single character

When an unprotected "character quoting symbol" is encountered, then the immediately following character is considered to be quoted. This is particularly useful when the user wishes to quote single characters, such a warning character, without having to maintain a balance of quoting character pairs.

Initially the "character quoting symbol" is set to be the "commercial at" (@).

Using the SAM76 syntax, functions, and trace

In the examples that follow the | ... | arrangement is used only to delineate output at the user terminal; if for instance the user wished to add two numbers making use of the "A" syntactic form, the following might be observed:

```
[ 5 + 5 ]=|10|
```

M76 example

In the following example the M76 syntax is used to create a "macro" which when invoked will square any desired number:

```
#DT,SQ,<#MU,q1,q1:>:=          DT means Define Text
                               MU means Multiply
                               SQ is name given to text

#PT,SQ,q1:=                    PT means Partition Text
```

There are four possible actions which may ensue when the macro, or user defined function named SQ is to be invoked:

```
1.      #SQ,5:=|25|            value returned is 25

2.      #FT,SQ,5:=|25|         FT - Fetch Text, value 25

3.      #SQ,5;=|25|            value returned is 25

4.      #FT,SQ,5;|#MU,5,5:|    value is Text of SQ
```

S76 example

The foregoing example is repeated below making use of the S76 syntax; however to illustrate its use, the infix system is used for the actual multiply function:

```
%DT,SQ,![q1 * q1]//=           q1 is dummy variable

%PT,SQ,q1/=                    q1 becomes partition valu
e
1
```

The four invocations of SQ are shown below:

```
1.      %SQ,5/=|25|            value is 25

2.      %FT,SQ,5/=|25|         value is again 25

3.      &SQ,5/=|25|            value is still 25

4.      &FT,SQ,5/=|[ 5 * 5 ]|  value is now "text" of SQ
```

Resident Functions

In general the mnemonics used in SAM76 to identify the resident functions are composed of two alphabetic characters. This is not a restriction imposed on the language, and three or more alphabetic characters may be used to identify resident functions. In fact the user may wish to employ the "@F" ("wh@" (what) Functions) mnemonic to ascertain the mnemonics in actual being in his system.

It should be remembered that when the "@" sign serves the purpose of "quoting single character" then the expression to list the resident functions would be:

```
M76     #@@F, ... :

S76     %@@F, ... /
```

where " ... " is the string of characters which will precede each resident function name in the result of executing this expression.

Descriptions of the functions and examples in which either S76 or M76 syntax is applicable employ the | (Long Vertical Mark) character to bound the expression. In actual use the | is replaced by the appropriate pair of warning characters.

In the case of "null" valued functions where the neutral form of the expression is to be illustrated, the \ "reverse slash" is used in lieu of the |.

The formal description of the "DT" - "Define Text" function is given below to illustrate the manner of presenting SAM76 resident functions.

|DT,t,s,dl| Define Text with id number

This function places in the "text area" the string denoted by "s" in the expression shown above; this text is given the name denoted by "t". An arbitrary identification decimal number denoted by "dl" may also be included as part of the "text" record. Previously defined text with same name is erased and replaced with this definition.

Examples:

```
M76     #DT,NAME,THE SKY IS BLUE:

S76     %DT,NAME,THE SKY IS BLUE/
```

In the foregoing examples no id. number was desired, and consequently the default value of zero is stored as part of the record created by this function.

The "VT" View Text function

Texts in the "text area" may be examined without evaluation using the "vt" function. This examination will display location and value of any "partitions", "multi-partitions", and the text divider(s) when at a location other than the head (or extreme left) end of the text.

Initial conventions for the display within a "text" of partitions, multi-partitions and text divider(s) are:

```
[1], [2], ... [d]              Partitions

[#1], [#2], ... [#d]           Multi-partitions

[^]                            Text Divider
```

Trace

When in the "Trace" condition, active expression about to be evaluated will be shown bounded by the |, neutral expression by \, regardless of the syntax (M or S) of the expression.

"id" numbers

The M6 language provides a unique identification number for each of the resident functions of the language, and means for the user to assign a similar type of number to defined functions. This numbering concept is preserved in SAM76; numbers between 1 and 99 inclusive are not assigned to any resident SAM76 function unless its name and operation is absolutely identical to its M6 counterpart. This permits user defined functions to be written to mimic M6 functions if desired. A list of M6 functions and their assigned "id" numbers is given in an appendix.

User defined function

To illustrate, the M6 function "DEF" might be mimicked in the SAM76 system as follows:

```
1a.     %DT,DEF,!%NI,<&>,<%>/DTmpl</>//

2.      %MP,DEF,mpl/
```

examination of the defined text (or function) DEF can be made using the SAM76 "VT - View Text" resident function thus:

```
3.      %VT,DEF/=|%NI,<&>,<%>/DT[#1]</>|
```

in the foregoing "mpl" is a dummy for a "multi-partition" which will be created by the second expression and visualized by the VT function as [#1].

The function "NI - Neutral Implied" is used to provide the expression using DT with either the active or neutral warning character as used with an expression containing "DEF" as its operative argument.

Alternatively expression (1a) could have been written in one of the following ways,with exactly the same end effect:

```
1b.     %DT,DEF,<%NI,@&,@%/DTmpl/>/

1c.     #DT,DEF,<#NI,@&,@%:DTmpl/>:

1d.     #DT,DEF,<#DTmpl%NI,@:,@:/>:

1e.     #DT,DEF,<#DTmpl#NI,<;>,<:>:>:
```

Algorithms in the SAM 76 Language

introduction

In the algorithms described and illustrated in this section, use is made of the procedure named "print" to display the script of the functions, expressions, and other text strings.

The "print" function is useful in that it augments the capability of the "vt - view text" function by prefixing each individual "text" displayed with its name bounded by angle brackets; assuming that texts have been created as shown:

```
{}~~~~~~~~~~~~~~~~~~~~~~~~
{}  %dt,a,apple/=
{}  %dt,b,bottle/=
{}  %pt,b,t/=
{}
~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The use of "print" would cause the following display:

```
< a >
apple

< b >
bo[1][1]le
```

print

The need and use of the "print" expression was described earlier; the expression to accomplish this task uses a multi partition value #2.

Typical use to print out name and contents of the text area which in this example only contains the "print" expression would be:

```
{}~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
{}  %print%lt,(,)//=
{}  < print >
{}  %ii,[1],,,!os,
{}  (< [1] >)/%vt,[1]/%os,
{}  /%print[#2]///
{}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

do "n" Times

A simple way of doing some procedure "proc" some desired "n" number of times is exemplified by the use of "ds - duplicate string" in the expression named "dnt":

```
< dnt >
%ds,[1],!%proc///
```

factorial

The simple recursive function named "fac" generates the factorial of any number which replaces partition value 1 in its use.

```
< fac >
%ig,1,[1],1,!%mu,[1],%fac,%su,[1],1/////
```

```
{}~~~~~~~~~~~~~~~~~~
{}  %fac,5/=120
{}
~~~~~~~~~~~~~~~~~~~~~
```

exponential

This recursive function named "exp" is used to generate the exponential of "m" to the "n"th. power; in use "m" replaces partition value 1, and "n" partition value 2.

```
< exp >
%ig,[2],,!%mu,[1],%exp,[1],%su,[2],1////,1/
```

```
{}~~~~~~~~~~~~~~~~~~~~
{}  %exp,3,7/=2187
{}
~~~~~~~~~~~~~~~~~~~~~~~
```

"m"th. root of "n"

The recursive expression named "mrn" extracts the "m"th. root of any number "n"; it should be noted that this function makes use of the expression named "exp", which generates an exponential.

```
< mrn >
%ig,%exp,[3],[2]/,[1],
!%su,[3],1//,
!%mrn,[1],[2],%ad,[3],1////
```

```
{}~~~~~~~~~~~~~~~~~~~~
{}  %mrn,2187,7/=3
{}
~~~~~~~~~~~~~~~~~~~~~~
```

tensor

Expression named "tensor" is used to generate a list of successive numbers separated by "."; the size of the list is specified by the integer number which replaces partition value 1 in the expression.

```
< tensor >
%ig,[1],1,!%tensor,%su,[1],1//.[1]/,1/
```

Decimal to Roman

A frequently needed capability is that of generating "roman" numbers, as for text section and chapter headings; when function "dtr" is invoked with a decimal number as its argument, the value returned is the same in roman notation.

Thanks are extended to Jim Gimpel, of the Bell Telephone Laboratories for suggesting this as a most useful adjunct to the SAM76 language, as it is to the SNOBOL language.

```
< dtr >
%dt,x,[1]/%roml/%y/%et,x,y,z/

< roml >
%ig,%crd,x/,,!%rom2,%fc,x//%roml///

< rom2 >
%dt,z,[1]/%pt,z%d//%dt,z,%z%r///%pt,y%u//%dt,y,%y%t//%z//

< d >
,0,1,2,3,4,5,6,7,8,9

< r >
,,I,II,III,IV,V,VI,VII,VIII,IX

< u >
,I,V,X,L,C,D,M

< t >
,X,L,C,D,M,*,$
```

The "hex" freak can fulfill his desires by defining the "htr" expression to achieve "hex to roman" conversion:

hex to roman

```
< htr >
%dtr,%xd,[1]//
```

The user often finds it necessary to derive a series of answers resulting from the execution of some script for a range of numbers "n1" to "n2"; the script loop provides this capability by enabling the user to specify for partition 1 the name of the function, the lower of the two numbers "n1" as partition 2 and the higher number "n2" as partition 3.

loop

```
< loop >
%zs,6,[2]/%zs,7,%su,[3],[2]//%zi,7/%lpl,[1]/

< lpl >
%lp2,[1],%zq,6//%zi,6/%zd,7,,,!%lpl,[1]///

< lp2 >
!
/%ps,-4, ,[2]/    %[1],[2]/
```

For example the user wishes to ascertain the "roman" number equivalent of the decimal numbers from 105 to 116:

```
{}~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
{}  %loop,dtr,105,116/=
{}   105    CV
{}   106    CVI
{}   107    CVII
{}   108    CVIII
{}   109    CIX
{}   110    CX
{}   111    CXI
{}   112    CXII
{}   113    CXIII
{}   114    CXIV
{}   115    CXV
{}   116    CXVI
{}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

The expression named "hanoi" is a classical example of recursive capability; although the original is said to be based on monks moving rings between the three towers of Bhrama, the name "hanoi" seems to be better known for this puzzle.

hanoi

The expression shown below was conceived by Dick Stone; it solves the problem, with a minimum number of moves:

```
< hanoi >
%ii,[1],0,,!%hanoi,%su,[1],1/,[2],[4],[3]/!
Move Ring [1] from [2] to [3]/
%hanoi,%su,[1],1/,[4],[3],[2]///
```

Given the need to move four rings from "hither" to "yon", using "thither" as a way station, the monks needed only to follow the following sequence to do the task in a minimum length of time:

```
{}~~~~~~~~~~~~~~~~~~~
```

In his book "Algorithms in SNOBOL4" Jim Gimpel, the author, offers SKIM as an example of string manipulation; it is interesting to note the typing effort required for the same function in each language; it is assumed that a speed of ten words per minute (or one character per second) is reasonable for intellectual typing. At that rate the SAM76 script would take approximately 100 seconds, while the SNOBOL4 example would require approximately 143 seconds to type up.

SAM 76 Language Functions

The SAM76 language functions described in the following pages represent a first draft for test and evaluation purposes, and include all of the functions available in a processor for that language initially coded for the INTEL 8080 and Zilog Z-80 microprocessors.

In particular these functions properly represent the action expected for processors which have the <ncra> identification; see function [@n].

The references listed below should be used to provide information as to language syntax, method of use, and applications:

References

1. "SAM76 - A language based on "Strachey's GPM and Mc Ilroy's M6", by Claude A. R. Kagan, dated august 1976, and available as report R 76-301 from the IEEE Computer Society, 5555 Naples Plaza, Long Beach, CA, 90803.

2. "SAM76 Language Primer" to be published.

3. Notes on loading and using SAM76, from the Amateur Computer Group of New Jersey.

The author would greatly appreciate feedback, particularly if errors are found in this document, or if the definitions do not seem to correspond to the action encountered in a given processor.

Ancelme Roichel
Box 257 - RR1
Pennington - N. J., 08534

Notation used in list of functions

#	Preliminary definition - no ID number assigned
|function,arg1,arg2,...,argn|	Active form of execution
\function,arg1,arg2,...,argn\	Neutral form of execution
@	"at" (wh@ means "what")
c2,c1, ... ,c	Single characters
s2,s1, ... ,s	Character strings
s0	Prefixing character string
t2,t1, ... ,t	Names of Texts
f2,f1, ... ,f	Names of Files
d2,d1, ... ,d	Decimal Numbers
n2,n1, ... ,n	Numbers in current radix
x2,x1, ... ,x	Binary numbers (octal/hex)
vz	Value if function fails
vt	Value if function True
vf	Value if function False
v0	Value if function zero
v+	Value if function positive
v-	Value if function negative
dev	Device
a2,a1, ... ,a	Abbreviations
xn1,yn1, ... xn,yn	Increments in X and Y

con	Condition
EQ	Equal
GT	Greater Than
LT	Less Than
ZE	Zero
GE	Greater than or Equal
LE	Less than or Equal
UC	Upper Case
LC	Lower Case

When a radix greater than 10 is used upper case alphabetic symbols represent the ensuing integers starting with "A" being equivalent to 10 in base 10, "B" to 11, and so on. Lower case characters are used when the radix exceeds 36.

Function Definitions

238 - |@f,s0| wh@ are Functions

The value of this function is a list of built-in functions available within the system. Each function mnemonic is preceded by the string of characters symbolized by "s0". It is advisable to use the "neutral" form of expression as shown below:

&@@f, /={value will be list of functions}

Note use of two "@"; since that symbol is a warning character it must be protected, and in the above example it is used as the single character protector to protect itself.

239 - |@n| wh@ is processor ser. Number

The value returned through execution of this function is the version number of the processor being used. This number is tested by some of the functions - and may also be so tested by the user - to ascertain compatibility between scripts and run time load modules that may be moved between systems and users.

237 - |@t| wh@ is processor Title

The value of this function provides information as to the authorship of the processor being used; a typical example is:

&@@t/=<ncra> indicating that the authors are:

"Neil Colvin Claude Roichel Roger Amidon.

159 - |ab,s1,s2,vt,vf| Alphabetic Branch

This function compares the character strings denoted by "s1" and "s2" in terms of their "ASCII" binary value; if the string "s1" has a lesser value than the string "s2" then the value of the expression is the string denoted by "vt", else the value is the string denoted by "vf"; if the value strings are expected to contain executable or syntactically meaningful symbols they should be protected within the expression.

128 - |ad,n1,n2,n3,...,n| Add

The value of this expression is the arithmetic sum of the integer numbers denoted by "n1", "n2", ... "n". Each of the arguments is examined from right to left until a symbol not representing a valid number in the current number base is encountered; non numeric symbols prefixing the arguments are ignored with the exception of those in argument "n1" which is carried into the value string of the expression.

For example:

```
%ad,dog5,cat6/={dog11}
```

Negative numbers are indicated in the usual manner with a "-" sign which must immediately precede the string of integers it applies to thus:

```
%ad,dog-5,cat6/={dog1}
```

Ommitted arguments (such as adjacent commas) have the value of the "null string" and a numeric value of zero (0) as illustrated in the following example:

```
%ad,dog,/={dog0}
```

160 - |ai,s0,s1,s2,...,s| Alphabetic Insertion

The value of this function is a character string in which the string denoted by "s1" has been inserted ahead of the first of the subsequent argument string which has a greater alphabetic value; each of the strings which form the arguments of the expression are returned with a copy of the string symbolized by "s0" preceding it.

187 - |and,x1,x2| And the bits

The value of this function is the result of performing an anding of the two binary numbers represented in the current "X" base by "x1" and "x2".

161 - |as,s0,s1,s2,...,s| Alphabetic Sort

The value of this function is a character string in which the sequence of the strings denoted by "s1", "s2", ... "s" is rearranged to be in proper alphabetic sequence; each of these strings is preceded by the string denoted by "s0" in the result:

```
%as,(,),horse,cat,dog/={,dog,cat,horse}
```

Note the use of "(,)" for "s0"; this is typical of lists that the user may wish to use in a variety of functions which have arbitrary number of arguments.

220 - |bf,f,vz| Bring File

Execution of this function serves to bring from auxiliary storage the file whose name is denoted by "f"; if this file does not exist, then the value of the function becomes the string symbolized by "vz".

113 - |ca,s| Change Activator (current)
\ca,s\ Change Activator (initial)

The purpose of this function is to change the symbol which terminates input from its current value to that denoted by the first character of the string denoted by "s"; active form is used to change the operational activating character whereas the neutral form is used to change the initial table.

195 - |cfc,d1,s| Change Fill Character schema
\cfc,d1,s\ Change Fill Char. (initial)

The purpose of this function is to enable the user to specify the number of that character which is the first one of the string denoted by "s" that will be automatically output at the beginning of each "new line"; usually these will be "nulls" but they may be any other character the user wishes to use. Active form of the expression changes the table in active use, while the neutral form changes the initial table; see [cnb].

193 - |cin,t1,d1,...,t,d| Change Id Number

To be defined and coded.

148 - |cld,t| Characters Left of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

191 - |cll,d| Change Line Length (active)
\cll,d\ Change Line Length (initial)

The purpose of this function is to change the number of characters that the processor will output before automatically inserting a "new line" code; the desired value is the decimal number denoted by "d" in the expression; the active form changes the actively used constant, while the neutral form changes the initial condition table; see [cnb].

133 - |cnb,d| Change Number Base (active)
\cnb,d\ Change Number Base (initial)

The purpose of this function is to change the radix of the processor's arithmetic functions; this is always expressed as the decimal number symbolized by "d" in the expression; see [qnb]; active form of the expression changes the value of the radix used during the actual operations while the neutral form changes the initial table. Note that the neutral change can only be effective while the processor is in unprotected memory, and will have no effect if the processor is executed from read only or protected memory.

266 - |cpc,t1,d1,...,t,d| Change Protection Class

To be defined and coded.

147 - |crd,t| Characters Right of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

203 - |cro,s1| Change Rub Out char. schema
\cro,s1\ Change Rub Out (initial)

The purpose of this function is to permit the user to specify as the first character of the string denoted by "s1" that symbol desired to serve the character delete function; in addition the user may specify the next two characters of that string to be output before and after, respectively, the deleted string of characters. The active form changes the user table, while the neutral form of this function changes the initial table; see [cnb].

132 - |ct,t1,t2,t3,...,t| Combine Texts (superseding)
\ct,t1,t2,t3,...,t\ Combine Texts (save current)

The purpose of this function is to create a text, whose name is denoted by "t1" which will contain the "concatenation" of the texts whose names are denoted by "t2", "t3", ... "t".

The active form of the expression will also delete the current text named "t1" if any; the neutral form of the expression will not cause deletion of any pre-existing texts.

250 - |cwc,s1| Change Warning Character
\cwc, ... \ Change Warn. Char. (initial)

The purpose of this function is to allow the user to select other symbols for syntactic purposes than those initially defined in the language; see [qwc]. The active form of this function changes the current user tables while the neutral form changes the initial table; see [cnb].

261 - |cws,d| Change Work Space
\cws,x\

This function allows a respecification of the upper limit of space required for function execution and text storage; the neutral form is used to allow a binary (hex, octal ...) number denoted by "x" to be entered; the active form uses the decimal number system instead; see [qws].

171 - |cx,s0,s| Character to "X"

This function returns as a value a sequence of binary numbers, each preceded by the string denoted by "s0" which result from converting the character string denoted by "s"; the conversion is in the current specification for the "X" base; see [cxb, qxb, xc].

200 - |cxb,d| Change "X" Base (active)
\cxb,d\ Change "X" Base (initial)

Through use of this function, the user may change the specification of the radix (or base) for those functions which operate in the "X" base; "d" symbolizes the decimal value of the desired new radix, and this should normally be either "2" or "4" or "8" or "16"; active and neutral forms operate with respect to the "X" base in the same manner as described for the "cnb - change number base" function.

259 - |da,s0| Date

The value of this function is the current calendar date in the sequence of "day, month, and year"; each of these elements is preceded by the string of characters denoted by "s0".

The date may be either user entered (using the "sda" function) or may be automatically generated by a system clock or calendar; see [sda].

131 - |di,n1,n2,vz| Divide

The value of this function is the integer result of dividing the number denoted by "n1", by the second number denoted by "n2"; if the result of the division would be indeterminate - "n2" = zero - then the value of the expression is that string denoted by "vz".

Treatment of non numeric, signs, and null strings is identical to that described in the "ad" function.

208 - |dq,s| Define Quote

The purpose of this function is to enable the user to specify the character that is to be recognized by the scan algorithm as an unconditional protecting character. To be further defined.

- |dr,t,a,o,v| Define Relationship

To be defined and coded.

164 - |ds,d,s| Duplicate String

The value of this expression is a concatenation of that number of copies of the string "s" which is denoted by the decimal number "d".

This function may be quite properly used to "DO" something a desired number of times as illustrated below:

```
%ds,5,!%dt,x,%ad,3,%x/////=
```

This example shows a most awkward way of creating a text named "x" which contains the product of 5 and 3.

103 - |dt,t,s,d1,d2| Define Text (superseding)
\dt,t,s,d1,d2\ Define Text (save current)

This function is used to create a "text" whose name is denoted by "t", containing the string of characters symbolized by "s"; arguments denoted by "d1", "d2" are for undefined purposes.

The active form of the expression serves also to erase the text whose name is "t" if currently in the text area; the neutral form will not cause any text deletion.

173 - |dx,d,x| Decimal to "X"

The value of this function is that number which results from converting the decimal number denoted by "d" to its equivalent in the current "X" base; the argument symbolized by "x" is undefined; see [xd, cxb, qxb].

206 - |ea,t1,t2,...,t| Erase All excepting

This function is used to erase from the text area all defined texts; excepted from this erasure are the texts whose names are denoted by "t1", "t2", ... "t"; full erasure is accomplished as shown below:

```
%ea/=
```

207 - |ed,t,d1,d2,vz| Extract "D" characters

The value of this function is a string of characters extracted from the text whose name is denoted by "t"; the first character returned is that which is found distant to the right of the text divider by the number denoted by "d1", and the number of characters returned is denoted by the number "d2".

If there are absolutely no such characters available then the value of the expression is that string denoted by "vz" which is always treated actively; this is regardless of whether the original expression was used actively or neutrally.

224 - |ef,f1,f2,...,f| Erase Files

This null valued function is used to erase from auxiliary storage the files whose names are denoted by "f1", "f2", ... "f".

151 - |ep,t,p1,p2,...,p| Erase Partitions

This function serves to delete from the text named "t" any partitions that may be found in it to the right of the internal text divider.

If arguments "p1", "p2", and so on are specified, then only those partitions with values denoted by these arguments are deleted.

- |er, ... | Express Relationship

To be defined and coded.

104 - |et,t1,t2,...,t| Erase Text
\et,t1,t2,...,t\ Erase all occurences of Text

This function is used to erase from the text area texts whose names are denoted by the arguments of the expression; the active form of the expression erases only the latest version of the named texts, should more than one version exists in the text area.

249 - |etb,s| Erase Trailing Blanks

This function, which Neil Colvin insisted be made available, serves to delete from the character string symbolized by "s" all blanks which immediately precede the occurence of "new line" codes; particularly useful when reading punched cards with many trailing blank columns.

112 - |ex,f| Exit

This function is used to exit from the processor back to a system monitor; if the expression contains any arguments, such as "f" then a file is created in auxiliary storage which contains all of the user work space including variables, working area and text area.

This file is given the name denoted by "f", and may be reloaded on any other system and the action interrupted by the execution of this function is then resumed at the same point.

Successful resumption of action is subject to compatibility as determined by verification of the system authorship and version number.

226 - |fb,f,vt,vf| File Branch

The value of this function is the string symbolized by "vt" if the file whose name is denoted by "f" is to be found in external storage; if not the value of this function is that string symbolized by "vf".

137 - |fc,t,vz| Fetch Character

The value of this function is the character to be found immediately to the right of the internal text divider of the text whose name is denoted by "t"; if such a character exists then the text divider is advanced to point to the next available character - or alternatively to the end of the "text".

If no character is available then the value of the expression is that string denoted by "vz" which is always treated actively.

138 - |fdc,t,d,vz| Fetch "D" Characters

The value of this function is that decimal number of characters denoted by "d" which is to be found to the right or left of the internal divider of the text whose name is denoted by "t"; a positive number specifies to the right, and a negative number to the left; the text divider is then relocated to point to the next character available if to the right, or to the first character returned in the value string if to the left.

Should there be absolutely no characters (not even one) available for the value, then "vz" represents the string which will be returned as the value of the expression, and this value is always treated actively.

139 - |fde,t,d,vz| Fetch "D" Elements

The value of this function is that string of characters to the right or left of the text divider of the text whose name is denoted by "t" which comprises the number of elements specified by the decimal number "d"; an element is that which is found between partitions.

The divider is then moved either to the right or left to point either to the next character to be read (if to the right) or that character which corresponds to the first character of the value string (if to the left).

If absolutely no characters are returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

140 - |fdm,t,d,s,vz| Fetch "D" Matches

The value of this function is a character string taken to the right of the text divider of the text whose name is denoted by "t"; the contents of the text "t" is scanned searching for occurences of the string denoted by "s"; when that number of occurences equal to the number "d" is found then the divider is relocated to point to the next immediately following character, and the value string is that which lies between the old and the new divider locations.

If the required number of matches is not found, then the divider is not moved, and the value of the expression is that string denoted by "vz", which is always treated actively.

If the desired number is negative, then the action takes place to the left of the divider instead of to the right.

141 - |fe,t,vz| Fetch Element

The value of this function is that string of characters which is to be found immediately to the right of the current location of the divider in the text named "t" up to the next encountered partition - or end of text.

The text divider is moved to point to the immediately available next character.

If absolutely no characters are to be found, then the value is the string denoted by "vz", which is always treated actively.

142 - |ff,t,d,vz| Fetch Field

The value of this expression is that string of characters to be found to the right of the first occurence of a partition of value symbolized by "d" until the next partition is encountered, or to the end of the text named "t"; the divider is then moved to point to the next available character.

If absolutely no characters are returned, then the divider is not moved, and the value of the expression is the string symbolized by "vz" which is always treated actively.

143 - |fl,t,s,vz| Fetch Left match

The text named "t" is searched to the left of its internal divider for a string identical to that denoted by "s"; if such a string is found then the divider is moved to point to the first character of that string and the value of the expression is that group of characters to be found between the new location of the divider and its old location.

If no such matching string is found, then the divider is not moved, and the value of the expression is the string "vz" which is always treated actively.

145 - |fp,t,xl,...,x| Fetch Partition

The value of this function is the next partition to be found in the text whose name is denoted by "t"; the text divider is moved to point to the next ensuing character.

144 - |fr,t,s,vz| Fetch Right match

The text named "t" is examined starting at the current location of the text divider for a string identical to that denoted by "s"; if such a string is found, then the divider is moved to the next character which follows this string (or the end of the text) and the value returned is that string of characters which lies between the old and the new locations of the text divider.

If no such match is found, then the value of the expression is the string denoted by "vz", always treated actively, and the divider is not moved.

106 - |ft,t,s1,s2,...,s| Fetch Text

The value of this function is the contents to be found in the text named "t" to the right of its internal text divider.

Any partitions found in that portion of the text are replaced with the strings denoted "s1", "s2", ... "s", in such a manner that partitions with value "1" are replaced by "s1", value 2 with "s2" and so forth.

210 - |ftb,t,s,vz| Fetch To Break character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is found in the string symbolized by "s"; the text divider is moved to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

211 - |fts,t,s,vz| Fetch To Span character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is not found in the string symbolized by "s"; the text divider is moved to point to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

212 - |hc,s| How many Characters

The value of this function is the actual number of characters comprising the string symbolized by "s".

150 - |hm,t,s| How many Matches

The value of this function is that number of occurences of the string of characters symbolized by "s" in the text whose name is denoted by "t".

149 - |hp,t,d| How many Partitions

The value of this function is the number of partitions to be found to the right of the text divider in the text whose name is "t"; if there is no explicit third argument, then the value represents the total number of partitions; if argument three is specified, then the value is that number of partitions whose value is equal to the number symbolized by "d".

114 - |ht,t| Hide Text
\ht\ Hide all Texts

The purpose of this function is to put a screen or fence in the text area such that only those texts created after that whose name is denoted by "t" are available.

This fence is removed by execution of this function with no arguments; if it is desired to conceal all of the text area, the the neutral form is to be executed.

115 - |ic| Input Character

The value of this function is one character the source of which is the currently selected input device; this may be absolutely any one single character in the system character set.

116 - |id,d| Input "D" characters

The value of this function is a stream of characters from the current selected input device; the action is terminated when that number symbolized by "d" has been received; in the case of certain sources, an End of File will also terminate.

153 - |idt,d| Input "D" Texts

This function serves to place into the text area "texts" which are loaded from various types of auxiliary storage media and created through use of the "output text" functions.

136 - |ig,d1,d2,vt,vf| If Greater

This expression compares in an arithmetic sense the two strings denoted by "d1" and "d2"; if the value of "d1" is greater than that of "d2", then the value of the expression is that string symbolized by "vt", else the value is that string symbolized by "vf".

135 - |ii,s1,s2,vt,vf| If Identical

Strings symbolized by "s1", and "s2" are compared, if found to be absolutely identical, then the value of the expression is that string symbolized by "vt"; if not identical the value is the string symbolized by "vf".

117 - |im,s1,s2,...,s| Input to Match

The value of this expression is a stream of characters coming from the currently selected input device; termination of this stream occurs as soon as one of the strings symbolized by "s1", "s2", ... "s" has been encountered; the value includes the string that caused termination.

102 - |is,dev| Input String

The value of this function is a stream of characters coming from the currently selected input device; termination occurs as soon as the currently specified "activating character" is encountered; this activating character does not become part of the value string of the function.

152 - |it| Input Text

The purpose of this function is to place in the text area one text loaded from some auxiliary storage media, which is in the format generated through use of the "output text" functions.

213 - |iw,n| Input Wait

This function sets up an internal timer which becomes effective for the next ensuing input function, such as "is", "ic", "idc", "im"; automatic termination of the input process takes place at the end of the specified time interval even though normal terminating conditions specified for these functions may not have been met.

The duration of the time interval is specified by "n" which is in seconds.

- |lef,dev| Load External Function

To be defined and coded.

216 - |lf,s0,d1,...,d| List Files

The value of this function is a list of the names of the files in auxiliary storage; each file name is preceded by the string symbolized by "s0".

Additional arguments are to be defined.

- |lr, ... | List Relationship

To be defined and coded.

105 - |lt,s0,d1,d2,...,d| List Texts

The value of this function is a list of the names of the texts to be found in the text area; each name is preceded by the string symbolized by "s0".

Arguments symbolized by "d1", "d2", ... have not been defined.

110 - |mc,d| Multi-partition Character

The value of this function is the "multi-partition character" whose value is denoted by the decimal number "d".

146 - |md,t,d| Move Divider to pos. "d"
\md,t,d\ Move Divider "d" increments

This function is used to move the text divider in the text whose name is denoted by "t". Active form serves to move the divider the number of positions from the left end of the text if "d" is positive, or from the right end if negative; the neutral form is used to advance or retreat the divider from its current location that number of positions specified by the number "d".

It should be noted that these moves are positional and do not distinguish any difference between partitions and characters; this is the only function available to position the text divider other than immediately preceding a character.

109 - |mt,t,s1,s2,...,s| Multi-part Text all matches
\mt,t,s1,s2,...,s\ Multi-part Text next match

This function serves to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", "s" with multi-partitions of values respectively "1", "2", ...; the neutral form of this function only replaces the first occurence of these strings found in the text.

130 - |mu,n1,n2,vz| Multiply

The value of this function is the arithmetic product of the numbers symbolized by "n1", and "n2". The same rules as to sign and non numeric matter apply for this function as they do for the "ad" function.

Should there be an overflow condition arise as a result of the execution of this function, then the value of the expression is that string symbolized by "vz".

111 - |ni,vt,vf| Neutral Implied

This function returns the string symbolized by "vt" if the last previously executed implied "fetch text" was neutral. If that last previously executed implied "fetch" was active, then the value of this function is that string symbolized by "vf".

188 - |not,x| Not (complement) the bits

The value of this function is the complement of the binary number (in the current "X" base) symbolized by "x".

209 - |nu,s1,s2,...,s| Null

The purpose of this function is to cause execution of the function whose name is denoted by "s1", with appropriate arguments "s2", ... "s"; that function is executed but any resulting value string is suppressed.

246 - |oj,s,sl,d,s2| Output Justified lines

To be defined and coded.

248 - |op,s,sl,d,s2| Output Paddded lines

To be defined and coded.

186 - |or,x1,x2| Or the bits

The value of this function is the logical or of the two binary number strings "x1" and "x2" expressed in the current "X" base.

101 - |os,s| Output String

This function causes the selected output device to output the string of characters symbolized by "s".

154 - |ot,t1,t2,...,t| Output Texts

This function outputs to the selected channel, the texts whose names are denoted by "t1", "t2", ... "t", in a format which preserves all of the internal conditions of the text; this includes location of the internal text divider, and text partitions if any.

108 - |pc,d| Partition Character

The value of this function is a "partition character" of value symbolized by the decimal number "d".

174 - |pl,s1,s2,...,s| Plot

This is a general purpose plotter or display control function; details may vary in its use as a function of the system in which it is used. Certain subfunctions as described below have been standardized:

|pl,lf,s0|	List of Plotter subfunctions
|pl,as,port,time|	Assign output port and time delay
|pl,nm|	Output plotter control information
|pl,vm|	Return as value control information
|pl,cal|	Set up for "Calcomp" plotter
|pl,tek|	Set up for "Tektronix" display
|pl,mq,x|	Rotate to quadrant "x"

162 - |ps,d,s1,s2| Pad String

The value of this function is the string "s2" to which is added either to its right or to its left enough duplications of string "s1" so that the total size of the value string becomes equal to the number symbolized by "d"; positive values of "d" pad to its right, and negative values to its left.

107 - |pt,t,s1,s2,...,s| Partition Text all matches
\pt,t,s1,s2,...,s\ Partition Text next match

The purpose of this function is to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", ... "s" with partitions identified by numbers which correspond to the position of the strings in the expression; that is to say "s1" is replaced by a partition value "1", "s2" by partition value "2" &c.

In its active form, this function serves to replace all occurences of the desired strings; in its neutral form only the first occurence found is so replaced for each of the specified strings.

Action takes place starting at the current location of the internal text divider which is not moved by this function.

A particular example of interest might be:

&pt,animal,dog,dog,dog,dog/=

This example would in the text whose name was animal, replace the first match with "dog" with partition value "1", the second, "2", the third with "3" and the fourth with "4".

196 - |qfc,s0| Query Fill Character schema

The value of this function is the current specification for the optional fill character output immediately after each "new line"; the parameters are each preceded by the string symbolized by "s0"; see [cfc].

194 - |qin,s0,t1,t2,...,t| Query Id Number

To be defined and coded.

197 - |qld,t| Query Left of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

192 - |qll| Query Line Length

The value of this function is the current specification for the line length; this specification is changed through use of the "cll" function.

134 - |qnb| Query Number Base

The value of this function is the current value of the radix for arithmetic operations and functions.

202 - |qof| Query Over Flow conditions

The value of this function are characters which may be used to identify the cause of an interrupt or overflow condition.

167 - |qp,t| Query Partition

The value of this function is the decimal identification of the next partition to be found in the text whose name is denoted by "t"; the text divider is not moved by this function.

267 - |qpc,s0,t1,t2,...,t| Query Protection Class

To be defined and coded.

198 - |qrd,t| Query Right of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

204 - |qro| Query Rub Out char. schema

The value of this function describes the current specification for the use of the "rub out"; see [cro].

205 - |qta| Query Text Area used

The value of this function is representative of the amount of space consumed in the text area by user defined texts.

251 - |qwc,a2,a1,...,a| Query Warning Characters

The value of this function is a list of the currently specified and functional warning characters of the language.

262 - |qws| Query Work Space
\qws\

The value of this function is the upper limit of the user work space; active form of this expression yields the value as a decimal number; neutral form yields the value in the current "X" base.

201 - |qxb| Query "X" Base

The value of this function is the currently set radix for the functions which operate in a "binary" or "X" base.

215 - |ra,d,s1,s2,s3,...,s| Return Argument

The value of this function is that string symbolized by one of "s1", "s2", "s3", ... "s" whose position corresponds to the value of the decimal number symbolized by "d".

263 - |rcp,d1,d2,s| Return Character Picture

To be coded.

166 - |ri| Restart Initialized

Execution of this function causes immediate termination of any unexecuted scripts, and causes return to the idling condition, reinitializing all normally changed user specifications.

245 - |rj,s,s1,d,s2| Return Justified lines

To be defined and coded.

252 - |rn,n| Random Number

The value of this function is a number ranging between 0 and "n", randomly selected through a computational algorithm; see [srn].

189 - |rot,d,x| Rotate the bits

The value of this function is that binary number which results from a rotation of the binary number "x" expressed in the current "X" base; the number of bits rotated is specified by the decimal number "d"; the direction is

clockwise if the number is positive, and counterclockwise if negative.

247 - |rp,s,sl,d,s2| Return Padded lines

To be defined and coded and coded.

165 - |rr,sl| Return to Restart

This function causes immediate cessation of execution of any unexecuted script, forces a return to the idling level of the processor and then returns the value of the string symbolized by "sl"; if that string is an executable expression, then said expression is executed.

163 - |rs,s| Reverse String

The value of this function is the string of characters symbolized by "s" reversed, end for end.

228 - |saf,dev| Select All File function dev.

To be defined.

158 - |sar| "Auto Return" on line feed
\sar\ no Auto Return on line feed

Execution of this function serves to enable (if active) or disable (if neutral) the automatic generation of a "carriager return" code when a "new line" code is output.

260 - |sda,da,mo,yr| Set Date

This function enables the user to set into the system the current (r any other) desired date.

199 - |sem,dev| Set "Echoplex" Mode active
\sem,dev\ "Echoplex" Mode inactive

This function enables character echo from the system if executed actively; if neutrally, then echo generation is suppressed.

222 - |sf,f,tl,t2,...,t| Store File

This function is used to place into auxiliary storage under the file name denoted "f", those texts whose names are denoted by "tl", "t2", ... "t"; on completion of this action, the named texts are erased from the text area.

If no texts are named, then the assumption is made that the entire text area is to be placed into auxiliary storage.

157 - |sfd,fun,dev| Specify Function Device

To be defined and coded.

190 - |sh,d,x| Shift the bits

The value of this function is that binary number string which results from a logical shift to the right or left of the string whose binary value, expressed in the current "X" base is symbolized by "x"; the number of shifts is specified by the decimal number "d" and the direction by its sign, positive to the right and negative to the left.

253 - |srn,n| Seed Random Number

Through use of this function the automatic generation of random numbers is initiated; the user seed number is symbolized by "n" in the above expression.

258 - |sti,tl,t2,t3| Set Time

Through use of this function the correct current (or incorrect if desired) time of day is established in the system.

129 - |su,nl,n2,...,n| Subtract

The value of this function is the result of subtracting the number symbolized by "n2" from that symbolized by "nl"; all rules indicated for sign and non numeric matter in the "ad" function apply.

231 - |sw,sl,s2,s3,...,s| Switches

Special system dependent function for user definition.

232 - |sy,sl,s2,...,s| System functions

Special system dependent function for user definition.

127 - |tb,t,vt,vf| Text Branch

If the text whose name is denoted by "t" is to be found in the text area, then the value of the function is that string symbolized by "vt", else the value is string symbolized by "vf".

257 - |ti,sl,s2| Time

The value of this function is the current time of day derived from a system clock, in the format HH MM SS, where HH represent Hours, MM represent Minutes and SS seconds; these elements are separated in the value string by the string symbolized by "sl".

125 - |tm,d| Trace Mode activated
\tm\ Trace Mode deactivated

This function is used to enable step by step execution of the scripts and functions of the language; once activated each step is initiated by depression of the "new line" key ("line feed") or a specified number of steps may be specified by the number symbolized by "d".

124 - |tma| Trace Mode All activated
\tma\ Trace Mode All deactivated

Active execution enables full trace action; neutral execution allows display only of expressions about to be executed.

168 - |tr,t,s| Trim

This function is used to replace multiple adjacent occurences of the character symbolized by "s" with a single such occurence in the text named "t".

218 - |uf,f,tl,t2,...,t| Update File

This function combines the action of "erase file" and "store file" by first storing a file successfully before erasing the old file version.

169 - |ut,cc| User Trap active
\ut\ User Trap inactive

Activating of this trap through use of this function will cause an automatic execution of a text (or function) whose name is denoted by "cc"; this automatic execution takes

place whenever the user attempts to make use of system interrupt capabilities to escape out of executing "scripts" when it is desired that such escape be inhibited.

118 - |vt,t1,t2,...,t| View Texts

This function enables examination of defined texts, indicating visually the location of internal text divider, partitions and their values, as well as multipartitions.

181 - |wc,s1,s| Write Characters

This plotter or display function generates a graphic display from a font of vectors named "s1"; the string symbolized by "s" is the text string to be plotted or displayed.

175 - |wi,xn1,yn1| Write Initialize

This null valued function serves to create a reference value for the current position of the display pointer; the "x" and "y" coordinates of this position are set equal to the decimal integer strings found in the second and third arguments respectively.

179 - |wl| Width Left

The value of this function is the incremental value of the width of a character being plotted to the left of its center of gravity.

178 - |wr| Width Right

The value of this function is the incremental value of the width of a character being plotted to the right of its center of gravity.

180 - |ws,xn1,yn1,...,xn,yn| Write Straight Lines

This function causes the generation of the appropriate control information to cause the display of one or more line segments starting at the current position of the display pointer. Successive points are referenced by consecutive incremental x and y argument pairs. The x and y components are expressed as decimal integer strings. The sign of the number indicates direction.

Non numeric matter preceding the "x" value causes certain auxiliary functions to take place as follows:

```
U       "pen up" or invisible vector
S       Scale change for "x" and "y"
A       Absolute position vector pair
I       Incremental vector pair
Q       Quadrant rotation
         0 - 3 clockwise, 4 - 7 counter clockwise
W       Character width information
         "x" left and "y" right of center
```

```
{}~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
{}  %dt,rect,(,8,0,0,5,-8,0,0,-5)/=
{}  %ws%rect//=
{}
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

176 - |wx| Write "X" displacement

The value of this function is equal to the algebraic difference between the initial value "X" of the display pointer position and the sum of any subsequent incremental displacement values.

177 - |wy| Write "Y" displacement

The value of this function is equal to the algebraic difference between the initial value "Y" of the display pointer position and the sum of any subsequent incremental displacement values.

An effective way to return the display pointer to its original location is illustrated as follows:

```
{}~~~~~~~~~~~~~~~~~~~~~~
{}  %ws,U%wx/,%wy//
{}
~~~~~~~~~~~~~~~~~~~~~~~~
```

Assuming it is desired to draw a rectangle,8 units wide, and 5 units high:

170 - |xc,x1,x2,...,x| "X" to Character

The value of this function is a string of characters whose value is represented by the numbers in "X" base symbolized by "x1", "x2", ... "x".

271 - |xcf,s,x| eXperimental Change Function

This function is used to assign a user defined machine adress symbolized by the "X" base number "x" for the built in function whose mnemonic is denoted by "s".

172 - |xd,x| "X" to Decimal

The value of this function is the decimal equivalent of the number in the current value of the "X" base symbolized by "x".

255 - |xi,port| eXperimental Input

The value of this function is that number in the current "X" base which results from an attempted input from the port whose designation in the "X" base is symbolized by "port".

123 - |xj,x| eXperimental Jump

Execution of this function cause a jump to the memory location symbolized in the "X" base by "x"; a return from programs encountered at that location will cause normal reentry into the system scanner.

256 - |xo,x,port| eXperimental Output

Execution of this function enable output of the "X" base value of "x" to the port whose "X" base id is symbolized by "port".

270 - |xqf,s| eXperimental Query Function

The value of this function is the machine adress in "X" base radix of the entry point for the function whose mnemonic is denoted by the string "s".

119 - |xr,x| eXamine Register

The value of this function is the contents of the memory location symbolized in the current "X" base by "x".

121 - |xrp,x| eXamine Register Pair

The value of this function is the combined contents of the two memory locations, whose first location is symbolized in the current "X" base by "x".

120 - |xw,x1,x2| eXperimental Write in reg.

Use of this function causes writing the value "x1" into the memory location "x2"; both of these are in the current "X" base.

122 - |xwp,x1,x2| eXperimental Write reg. Pair

Use of this function causes writing of the double word data symbolized in the "X" base by "x1" in the memory location whose first word is identified by "x2".

126 - |yt,t,s,vt,vf| Ys There

This function searches the text whose name is denoted by "t" for an exact match with the string symbolized by "s"; if such a match is found, then the value of the expression is that string symbolized by "vt", else it is that string symbolized by "vf".

182 - |zd,r,v-,v0,v+| "Z" reg. Decrement and branch

Execution of this function causes the contents of a special register identified by "r" to be decreased by a count of one.; if the resulting contents of that register are greater than zero, then the expression value is that string symbolized by "v+"; if equal to zero, then the value is symbolized by "v0", and if less than zero the value becomes the string symbolized by "v-".

183 - |zi,r,v-,v0,v+| "Z" reg. Increment and branch

The action of this function is identical to that described for "zd", except that the designated register "r" is increased by a count of one prior to testing.

184 - |zq,r| "Z" reg. Query

The value of this function is the current contents of the special register designated by "r"; execution of this function does not cause any change in said register.

185 - |zs,r,n| "Z" reg. Set

This function is used to preset designated special register "r" to the number symbolized by "n".

SAM 76 – Setting up the System

There are two versions available:

Availability

a. Z80 - little less than 8K
   Uses RST0 thru RST6

b. 8080 - about 9 K
   Does not use any RST locations

In addition the source code is available from a number of ACNJ members if special adressing requirements are encountered or if it is desired to configurate a system with fewer functions, or different use of the RST locations.

Except as indicated above both versions are virtually identical, and the information that follows applies to both of these versions; in consequence user defined texts and procedures may be successfully moved between systems using either of these two versions.

Possible incompatibility may be ascertained through use of the SAM76 function which reveals the version number thus:

```
{}~~~~~~~~~~~~~~~~~
{}  %@@n/=21
{}
~~~~~~~~~~~~~~~~~~~
```

reveals the version number of the "romable object code".

a. Romable program loads at ^h8000.

b. "Once Only" start at ^h8000 initializes variables.

c. Restarts may be made at 8003 or 0.

d. System and user variables start at ^h400

e. Workspace is above ^h600.

2 Memory utilization

I/O specifications, and user options:

a. APPLE/ZAPPLE I/O Conventions apply.

b. Normally initialized to seek and take available RAM.

System Differences

c. Special use of control codes is as follows:

^C	Return to Monitor
^X	Cancel current material typed in
DEL	Rubout (changeable by user)
^N	Shift Out to alternate character set
^O	Shift In back to regular character set.

The Line Feed key is viewed by SAM76 as being equivalent to the "NULIN" code, as specified by ASCII; use of this key serves to go to the beginning of the next line - combining the functions of "Carriage Return and Line Feed". The Carriage Return serves only to return to the beginning of the same line; this permits underlining &c.

Modifying SAM76 tables and options:

There are two levels of modifications which the user may apply in the process of tailoring the SAM76 code to suit his system and needs:

4 Fiddler's Cuide

a. Permanent.

b. Local or temporary.

The type of changes are identical but the location of the table to be changed and the time at which the table is to be changed are different.

Effectively there is a main table of variables and options located in the "romable" or protected section of code; after having executed a "GO" to ^h8000 this table is copied into the user work space starting at location ^h40C.

After this initial entry at ^h8000 entry should be made only at ^h8003 or at location 0 if the user wishes to preserve any changes made in the duplicate table at ^h40C either with monitor facilities or through the execution of SAM76 functions which are available to establish a variety of user options.

Typical Permanent Changes:

Normally changes in the table at ^h800C should be limited to necessary respecification of I/O adresses, and in some special cases to the location of the beginning and or end of the user work space.

5 Useful Fiddles

In addition the user may make permanent changes to such parameters as the maximum length of line, and the specification of the number of fill characters after each "carriage return/line feed". (Note Line Feed is the proper character used for this combined function).

Table 1 lists the primary portion of the table at ^h800C with corresponding adresses of its duplicate at ^h40C.

Table 2 lists locations that may be of interest to the user as candidates for permanent changes; since these locations may all be changed within the framework of the SAM76 language, it is suggested that changes be made first in that manner, and that the modified duplicate table at ^h40C be examined before making the desired changes in the permanent table.

TABLE 1

```
 .SBTTL /Variables & once only code/
 .PAGE
;[VAR1]
VAR1:   .BLKB   1       ; [80/4-0C] JMP
BOOT:   .BLKW   1       ; [80/4-0D] Monitor Exit
MSCI:   .BLKB   3       ; [80/4-10] Console Input
MSRI:   .BLKB   3       ; [80/4-13] Channel 2 Input
MSCO:   .BLKB   3       ; [80/4-16] Console Output
MSPO:   .BLKB   3       ; [80/4-19] Channel 2 Output
MSLO:   .BLKB   3       ; [80/4-1C] Channel 3 Output
MSCSTS: .BLKB   3       ; [80/4-1F] Console Status
MSCHK:  .BLKB   3       ; [80/4-22] IO Check
MSSET:  .BLKB   3       ; [80/4-25] IO Set
MSMCK:  .BLKB   3       ; [80/4-28] Memory Check
;
$232:   .BLKB   3       ; [80/4-2B] %sy,x1,x2,.../
$231:   .BLKB   3       ; [80/4-2E] %sw,x0,.../
        .BLKB   3       ; [80/4-31] spare
;
;
OR2:    .BLKW   1       ; [80/4-33] start active zone
TOPMAX: .BLKW   1       ; [80/4-35] text area limit
TOPLOC: .BLKW   1       ; [80/4-37] future3
SPARE4: .BLKW   1       ; [80/4-39] future4

;
```

Notes with reference to Table 1:

a. The adresses indicated are those of the actual locations which would be changed as required to suit individual operating system requirements.

b. If it is desired to disable any of the monitor functions which would be adressed via the vectors of Table 1, replace the JMP with a RET - (for instance one might wish to disable the automatic RAM grabber by putting a "RET" at ^h8027" - If this is done it becomes necessary to actually stipulate the adress of the end of user workspace by entering said adress at ^h8037).

c. $232, and $231 are locations of jump vectors for the SAM76 functions "sy" and "sw" respectively. These are intended for user applications; if not required it is wise to change the adresses for these functions at ^h802B and ^h802E respectively to the adress found at ^h8031 - this causes the message <nav-xxx> to be returned should these two functions be tested.

d. Location labelled "TOPMAX" is usually shown as 00-00; this in combination with the availability of the MEMCHECK function of the ZAPPLE monitor triggers the automatic determination of the upper limit of user work space. A non zero value at that location takes precedence and serves to restrict the user work space to that specified.

e. Location labelled "OR2" defines the beginning of the user work space; this may be changed if the user wishes to have clear space for his own purposes at it's normally initialized value of 05FE.

Caution should be exercised in changing OP2 and TOPMAX so that TOPMAX be greater than OR2 by at least ^h400 bytes.

TABLE 2 (is continuation of TABLE 1)

```
TIR:    .BLKW   1 ; 0
TO:     .BLKW   1 ; TCO
TI:     .BLKW   1 ; TCI
TT:     .BLKW   1 ; TCT
TR:     .BLKW   1 ; TCR
TE:     .BLKW   1 ; TIH
RIX:    .BLKB   1 ; N12
BIX:    .BLKB   1 ; N10
MCH:    .BLKB   1 ; "=" EQUAL
TCE:    .BLKW   1 ; FNP1
CE:     .BLKW   1 ; SCE
TM2:    .BLKW   1 ; 0
TC3:    .BLKW   1 ; 0
TCECT:  .BLKW   1 ; 0
TCEAN:  .BLKB   1 ; 0
RUBOT:  .BLKB   1 ; N177
RUBO1:  .BLKB   1 ; N74
RUBO2:  .BLKB   1 ; N76
NULCH:  .BLKB   1; null character
NULCT:  .BLKB   1; filler count
PLIN:   .BLKB   1; line length
```

Notes regarding changes in Table 2:

a. PLIN contains the number of characters on a line at which SAM76 is required to automatically generate a "NULINE" on the console device. This is usually initialized in the permanent table at 72 decimal (^h48), and is changed in the duplicate table by using the "cll - Change Line Length" function thus:

```
{}~~~~~~~~~~~~~~~~~~
{}  %cll,64/=
{}
~~~~~~~~~~~~~~~~~~
```

would set line length at decimal 64; the permanent table may be changed prior to being placed into protected memory using the neutral form of this function.

b. NULCT, and NULCH locations are used to specify the number and the type of "null" character the user wishes to have automatically generated on the console device when a "NULIN" is output to it. This is usually initialized at zero count, and ASCII-0 for the character; to change this requires the use of the "cfc - Change Fill Character" function thus:

```
{}~~~~~~~~~~~~~~~~~~~~
{}  %cfc,*,4/=
{}
~~~~~~~~~~~~~~~~~~~~~~
```

puts 4 asterisks at beginning of each new line; the neutral form of the function is used to change the permanent table.

c. RUBOT, RUBO1, and RUBO2 are used to specify the character to be used for character deletion purposes, and the symbols that are desired to surround the string of consecutively deleted characters; initially this is set to be the ASCII - DEL code and the surrounding symbols are the < and > brackets. Changing this makes use of the "cro - Change Rub Out" function thus:

```
{}~~~~~~~~~~~~~~~~~~~
{}  %cro,$AB/=
{}
~~~~~~~~~~~~~~~~~~~
```

sets $ to be the deleting character and the deleted string is bracketed by A and B; the neutral form of this function may be used to change the permanent table.

d. MCH is the location of the character used to terminate entry; this character is known as the "Activator". Initially this is set to be the = sign, and is changed via use of the "ca - Change Activator" function thus:

```
{}~~~~~~~~~~~~~~~
{}  %ca,Z/=
{}
~~~~~~~~~~~~~~~
```

changes activator to be "Z"; use the neutral form to change the permanent table.

e. RIX and BIX are the current specifications for the arithmetic and logical or binary bases respectively. These locations are initialized to provide base 10 for arithmetic and base 16 or hex for logical functions. Within the framework of SAM76 they are changed using the "cnb - Change Number Base" and "cxb - Change X Base" functions; for example:

```
{}~~~~~~~~~~~~~~~~~~
{}  %cnb,7/=
{}
~~~~~~~~~~~~~~~~~~
```

sets arithmetic base to be 7.

```
{}~~~~~~~~~~~~~~~~~
{}  %cxb,8/=
{}
~~~~~~~~~~~~~~~~~~
```

sets logical or binary base to octal.

The neutral form of these functions may be used to change the permanent initialization table.

SAM76 - Loading the object code tape

The two versions are available in compressed binary form for which a special loader is required; this loader is usually to be found in normal hex loader format ahead of the binary.

The normal procedure to load is as follows:

a. Load the hex formatted loader;

b. The loader should be at ^h1000 to ^h475; simply go to ^h1000 if using APPLE/ZAPPLE monitors, otherwise change the calls to F006 to reflect your own reader input routine before running.

c. If using APPLE/ZAPPLE just do a G8000 to start SAM76, otherwise make the appropriate changes as required with reference to Table 1 first.

If you are using a machine with output lights at port FF, then the progress of loading will be displayed. As each block is successfully loaded a check sum will appear for a few seconds then the adress of the new block being loaded will be displayed. If the check sum remains, an error is indicated and then it is necessary to stop the reader, back up the tape to the beginning of the previous block, start the reader and G400 again.

d. Test SAM76 by getting a list of available built in functions by typing in the following:

```
&@@f, /=
```

e. Three types of anomalies are flagged by output to the console:

T	Time out no tape or jammed tape
M	Memory error
P	Parity error in block

f. For the afficionado format is described below:

16 Bytes	^h80 or ^o200
SOH	^h1
DEL	^hFF or ^o377
1 byte	Number of data bytes (0 means 256)
2 bytes	Load adress (Low then High)
variable	data bytes
2 bytes	Check Sum

LOADIT - loader

```
.RADIX  8       ; added by editor

;
 .TITLE /loadit/

        .LOC    002000  ; ^h1000

ZAPMON= 1
ZAPPLE= 170000  ; ^hF000

SENSE=  377     ; ^hFF

;Control Codes
CCDLE=  20      ; Data Link Escape
CCENQ=  5       ; Enquiry
CCEM=   31      ; End Message
CCSOH=  1       ; Start of Header
CCETB=  27      ; End of Tx Block
CCACK=  6       ; Acknowledge
CCNAK=  25      ; Neg. Acknowledge
CCSTX=  2       ; Start Text

;
 .DEFINE SKIP2[R]=
        [.BYTE  001![[[R]&6]<3]]

GETLDR: CALL    GETBY
        CPI     CCSOH
        JZ      SOHFND
GETLDX: CPI     CCETB
        JZ      RSEXIT
        CPI     170
        JNZ     GETLDR
        CALL    GETAD
        PCHL    ;all done

SOHFND: CALL    GETBY
        CPI     377
        JNZ     GETLDX

LDR377: CALL    GETBY
        MOV     C,A
        MVI     E,0
        MOV     D,E
        CALL    GETAD

LOAD:   CALL    GETBY

RCVOK:  MOV     M,A
        MOV     B,M
        CMP     B
        JNZ     MEMERR
        INX     H
        DCR     C
        JNZ     LOAD
```

```
CALL    PARTST

JMP     GETLDR

NIXNIX: MVI     C,"?"
        SKIP2   D
MEMERR: MVI     C,"M"
        SKIP2   D
PARERR: MVI     C,"P"
        SKIP2   D
TIMERR: MVI     C,"T"

EREXIT: CALL    CO
RSEXIT: JMP     RESTART

;
GETAD:  CALL    GETBY
        MOV     L,A
        CALL    GETBY
        MOV     H,A
        CMA
        OUT     SENSE
        RET

GETBY:  CALL    RI
        JC      TIMERR
        PUSH    PSW
        ADD     E
        MOV     E,A
        JNC     GETBYR
        INR     D
GETBYR: POP     PSW
        RET

;
PARTST: PUSH    D
        CALL    GETAD
        POP     D
        MOV     A,H
        CMP     D
        RNZ
        MOV     A,L
        CMP     E
        RET

;

        .END            ;Pseudo op added by editor

;
```

Quick Reference Information for the SAM 76 Language

Mnemonics

An attempt was made, in the selection of the mnemonics of the resident functions, to be consistent in the use and meaning of words or terms represented by the function names; generally, whenever reasonable, letters used in the function names may be expanded as follows:

letter	position 1	positions 2 and/or 3	
a	alphabetic	all	activator
b	bring	branch	base
c	change	character	c3
d	define	decimal	divider
e	erase	element	e3
f	fetch	file	field, function
g	gl	greater	g3
h	how many	h2	h3
i	input, if	identical	initialize
j	jl	jump	j3
k	kl	k2	k3
l	list	left	length
m	ml	match	mode
n	neutral	number	n3
o	output	o2	o3
p	partition	p2	p3
q	query	q2	q3
r	return	right	register
s	set, select	string	space
t	trace	text	t3
u	update, user	u2	u3
v	view	v2	v3
w	write	warning	work
x	eXperimental	"X" number base	x3
y	yl	y2	y3
z	Zpecial	z2	z3

In the above table only the more frequently or major terms are indicated; positions marked with a letter number pair are for possible reference.

It should be noted that the meaning of the words used in the above table are consistent; that is to say that the same connotation may be applied - for instance "text" always means; "a named string of symbols to be found in the text area".

Quick reference resident function list

```
238 - |@f,s0|              wh@ are Functions
239 - |@n|                 wh@ is processor ser. Number
237 - |@t|                 wh@ is processor Title
159 - |ab,s1,s2,vt,vf|     Alphabetic Branch
128 - |ad,n1,n2,n3,...,n|  Add
160 - |ai,s0,s1,s2,...,s|  Alphabetic Insertion
187 - |and,x1,x2|          And the bits
161 - |as,s0,s1,s2,...,s|  Alphabetic Sort
220 - |bf,f,vz|            Bring File
113 - |ca,s|               Change Activator (current)
      \ca,s\               Change Activator (initial)
195 - |cfc,d1,s|           Change Fill Character schema
      \cfc,d1,s\           Change Fill Char. (initial)
193 - |cin,t1,d1,...,t,d|  Change Id Number
148 - |cld,t|              Characters Left of Divider
191 - |cll,d|              Change Line Length (active)
      \cll,d\              Change Line Length (initial)
133 - |cnb,d|              Change Number Base (active)
      \cnb,d\              Change Number Base (initial)
```

```
266 - |cpc,t1,d1,...,t,d|     Change Protection Class
147 - |crd,t|                 Characters Right of Divider
203 - |cro,s1|                Change Rub Out char. schema
      \cro,s1\                Change Rub Out (initial)
132 - |ct,t1,t2,t3,...,t|    Combine Texts (superseding)
      \ct,t1,t2,t3,...,t\     Combine Texts (save current)
250 - |cwc,s1|                Change Warning Character
      \cwc, ... \             Change Warn. Char. (initial)
261 - |cws,d|                 Change Work Space
      \cws,x\
171 - |cx,s0,s|               Character to "X"
200 - |cxb,d|                 Change "X" Base (active)
      \cxb,d\                 Change "X" Base (initial)
259 - |da,s0|                 Date
131 - |di,n1,n2,vz|           Divide
208 - |dq,s|                  Define Quote
    - |dr,t,a,o,v|           Define Relationship
164 - |ds,d,s|               Duplicate String
103 - |dt,t,s,d1,d2|         Define Text (superseding)
      \dt,t,s,d1,d2\          Define Text (save current)
173 - |dx,d,x|               Decimal to "X"
206 - |ea,t1,t2,...,t|       Erase All excepting
207 - |ed,t,d1,d2,vz|        Extract "D" characters
224 - |ef,f1,f2,...,f|       Erase Files
151 - |ep,t,p1,p2,...,p|     Erase Partitions
    - |er, ... |            Express Relationship
104 - |et,t1,t2,...,t|       Erase Text
      \et,t1,t2,...,t\        Erase all occurences of Text
249 - |etb,s|                Erase Trailing Blanks
112 - |ex,f|                 Exit
226 - |fb,f,vt,vf|           File Branch
137 - |fc,t,vz|              Fetch Character
138 - |fdc,t,d,vz|           Fetch "D" Characters
139 - |fde,t,d,vz|           Fetch "D" Elements
140 - |fdm,t,d,s,vz|         Fetch "D" Matches
141 - |fe,t,vz|              Fetch Element
142 - |ff,t,d,vz|           Fetch Field
143 - |fl,t,s,vz|           Fetch Left match
145 - |fp,t,x1,...,x|       Fetch Partition
144 - |fr,t,s,vz|           Fetch Right match
106 - |ft,t,s1,s2,...,s|    Fetch Text
210 - |ftb,t,s,vz|          Fetch To Break character
211 - |fts,t,s,vz|          Fetch To Span character
212 - |hc,s|                How many Characters
150 - |hm,t,s|              How many Matches
149 - |hp,t,d|              How many Partitions
114 - |ht,t|                Hide Text
      \ht\                    Hide all Texts
115 - |ic|                   Input Character
116 - |id,d|                 Input "D" characters
153 - |idt,d|                Input "D" Texts
136 - |ig,d1,d2,vt,vf|      If Greater
135 - |ii,s1,s2,vt,vf|      If Identical
117 - |im,s1,s2,...,s|      Input to Match
102 - |is,dev|              Input String
152 - |it|                  Input Text
213 - |iw,n|                Input Wait
    - |lef,dev|            Load External Function
216 - |lf,s0,d1,...,d|      List Files
    - |lr, ... |            List Relationship
105 - |lt,s0,d1,d2,...,d|   List Texts
110 - |mc,d|                Multi-partition Character
146 - |md,t,d|              Move Divider to pos. "d"
      \md,t,d\                Move Divider "d" increments
109 - |mt,t,s1,s2,...,s|    Multi-part Text all matches
      \mt,t,s1,s2,...,s\      Multi-part Text next match
130 - |mu,n1,n2,vz|         Multiply
111 - |ni,vt,vf|            Neutral Implied
188 - |not,x|               Not (complement) the bits
209 - |nu,s1,s2,...,s|      Null
246 - |oj,s,s1,d,s2|        Output Justified lines
248 - |op,s,s1,d,s2|        Output Paddded lines
186 - |or,x1,x2|            Or the bits
101 - |os,s|                Output String
154 - |ot,t1,t2,...,t|      Output Texts
108 - |pc,d|                Partition Character
174 - |pl,s1,s2,...,s|      Plot
162 - |ps,d,s1,s2|          Pad String
107 - |pt,t,s1,s2,...,s|    Partition Text all matches
      \pt,t,s1,s2,...,s\      Partition Text next match
196 - |qfc,s0|              Query Fill Character schema
194 - |qin,s0,t1,t2,...,t|  Query Id Number
197 - |qld,t|               Query Left of Divider
192 - |qll|                 Query Line Length
134 - |qnb|                 Query Number Base
202 - |qof|                 Query Over Flow conditions
167 - |qp,t|                Query Partition
267 - |qpc,s0,t1,t2,...,t|  Query Protection Class
198 - |qrd,t|               Query Right of Divider
204 - |qro|                 Query Rub Out char. schema
205 - |qta|                 Query Text Area used
251 - |qwc,a2,a1,...,a|     Query Warning Characters
262 - |qws|                 Query Work Space
      \qws\
201 - |qxb|                 Query "X" Base
215 - |ra,d,s1,s2,s3,...,s| Return Argument
263 - |rcp,d1,d2,s|         Return Character Picture
166 - |ri|                  Restart Initialized
245 - |rj,s,s1,d,s2|        Return Justified lines
252 - |rn,n|                Random Number
189 - |rot,d,x|             Rotate the bits
247 - |rp,s,s1,d,s2|        Return Padded lines
165 - |rr,s1|               Return to Restart
163 - |rs,s|                Reverse String
228 - |saf,dev|            Select All File function dev.
158 - |sar|                 "Auto Return" on line feed
      \sar\                   no Auto Return on line feed
260 - |sda,da,mo,yr|       Set Date
199 - |sem,dev|            Set "Echoplex" Mode active
      \sem,dev\               "Echoplex" Mode inactive
222 - |sf,f,t1,t2,...,t|   Store File
157 - |sfd,fun,dev|        Specify Function Device
190 - |sh,d,x|             Shift the bits
253 - |srn,n|              Seed Random Number
258 - |sti,t1,t2,t3|       Set Time
129 - |su,n1,n2,...,n|     Subtract
231 - |sw,s1,s2,s3,...,s|  Switches
232 - |sy,s1,s2,...,s|     System functions
127 - |tb,t,vt,vf|         Text Branch
257 - |ti,s1,s2|           Time
125 - |tm,d|                Trace Mode activated
      \tm\                    Trace Mode deactivated
124 - |tma|                 Trace Mode All activated
      \tma\                   Trace Mode All deactivated
168 - |tr,t,s|             Trim
218 - |uf,f,t1,t2,...,t|   Update File
169 - |ut,cc|              User Trap active
      \ut\                    User Trap inactive
118 - |vt,t1,t2,...,t|     View Texts
181 - |wc,s1,s|            Write Characters
175 - |wi,xn1,yn1|         Write Initialize
179 - |wl|                 Width Left
178 - |wr|                 Width Right
180 - |ws,xn1,yn1,...,xn,yn|  Write Straight Lines
176 - |wx|                 Write "X" displacement
177 - |wy|                 Write "Y" displacement
170 - |xc,x1,x2,...,x|     "X" to Character
271 - |xcf,s,x|           eXperimental Change Function
172 - |xd,x|               "X" to Decimal
255 - |xi,port|           eXperimental Input
123 - |xj,x|              eXperimental Jump
256 - |xo,x,port|         eXperimental Output
270 - |xqf,s|             eXperimental Query Function
119 - |xr,x|              eXamine Register
121 - |xrp,x|             eXamine Register Pair
120 - |xw,x1,x2|          eXperimental Write in reg.
122 - |xwp,x1,x2|         eXperimental Write reg. Pair
126 - |yt,t,s,vt,vf|      Ys There
182 - |zd,r,v-,v0,v+|     "Z" reg. Decrement and branch
183 - |zi,r,v-,v0,v+|     "Z" reg. Increment and branch
184 - |zq,r|              "Z" reg. Query
185 - |zs,r,n|            "Z" reg. Set
```

Editorial Note: Object tapes for the 8080 and Z-80 versions of SAM 76 are available from Computer Mart of New Jersey, 501 Rt. 27, Iselin, NJ 08830. Telephone: (201) 283-0600. Tentative price will be $6.00. Computer Mart of New Jersey will also be publishing a book containing all available documentation on SAM 76. Available in early Spring, it will sell for $9.95. – *TRW*

An 8080 Disassembler and Support Software for SOL

by Don Van Dyke
1125 Lakehurst Rd.
Livermore, CA 94550

This is a guide for the use of the following disassembler. This disassembler performs most of the work of creating a file, but may require human help in the case of lookup tables.

Requirements

1. 4k of ram in the area from D000 to E000 (hex in the same area of ALS8 requires its scratchpad.
2. ALS8 assembler with TXT-2 addition. The user must therefore also have a VDM-1 or SOL video display.
3. Enough ram to contain the disassembler, (2k) the created file, the object code and the ALS8 assembler, though operation of this program does not require that ALS8 be in memory at the same time if the return point is modified.
4. SOLOS or CUTER. This program calls the standard input routine and inputs from the current pseudo port.

Instructions

1. Load the object code to be disassembled into the portion of memory where it normally resides.
2. Choose the correct version of the disassembler. Pick one that is located out of the way of the object code, and leave enough room for the ALS8 file. Remember that 1k of object will usually create about 11k of source. Load the program into memory.
3. Execute the program at the proper location, depending on the version selected from the tape. Instructions will be displayed on the screen.
4. After a few seconds the disassembly will be completed, at which time it will jump to E060 (the standard ALS8 return point), and will display the message "READY".
5. Create a new file with the command:

   FILE /name/ location

   and perform an FCHK on it. You should then be able to list, edit, or assemble the file at any location by using the proper ALS8 commands.

Helpful Hints

1. In some cases, it may be necessary for the user to decide where data tables are located in the object program. Methods for determining where these tables are located are provided in the program:
   a. The ASCII equivalent for each operation is provided in the comment field, if it is a letter, number, or punctuation symbol. This provides easy location of key words in tables, and the user should exercise caution when changing these areas.
   b. Whenever the file contains a line such as:

      0025 LC03BH DB 0CBH

      this means the op found at that location contained an unused 8080 instruction, and is therefore part of a table of some kind.
2. If the file contains any undefined symbol errors during assembly, this means a label is required in a table, where a two or three byte operation overlapped the op where the label was to be placed. This situation must be resolved by the user by the following method:
   a. Assemble the file, noting any undefined labels and write the labels down on paper.
   b. Use the TXT-2 video editor to add an 'EQU' pseudo statement immediately preceding the statement that required the label. An example is shown below.

      Suppose the disassembler creates a file as follows:

```
0001 L0104FH    LXI     H, L029FH
0002            MOV     A, M
0003            INX     H
....            ...     ...
....  (etc.)    ...     ...
0455            LXI     B, 04567H
....  (etc.)    ...     ...
```

      Then during assembly, this is generated:

```
017F 21 00 00    U 0455 L0104FH    LXI    H, L029FH
```

      (the 'U' denotes and undefined symbol)

      Where L029FH should be on the SECOND BYTE of the statement:

```
0455            LXI     B, 04567H
```

      The user should add the following line to the file:

```
>>> 0000 L029FH    EQU     $ + 1  <<<
    0455            LXI     B, 04567H
```

      Note that this will assemble into the same thing, but will allow the use of the label.
3. Notice that the line numbers generated are all 0000. This is not a problem since the RNUM function may be used to renumber the lines. Be sure to do this before using the line editor, but it is not required for the video editor.
4. To insure compatability with assemblers, this program generates a '6', instead of SP, PSW, or M. Also, three byte instructions contain the address in normal order, rather than the reverse INTOL standard.
5. Since PCHL instructions can be hazardous in the case of modifying H&L, this program generates a warning when it encounters a PCHL op. The following message will appear in the comment field:

   *******<<< WARNING >>>*******

Support Software

This is a guide for the use of the support software for the accompanying disassembler.

Requirements

1. ALS8 assembler with TXT-2 addition. The user must therefore also have a VDM-1 or SOL video display.

2. 12k of ram from D000 to FFFF (hex) (the standard ALS8 configuration).
3. SOLOS or CUTER. This program calls the standard input routine and inputs from the current pseudo port.

Debug

1. This program uses the current file pointers from ALS8 and will help to restore crashed files. Although it cannot work miracles, in most instances it will restore the file to a point where the video editor can finish the job if necessary.
2. It is recommended that DEBUG be loaded into memory and be made an ALS8 custom command – i.e.:
   CUSTE / DBUG /CB00 then when the file needs to be fixed, make it the current file using: FILE /name/ <cr>. Then type 'DBUG'. After a few moments, ALS8 should respond with an FCHK and a 'READY'.

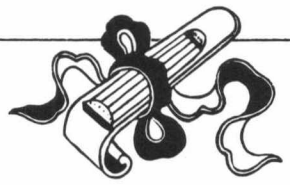
Adump

1. This program works in conjunction with SOLOS or CUTER and will display the contents of memory in its ASCII equivalent.
2. To use the program, load it into memory from the tape, then make it a custom command by typing:
   >CUST AD CB00
   The program will now operate like a standard dump by typing this command:
   >AD (beginning address) (end address)
   Note that only characters inside the range from 10 to 7E (hex) will appear on the screen, and anything outside that range shows up as a '.'.

Exam

1. This program allows the user to examine the contents of memory, and is very useful in locating tables and text in memory when used in conjunction with ADUMP.
2. To use this program, load it into memory and execute it at the beginning. What you will see is the first 1k of memory. To advance to the next 1k segment, press the up arrow (or ctrl-W), to back up to the previous 1k segment, press the down arrow (or ctrl-Z). To advance 8k at once, press tab (or ctrl-I). To exit to SOLOS or CUTER, press escape.

ALS8, TXT-2, VDM-1, CUTS, SOL, SOLOS and CUTER are copyright by Processor Technology Corporation.



```
0001 **********************************************************
0002 *                                                        *
0003 *    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  *
0004 *    % Written Sept 1977 by Don Van Dyke %               *
0005 *    % for quarter project in Math 4928 %                *
0006 *    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  *
0007 *                                                        *
0008 *  This is a disassembler. It disassembles object code *
0009 *  and creates an ALS8 (TM) compatable file, complete   *
0010 *  with labels. The file may by reassembled at any      *
0011 *  location by using the ALS8 assembler.                *
0012 **********************************************************
0013 *
0014 *
0015 *
0016 * INPUT ROUTINES TO PROCESS ADDRESSES BEFORE DISASSEMBLY
0017 * *
0018 *
0000 0019          LXI   6+0D700H-1  SET STACK
0003 21 00 D6      LXI   H,0CD00H    SETUP FOR MESSAGES
0006 22 3A 05      SHLD  INSLOC  INSTRUCTION LOCATION
0009 21 34 05      LXI   H,BUFF  BUFFER LOCATION
000C 22 32 05      SHLD  BUFLOC  SAVE IT
000F AF            XRA   A       INITIALIZE SCREEN
0010 D3 FE         OUT   0FEH    FOR VDM
0012 D3 CB         OUT   0CBH    FOR SOL
0014 CD 8A 00      CALL  BLANK   CLEAR THE SCREEN
0017 01 40 00      LXI   B,40H   OFFSET FOR EACH INSTRUCTION
001A 11 4A 05      LXI   D,INST  FIRST MESSAGE
001D 2A 3A 05      LHLD  INSLOC  GET THE POSITION
0020 CD B2 00      CALL  MESS    DISPLAY FIRST MESSAGE
0023 2A 3A 05      LHLD  INSLOC  GET IT AGAIN
0026 09            DAD   B       OFFSET FOR SECOND MESSAGE
0027 09            DAD   B       MORE OFFSET
0028 22 3A 05      SHLD  INSLOC  SAVE NEW LINE POSITION
002B 13            INX   D       SETUP FOR NEXT MESSAGE
002C 7C      MSD   MOV   A,H     GET HIGH ADDRESS
002D FE CF         CPI   0CFH    READY FOR FIRST PASS ?
002F CA BC 00      JZ    FSTPSS  YES
0032 CD B2 00      CALL  MESS    NO, DISPLAY ANOTHER MESSAGE
0035 13            INX   D       NEXT MESSAGE
0036 2A 3A 05      LHLD  INSLOC  GET OLD POSITION
0039 09            DAD   B       ADD THE OFFSET
003A 22 3A 05      SHLD  INSLOC  SAVE IT
003D CD 97 00      CALL  INPT    GET THE CHAR
0040 FE 0D         CPI   0DH     END OF LINE ?
0042 C2 3D 00      JNZ   CIN     NO, GET ANOTHER ONE
0045 36 20         MVI   M,' '   YES, DON'T DISPLAY THE <CR>
0047 CD 54 00      CALL  PROCESS TURN IT INTO NUMBERS
004A 2A 3A 05      LHLD  INSLOC  GET OLD POSITION
004D 09            DAD   B       ADD OFFSET
004E 22 3A 05      SHLD  INSLOC  SAVE IT
0051 C3 2C 00      JMP   MSD     DO IT AGAIN
0054
0054 * ROUTINE TO CHANGE FOUR DIGIT ASCII TO BINARY
0054 *
0054 PROCESS PUSH  D       SAVE THE LITTLE MONSTERS
0055 C5            PUSH  B       AND SOME MORE
0056 2A 32 05      LHLD  BUFLOC  GET POSITION IN BUFFER
0059 EB            XCHG
005A 2A 3A 05      LHLD  INSLOC  GET LOCATION OF INPUT
005D 23            INX   H       PAST FIRST
005E 23            INX   H       TO SECOND
005F CD 73 00      CALL  ASC     ASCIFY THEM
0062 12            STAX  D       SAVE THE CONVERTED ONES
0063 13            INX   D       POINT TO NEXT
0064 2B            DCX   H       BACK UP
0065 2B            DCX   H
0066 2B            DCX   H
0067 CD 73 00      CALL  ASC     ASCIFY THEM
006A 12            STAX  D       SAVE THEM
006B 13            INX   D       NEXT POSITION
006C EB            XCHG
006D 22 32 05      SHLD  BUFLOC  SAVE NEW BUFFER LOCATION
0070 C1            POP   B       RESTORE THE OLD VALUES
0071 D1            POP   D
0072 C9            RET
0073
0073 *
0073 CD 81 00      CALL  AGET    STRIP ASCII BIAS
0076 07            RLC   SHIFT
0077 07            RLC   IT
0078 07            RLC   OVER
0079 07            RLC   !
007A 47            MOV   B,A     SAVE THE FIRST HALF
```

```
007B 23          0085         INX    H       GET THE OTHER HALF
007C CD 81 00    0086         CALL   AGET    STRIP OFF BIAS
007F 80          0087         ADD    B       ADD IT TO THE OTHER HALF
0080 C9          0088         RET
0081 7E          0089 AGET    MOV    A,M     GET THE CHARACTER
0082 D6 30       0090         SUI    30H     REMOVE BIAS FOR NUMBER
0084 FE 0A       0091         CPI    0AH     STILL A NUMBER ?
0086 D8          0092         RC     .       YES
0087 D6 07       0093         SUI    7       NO, TAKE 7 MORE
0089 C9          0094         RET
008A             0095 *
008A             0096 * ROUTINE TO CLEAR THE SCREEN
008A             0097 *
008A 21 00 CC    0098 BLANK   LXI    H,0CC00H  BEGINNING OF SCREEN
008D 36 20       0099 BLK     MVI    M,' '   PUT A BLANK
008F 23          0100         INX    H       NEXT
0090 7C          0101         MOV    A,H     GET HIGH ADDRESS
0091 FE D0       0102         CPI    0D0H    DONE ?
0093 C2 8D 00    0103         JNZ    BLK     NO
0096 C9          0104         RET    YES
0097             0105 *
0097             0106 * THIS IS THE INPUT ROUTINE, COMPLETE WITH CURSOR
0097             0107 * RECEIVING INPUT FROM THE CURRENT SOLOS/CUTER
0097             0108 * PSEUDO INPUT PORT.
0097             0109 *
0097 36 A0       0110 INPT    MVI    M,0A0H  PLACE THE CURSOR
0099 CD 1F C0    0111 IN      CALL   0C01FH
009C CA 99 00    0112         JZ     IN      GET THE CHARACTER
009F FE 0D       0113         CPI    0DH     <CR> ?
00A1 C8          0114         RZ     YES
00A2 FE 7F       0115         CPI    7FH     DELETE ?
00A4 C2 AD 00    0116         JNZ    ARD     NO
00A7 36 20       0117         MVI    M,' '   YES, PUT A BLANK
00A9 2B          0118         DCX    H       BACK UP
00AA C3 AF 00    0119         JMP    CURS
00AD 77          0120 ARD     MOV    M,A     DISP THE CHAR
00AE 23          0121         INX    H       NEXT
00AF 36 A0       0122 CURS    MVI    M,0A0H
00B1 C9          0123         RET
00B2             0124 *
00B2             0125 * THIS ROUTINE DISPLAYS A MESSAGE AT HL
00B2             0126 * THE MESSAGE IS POINTED TO BY DE
00B2             0127 *
00B2 1A          0128 MESS    LDAX   D       GET THE CHAR
00B3 FE 00       0129         CPI    0       END OF MESSAGE ?
00B5 C8          0130         RZ     YES
00B6 77          0131         MOV    M,A     NO, DISPLAY IT
00B7 13          0132         INX    D       NEXT
00B8 23          0133         INX    H       NEXT DISPLAY POSITION
00B9 C3 B2 00    0134         JMP    MESS    AGAIN
00BC             0135 MOV     EQU    40H
00BC             0136 *
00BC             0137 *     <<<<    FIRST PASS    >>>>
00BC             0138 *
00BC             0139 * THIS PASS FINDS ALL JUMPS AND CALLS AND PLACES
00BC             0140 * THE ADRESSES IN THE ALS8 SYMBOL TABLE AREA
00BC             0141 *
00BC 21 00 D7    0142 FSTPSS  LXI    H,0D700H
00BF 22 3C 05    0143         SHLD   STABLOC  SAVE POSITION
00C2 01 00 D7    0144         LXI    B,0D700H  START OF SYMBOL TABLE
00C5 2A 34 05    0145         LHLD   BGNLOC  BEGINNING OF CODE
00C8 EB          0146         XCHG   GET IT IN DE
00C9 2A 36 05    0147         LHLD   END     END OF CODE
00CC CD 45 01    0148 AGNI    CALL   ADCMP
00CF CA 4E 01    0149         JZ     DOONE
00D2 DA 4E 01    0150         JC     DOONE   END OF FIRST PASS ?
00D5 1A          0151         LDAX   D       NO
00D6 FE D3       0152         CPI    0D3H    OUT ?
00D8 CA 29 01    0153         JZ     BT2
00DB FE DB       0154         CPI    0DBH    IN ?
00DD CA 29 01    0155         JZ     BT2
00E0 FE EB       0156         CPI    0EBH    XCHG ?
00E2 CA 25 01    0157         JZ     INHH
00E5 FE E3       0158         CPI    0E3H    XTHL ?
00E7 CA 25 01    0159         JZ     INHH
00EA FE 2A       0160         CPI    2AH     LHLD ?
00EC CA 2E 01    0161         JZ     FIND
00EF FE 22       0162         CPI    22H     SHLD ?
00F1 CA 2E 01    0163         JZ     FIND
00F4 FE 3A       0164         CPI    3AH     LDA ?
00F6 CA 2E 01    0165         JZ     FIND
00F9 FE 32       0166         CPI    32H     STA ?
00FB CA 2E 01    0167         JZ     FIND
00FE E6 CF       0168         ANI    0CFH    STRIP SOME OFF
0100 FE CD       0169         CPI    0CDH    CALL ?
0102 CA 2E 01    0170         JZ     FIND
0105 FE 01       0171         CPI    1       LXI ?
0107 CA 2E 01    0172         JZ     FIND
010A E6 C7       0173         ANI    0C7H    AND SOME MORE
010C FE C2       0174         CPI    0C2H    JUMP (cond) ?
010E CA 2E 01    0175         JZ     FIND
0111 FE C4       0176         CPI    0C4H    CALL (cond) ?
0113 CA 2E 01    0177         JZ     FIND
0116 FE C3       0178         CPI    0C3H    JUMP ?
0118 CA 2E 01    0179         JZ     FIND
011B FE 06       0180         CPI    6       DONE WITH 3 BYTE OP'S, NOW DO 2 BYTE
011D CA 29 01    0181         JZ     BT2
0120 FE C6       0182         CPI    0C6H    ACC IMMEDIATES ?
0122 CA 29 01    0183         JZ     BT2
0125 13          0184 INHH    INX    D       IF IT COMES HERE, OP IS 1 BYTE
0126 C3 CC 00    0185         JMP    AGNI    GET NEXT OP
0129             0186 *
0129 13          0187 BT2     INX    D       OP WAS 2 BYTES
012A 13          0188         INX    D
012B C3 CC 00    0189         JMP    AGNI    INCREMENT OP COUNT
012E             0190 *
012E             0191 * THIS ROUTINE PLACES LABEL ADDRESS IN ALS8
012E             0192 * SYMBOL TABLE AREA.
012E             0193 *
012E 13          0194 FIND    INX    D       POINT TO NEXT CHAR
012F 1A          0195         LDAX   D       GET IT
0130 02          0196         STAX   B       STORE IT IN TABLE
0131 13          0197         INX    D       INCREMENT FOR SECOND BYTE
0132 03          0198         INX    B
0133 1A          0199         LDAX   D       GET SECOND BYTE
0134 02          0200         STAX   B       STORE IT
0135 13          0201         INX    D       SETUP FOR NEXT ADDRESS
0136 03          0202         INX    B
0137 2A 3C 05    0203         LHLD   STABLOC  GET THE CURRENT TABLE POSITION
013A 23          0204         INX    H       SET IT UP FOR NEXT ADDRESS
013B 23          0205         INX    H
013C 22 3C 05    0206         SHLD   STABLOC  PUT IT BACK
013F 2A 36 05    0207         LHLD   END
0142 C3 CC 00    0208         JMP    AGNI    END OF OBJECT ?
0145             0209 *
0145             0210 * ROUTINE TO COMPARE 2 ADDRESSES
0145             0211 *
0145 7D          0212 ADCMP   MOV    A,L     GET LOW ORDER ADDRESS
0146 BB          0213         CMP    E       EQUAL ?
0147 CA 4B 01    0214         JZ     FUR2    YES, CHECK SOME MORE
014A D0          0215         RNC    .       <>
014B 7C          0216 FUR2    MOV    A,H     GET HIGH
014C BA          0217         CMP    D       SET FLAGS
014D C9          0218         RET
014E             0219 *
014E             0220 *     <<<<    SECOND PASS    >>>>
014E             0221 *
014E             0222 * THIS PASS DISASSEMBLES INTO THE ACTUAL MNEMONICS,
014E             0223 * PLACING LINE NUMBERS AND LABELS.
014E             0224 *
014E 2A 38 05    0225 DOONE   LHLD   PLO     GET STARTING POSITION
0151 22 47 05    0226         SHLD   PLOC    SAVE IT
0154 2A 34 05    0227 START   LHLD   BGNLOC  GET STARTING ADDRESS
0157 22 40 05    0228         SHLD   POSLOC  SAVE THIS, TOO
015A 2A 40 05    0229 NEWOP   LHLD   POSLOC  GET POSITION
015D EB          0230         XCHG
015E 2A 36 05    0231         LHLD   END     GET END ADDRESS
0161 CD 45 01    0232         CALL   ADCMP   DONE YET ?
0164 CA 6A 01    0233         JZ     THEND   YES
0167 D2 75 01    0234         JNC    ANMM    NO
016A 2A 47 05    0235 THEND   LHLD   PLOC
016D 36 01       0236         MVI    M,1     PLACE EOF MARKER
016F 23          0237         INX    H
0170 36 20       0238         MVI    M,' '   AND A BLANK
0172 C3 E0 04    0239         JMP    FILES   NOW GO SET LINE LENGTHS
0175 2A 40 05    0240 ANMM    LHLD   POSLOC
0178 7E          0241         MOV    A,M     GET DATA (OP)
0179 32 44 05    0242         STA    OPLOC   STORE IT
017C C3 9D 01    0243         JMP    LOKUP   LO(O)K UP BASIC MNEMONIC
017F 1A          0244 NEXBT   LDAX   D       GET THE DATA
0180 FE 08       0245         CPI    8       THESE ARE DIVIDERS FOR THE MNEMONIC TABLE
0182 C8          0246         RZ
0183 FE 10       0247         CPI    10H
0185 C8          0248         RZ
0186 FE 18       0249         CPI    18H
0188 C8          0250         RZ
0189 FE 20       0251         CPI    20H
018B C8          0252         RZ
018C FE 28       0253         CPI    28H
018E C8          0254         RZ
```

```
018F FE 00          0255          CPI   0       THERE YET?
0191 13             0256          INX   D
0192 1A             0257          LDAX  D
0193 C2 7F 01       0258          JNZ   NEXBT   GET THE NEXT ONE
0196 C9             0259          RET
0197 7D             0260 ADC22    MOV   A,L
0198 BB             0261          CMP   E
0199 C0             0262          RNZ
019A 7C             0263          MOV   A,H
019B BA             0264          CMP   D
019C C9             0265          RET
019D                0266 *
019D                0267 * ROUTINE TO LOOK UP THE MNEMONIC
019D                0268 *
019D 2A 40 05       0269 LOKUP    LHLD  POSLOC  GET THE POSITION
01A0 AF             0270          XRA   A
01A1 32 31 05       0271          STA   FLAG    FLAG <=0
01A4 01 00 D7       0272          LXI   B,0D700H  START OF SYMBOL TABLE
01A7 E5             0273 LABL     PUSH  H
01A8 2A 3C 05       0274          LHLD  STABLOC  GET LOCATION IN SYMBOL TABLE
01AB 50             0275          MOV   D,B     PUT IT IN DE
01AC 59             0276          MOV   E,C
01AD CD 97 01       0277          CALL  ADC22   IS THE ADDRESS IN THE TABLE
01B0 E1             0278          POP   H       NO, TRY THE NEXT ADDRESS
01B1 CA C8 01       0279          JZ    NXT     YES, NO LABEL NEEDED
01B4 0A             0280          LDAX  B       GET THE ADDR. IN THE TABLE
01B5 5F             0281          MOV   E,A     PUT IT IN DE
01B6 03             0282          INX   B
01B7 0A             0283          LDAX  B
01B8 57             0284          MOV   D,A
01B9 CD 97 01       0285          CALL  ADC22   ADRESSES EQUAL ?
01BC CA C3 01       0286          JZ    MATCH   ADDRESS IS IN THE TABLE
01BF 03             0287          INX   B       NO, TRY AGAIN
01C0 C3 A7 01       0288          JMP   LABL
01C3 3E 01          0289 MATCH    MVI   A,1     SET FLAG INDICATING LABEL IS NEEDED
01C5 32 31 05       0290          STA   FLAG
01C8 11 24 05       0291 NXT      LXI   D,TAB3H  NOW LOOK FOR ILLEGAL OP
01CB 3A 44 05       0292          LDA   OPLOC   GET THE OP
01CE 47             0293          MOV   B,A     SAVE A COPY
01CF 1A             0294 LKJ      LDAX  D       GET THE BYTE
01D0 13             0295          INX   D
01D1 FE 00          0296          CPI   0       END OF TABLE ?
01D3 CA EE 01       0297          JZ    ENOD    YES, NO ILLEGAL OP
01D6 B8             0298          CMP   B       OP = OP ?
01D7 C2 CF 01       0299          JNZ   LKJ     TRY AGAIN
01DA CD 70 04       0300          CALL  LINE    ILLEGAL OP
01DD 36 44          0301          MVI   M,'D'   PUT A DB PSEUDO OP IN THE FILE
01DF 23             0302          INX   H
01E0 36 42          0303          MVI   M,'B'
01E2 23             0304          INX   H
01E3 3E 01          0305          MVI   A,1     TELL THE PROGRAM OP IS 1 BYTE
01E5 32 49 05       0306          STA   CLOCLOC
01E8 32 3F 05       0307          STA   LOCLOC
01EB C3 B7 03       0308          JMP   SPACE   FINISH THE LINE
01EE 3A 44 05       0309 ENOD     LDA   OPLOC   GET THE OP
01F1 FE 01          0310          CPI   1       =1 ?
01F3 C2 FD 01       0311          JNZ   LKOO    NO
01F6 47             0312          MOV   B,A     SAVE A COPY
01F7 11 C3 06       0313          LXI   D,OPTB  START OF OPTABLE
01FA C3 0E 02       0314          JMP   RPOFF
01FD 47             0315 LKOO     MOV   B,A     SAVE A COPY
01FE 11 E9 05       0316          LXI   D,OPTAB  STARTING ADDRESS OF TABLE
0201 CD 7F 01       0317 LOOK1    CALL  NEXBT   GET THE DATA
0204 B8             0318          CMP   B       IS IT THE ONE?
0205 CA DA 02       0319          JZ    ADDIT   ADD IT TO THE FILE
0208 FE 08          0320          CPI   08H     END OF SECTION?
020A C2 01 02       0321          JNZ   LOOK1   NO
020D 13             0322          INX   D
020E 78             0323 RPOFF    MOV   A,B     GET IT BACK
020F E6 CF          0324          ANI   0CFH    CHECK FOR RP's
0211 47             0325          MOV   B,A     SAVE IT AGAIN
0212 CD 7F 01       0326 LOOK2    CALL  NEXBT   GET THE NEXT BYTE
0215 B8             0327          CMP   B
0216 CA A0 02       0328          JZ    RPS     REG PAIR
0219 FE 10          0329          CPI   10H
021B C2 12 02       0330          JNZ   LOOK2
021E 13             0331          INX   D
021F 78             0332          MOV   A,B
0220 E6 C7          0333          ANI   0C7H    DEST's
0222 47             0334          MOV   B,A
0223 CD 7F 01       0335 LOOK3    CALL  NEXBT
0226 B8             0336          CMP   B
0227 CA 86 02       0337          JZ    DEST    DESTINATION
022A FE 18          0338          CPI   18H
022C C2 23 02       0339          JNZ   LOOK3
022F 13             0340          INX   D
0230 3A 44 05       0341          LDA   OPLOC   GET THE COPY
0233 E6 F8          0342          ANI   0F8H    SOURCES
0235 47             0343          MOV   B,A
0236 CD 7F 01       0344 LOOK4    CALL  NEXBT
0239 B8             0345          CMP   B
023A CA 6F 02       0346          JZ    SRCS
023D FE 20          0347          CPI   20H
023F C2 36 02       0348          JNZ   LOOK4
0242 78             0349          MOV   A,B
0243 E6 C0          0350          ANI   0C0H
0245 FE 40          0351          CPI   40H
0247 C2 60 E0       0352          JNZ   ERROR   OP NOT IN TABLE
024A EB             0353 MOVE     XCHG
024B 22 45 05       0354          SHLD  MNLOC MNEMONIC LOCATION
024E 3E 05          0355          MVI   A,5     DEST, SOURCE
0250 32 43 05       0356          STA   TYPE
0253 3A 44 05       0357          LDA   OPLOC   GET IT!
0256 E6 07          0358          ANI   7       SOURCE
0258 CD 62 04       0359          CALL  ASCIT
025B 32 42 05       0360          STA   SRCLOC  STORE ASCII FOR SOURCE
025E 3A 44 05       0361          LDA   OPLOC
0261 E6 38          0362          ANI   38H
0263 1F             0363          RAR
0264 1F             0364          RAR
0265 1F             0365          RAR
0266 CD 62 04       0366          CALL  ASCIT
0269 32 3E 05       0367          STA   DSTLOC  STORE DEST
026C C3 E3 02       0368          JMP   PUT     PUT IT IN THE FILE
026F EB             0369 SRCS     XCHG  SOURCES
0270 22 45 05       0370          SHLD  MNLOC
0273 3E 03          0371          MVI   A,3     SOURCE ONLY
0275 32 43 05       0372          STA   TYPE
0278 3A 44 05       0373          LDA   OPLOC
027B E6 07          0374          ANI   7
027D CD 62 04       0375          CALL  ASCIT
0280 32 42 05       0376          STA   SRCLOC
0283 C3 E3 02       0377          JMP   PUT
0286 EB             0378 DEST     XCHG  DESTINATIONS ONLY
0287 22 45 05       0379          SHLD  MNLOC
028A 3E 02          0380          MVI   A,2
028C 32 43 05       0381          STA   TYPE
028F 3A 44 05       0382          LDA   OPLOC
0292 E6 38          0383          ANI   38H
0294 1F             0384          RAR
0295 1F             0385          RAR
0296 1F             0386          RAR
0297 CD 62 04       0387          CALL  ASCIT
029A 32 3E 05       0388          STA   DSTLOC
029D C3 E3 02       0389          JMP   PUT
02A0 EB             0390 RPS      XCHG  REGISTER PAIRS
02A1 22 45 05       0391          SHLD  MNLOC
02A4 3E 04          0392          MVI   A,4
02A6 32 43 05       0393          STA   TYPE
02A9 3A 44 05       0394          LDA   OPLOC
02AC E6 30          0395          ANI   30H
02AE 1F             0396          RAR
02AF 1F             0397          RAR
02B0 1F             0398          RAR
02B1 1F             0399          RAR
02B2 FE 00          0400          CPI   0
02B4 C2 BC 02       0401          JNZ   LP1
02B7 06 42          0402          MVI   B,42H  B
02B9 C3 E3 02       0403          JMP   ARND
02BC FE 01          0404 LP1      CPI   1
02BE C2 C6 02       0405          JNZ   LP2
02C1 06 44          0406          MVI   B,44H  D
02C3 C3 E3 02       0407          JMP   ARND
02C6 FE 02          0408 LP2      CPI   2
02C8 C2 D0 02       0409          JNZ   LP3
02CB 06 48          0410          MVI   B,48H  H
02CD C3 E3 02       0411          JMP   ARND
02D0 FE 03          0412 LP3      CPI   3
02D2 C2 60 E0       0413          JNZ   ERROR
02D5 06 36          0414          MVI   B,'6'  SP,PSW
02D7 C3 E3 02       0415          JMP   PUT
02DA 3E 01          0416 ADDIT    MVI   A,1     SIMPLE
02DC 32 43 05       0417          STA   TYPE
02DF EB             0418          XCHG
02E0 22 45 05       0419          SHLD  MNLOC  SAVE LOCATION OF MNEMONIC
02E3 CD 70 04       0420 PUT      CALL  LINE    ADD THE LINE NUMBERS
02E6                0421 ARND     EQU   PUT
02E6 2A 45 05       0422          LHLD  MNLOC   GET THE LOC
```

```
02E9 23          0423         INX    H
02EA 7E          0424         MOV    A,M    NUMBER OF BYTES
02EB 32 3F 05    0425         STA    LOCLOC  SAVE IT
02EE 23          0426         INX    H
02EF EB          0427         XCHG
02F0 2A 47 05    0428         LHLD   PLOC   GET POSITION IN FILE
02F3 3A 31 05    0429         LDA    FLAG   GET THE FLAG
02F6 FE 00       0430         CPI    0      NEED LABEL ?
02F8 CA 04 03    0431         JZ     NCHR   NO
02FB C5          0432         PUSH   B      YES
02FC D5          0433         PUSH   D
02FD 2B          0434         DCX    H
02FE CD 89 04    0435         CALL   LABLER  PUT ON THE LABEL
0301 23          0436         INX    H
0302 D1          0437         POP    D
0303 C1          0438         POP    B
0304 1A          0439 NCHR    LDAX   D
0305 FE 00       0440         CPI    0
0307 CA 10 03    0441         JZ     ENDOP  END OF OP
030A 77          0442         MOV    M,A
030B 13          0443         INX    D
030C 23          0444         INX    H
030D C3 04 03    0445         JMP    NCHR
0310 3A 43 05    0446 ENDOP   LDA    TYPE
0313             0447 * 1 = SIMPLE
0313             0448 * 2 = DEST ONLY
0313             0449 * 3 = SOURCE ONLY
0313             0450 * 4 = REG PAIR
0313             0451 * 5 = DEST,SOURCE
0313 3D          0452         DCR    A
0314 CA 9A 03    0453         JZ     CR     SIMPLE
0317 3D          0454         DCR    A
0318 CA 42 03    0455         JZ     DO     DEST ONLY
031B 3D          0456         DCR    A
031C CA 92 03    0457         JZ     SO     SOURCE ONLY
031F 3D          0458         DCR    A
0320 CA 3A 03    0459         JZ     RP     REG PAIR
0323 3D          0460         DCR    A
0324 C2 60 E0    0461         JNZ    ERROR  NOTHING ?
0327 36 20       0462         MVI    M,' '
0329 23          0463         INX    H
032A 3A 3E 05    0464         LDA    DSTLOC  GET THE DESTINATION
032D 77          0465         MOV    M,A
032E 23          0466         INX    H
032F 36 2C       0467         MVI    M,','
0331 23          0468         INX    H
0332 3A 42 05    0469         LDA    SRCLOC  GET THE SOURCE
0335 77          0470         MOV    M,A
0336 23          0471         INX    H
0337 C3 9A 03    0472         JMP    CR
033A 36 20       0473 RP      MVI    M,' '  REG PAIR
033C 23          0474         INX    H
033D 70          0475         MOV    M,B
033E 23          0476         INX    H
033F C3 9A 03    0477         JMP    CR
0342 3A 44 05    0478 DO      LDA    OPLOC  DESTINATION ONLY
0345 E6 C7       0479         ANI    0C7H
0347 FE C2       0480         CPI    0C2H
0349 CA 61 03    0481         JZ     DOO
034C FE C4       0482         CPI    0C4H
034E CA 61 03    0483         JZ     DOO
0351 FE C0       0484         CPI    0C0H
0353 CA 61 03    0485         JZ     DOO
0356 36 20       0486         MVI    M,' '
0358 23          0487         INX    H
0359 3A 3E 05    0488         LDA    DSTLOC
035C 77          0489         MOV    M,A
035D 23          0490         INX    H
035E C3 9A 03    0491         JMP    CR
0361 11 80 03    0492 DOO     LXI    D,TAB2-2  GET THE CONDITION
0364 3A 44 05    0493         LDA    OPLOC
0367 E6 38       0494         ANI    38H
0369 1F          0495         RAR
036A 1F          0496         RAR
036B 1F          0497         RAR
036C 3C          0498         INR    A
036D 3D          0499 LPDCR   DCR    A      LOOK IT UP IN THE TABLE
036E 13          0500         INX    D
036F 13          0501         INX    D
0370 C2 6D 03    0502         JNZ    LPDCR
0373 1A          0503         LDAX   D
0374 77          0504         MOV    M,A
0375 23          0505         INX    H
0376 13          0506         INX    D
0377 1A          0507         LDAX   D
0378 FE 00       0508         CPI    0      END OF TABLE ?
037A CA 9A 03    0509         JZ     CR     FINISH LINE
037D 77          0510         MOV    M,A    PLACE IN FILE
037E 23          0511         INX    H
037F C3 9A 03    0512         JMP    CR
0382             0513 *
0382             0514 * TABLE OF CONDITIONS FOR CALL,RET,JMP
0382             0515 *
0382 4E 5A       0516 TAB2    ASC    'NZ'
0384 5A          0517         ASC    'Z'
0385 00          0518         DB     0
0386 4E 43       0519         ASC    'NC'
0388 43          0520         ASC    'C'
0389 00          0521         DB     0
038A 50 4F       0522         ASC    'PO'
038C 50 45       0523         ASC    'PE'
038E 50          0524         ASC    'P'
038F 00          0525         DB     0
0390 4D          0526         ASC    'M'
0391 00          0527         DB     0
0392 36 20       0528 SO      MVI    M,' '  SOURCE ONLY
0394 23          0529         INX    H
0395 3A 42 05    0530         LDA    SRCLOC
0398 77          0531         MOV    M,A
0399 23          0532         INX    H
039A             0533 *
039A             0534 * THIS ROUTINE PLACES INFORMATION AFTER THE MNEMONIC
039A             0535 *
039A             0536 CR      EQU    $
039A 3A 3F 05    0537         LDA    LOCLOC  GET # OF BYTES
039D 3D          0538         DCR    A
039E CA EC 03    0539         JZ     LPLPLP  1 BYTE
03A1 32 49 05    0540         STA    CLOCLOC  PUT IT BACK (-1)
03A4 36 2C       0541         MVI    M,','
03A6 3A 44 05    0542         LDA    OPLOC
03A9 E6 CF       0543         ANI    0CFH
03AB FE 01       0544         CPI    1
03AD CA B9 03    0545         JZ     COMMA  NEED A COMMA
03B0 E6 C7       0546         ANI    0C7H
03B2 FE 06       0547         CPI    6
03B4 CA B9 03    0548         JZ     COMMA
03B7 36 20       0549 SPACE   MVI    M,' '  NEED A SPACE
03B9 23          0550 COMMA   INX    H
03BA 36 30       0551         MVI    M,'0'  0 FOR SECOND AND/OR THIRD BYTE
03BC 23          0552         INX    H
03BD E5          0553         PUSH   H
03BE 2A 40 05    0554         LHLD   POSLOC  GET THE OBJECT POSITION
03C1 3A 3F 05    0555         LDA    LOCLOC  # OF BYTES
03C4 3D          0556         DCR    A
03C5 CA C9 03    0557         JZ     BYINX  TWO BYTES
03C8 23          0558         INX    H
03C9 7E          0559 BYINX   MOV    A,M    PUT ON THE DATA
03CA CD 43 04    0560         CALL   ASSGET  GET THE ASCII
03CD 50          0561         MOV    D,B    PUT IT IN DE
03CE 59          0562         MOV    E,C
03CF 23          0563         INX    H
03D0 7E          0564         MOV    A,M    NEXT CHARACTER
03D1 CD 43 04    0565         CALL   ASSGET  GET THE ASCII
03D4 E1          0566         POP    H
03D5 3A 49 05    0567         LDA    CLOCLOC
03D8 3D          0568         DCR    A
03D9 CA E5 03    0569         JZ     BEYOND  DONE ?
03DC D5          0570         PUSH   D
03DD CD A7 04    0571         CALL   CHEKR  IS ADDRESS INSIDE PROGRAM ?
03E0 D1          0572         POP    D
03E1 70          0573         MOV    M,B
03E2 23          0574         INX    H
03E3 71          0575         MOV    M,C
03E4 23          0576         INX    H
03E5 72          0577 BEYOND  MOV    M,D    PUT IT IN THE FILE
03E6 23          0578         INX    H
03E7 73          0579         MOV    M,E
03E8 23          0580         INX    H
03E9 36 48       0581         MVI    M,'H'  H FOR HEX
03EB 23          0582         INX    H
03EC 36 20       0583 LPLPLP  MVI    M,' '  SPACE OUT TO COMMENT FIELD
03EE 23          0584         INX    H
03EF 36 20       0585         MVI    M,' '  ANOTHER BLANK
03F1 23          0586         INX    H
03F2 EB          0587         XCHG
03F3 2A 40 05    0588         LHLD   POSLOC  WHERE ARE WE ?
03F6 EB          0589         XCHG
03F7 3A 3F 05    0590         LDA    LOCLOC  GET # OF BYTES
03FA 47          0591         MOV    B,A
```

```
03FB 3D          0592          DCR   A
03FC C2 16 04    0593          JNZ   LP5H    ; 1 BYTE OP
03FF 3A 43 05    0594          LDA   TYPE
0402 3D          0595          DCR   A
0403 C2 16 04    0596          JNZ   LP5H    PUT IN A "." TO MAKE ALS8 HAPPY
0406 2B          0597          DCX   H
0407 36 2E       0598          MVI   M,'.'
0409 23          0599          INX   H
040A 36 20       0600          MVI   M,' '
040C 23          0601          INX   H
040D 36 20       0602          MVI   M,' '
040F 23          0603          INX   H
0410 1A          0604          LDAX  D
0411 FE E9       0605          CPI   0E9H    EGAD!!!!------ A PICKLE !!!!!
0413 CA 25 04    0606          JZ    PHL1
0416 1A          0607 LP5H     LDAX  D
0417 FE 7F       0608          CPI   7FH     DON'T DISPLAY ANYTHING LARGER THAN 7FH
0419 D2 2D 04    0609          JNC   NEXTS
041C FE 20       0610          CPI   20H     NO CONTROL CHARACTERS
041E DA 2D 04    0611          JC    NEXTS
0421 77          0612          MOV   M,A     PLACE THE ASCII
0422 C3 2F 04    0613          JMP   BHU
0425 D5          0614 PHL1     PUSH  D       DISPLAY PICKLE WARNING
0426 11 CD 05    0615          LXI   D,PCHL  BEGINNING OF MESSAGE
0429 CD B2 00    0616          CALL  MESS    DISPLAY IT
042C D1          0617          POP   D
042D 36 20       0618 NEXTS    MVI   M,' '
042F 23          0619 BHU      INX   H
0430 13          0620          INX   D
0431 05          0621          DCR   B
0432 C2 16 04    0622          JNZ   LP5H    DONE ?
0435 36 0D       0623          MVI   M,0DH   YES, PUT IN <CR>
0437 23          0624          INX   H
0438 22 47 05    0625          SHLD  PLOC
043B EB          0626          XCHG
043C 22 40 05    0627          SHLD  POSLOC  REPLACE POINTERS
043F EB          0628          XCHG
0440 C3 5A 01    0629          JMP   NEWOP   GET THE NEXT OP
0443             0630 *
0443             0631 * ROUTINE TO CONVERT A TO 2 BYTE DISPLAY NUMBER
0443             0632 *
0443 47          0633 ASSGET MOV   B,A
0444 E6 0F       0634          ANI   0FH
0446 C6 30       0635          ADI   30H     ADD ASCII BIAS
0448 FE 3A       0636          CPI   3AH
044A DA 4F 04    0637          JC    LPPL
044D C6 07       0638          ADI   7       NEED 7 MORE
044F 4F          0639 LPPL     MOV   C,A     NOW DO SECOND HALF
0450 78          0640          MOV   A,B
0451 E6 F0       0641          ANI   0F0H
0453 1F          0642          RAR           PUSH IT OVER
0454 1F          0643          RAR
0455 1F          0644          RAR
0456 1F          0645          RAR
0457 C6 30       0646          ADI   30H     ADD ASCII BIAS
0459 FE 3A       0647          CPI   3AH
045B DA 60 04    0648          JC    PLLP
045E C6 07       0649          ADI   7       AND 7 MORE
0460 47          0650 PLLP     MOV   B,A
0461 C9          0651          RET
0462 3C          0652 ASCIT    INR   A       GET THE REGISTER
0463 11 1C 05    0653          LXI   D,TAB
0466 3D          0654 LP       DCR   A
0467 CA 6E 04    0655          JZ    LOOK
046A 13          0656          INX   D
046B C3 66 04    0657          JMP   LP
046E 1A          0658 LOOK     LDAX  D
046F C9          0659          RET
0470             0660 *
0470             0661 * ROUTINE TO PLACE LINE NUMBERS IN THE FILE
0470             0662 *
0470 2A 47 05    0663 LINE     LHLD  PLOC    PLACE IN FILE
0473 36 20       0664          MVI   M,' '   A BLANK
0475 23          0665          INX   H
0476 3E 04       0666          MVI   A,4     4 OF THEM
0478 36 30       0667 LPP      MVI   M,'0'
047A 3D          0668          DCR   A
047B 23          0669          INX   H
047C C2 78 04    0670          JNZ   LPP     4 ZEROES
047F 36 20       0671          MVI   M,' '
0481 23          0672          INX   H
0482 36 20       0673          MVI   M,' '
0484 23          0674          INX   H       TWO SPACES
0485 22 47 05    0675          SHLD  PLOC
0488 C9          0676          RET
0489             0677 *
0489             0678 * ROUTINE TO PLACE LABELS IN FILE
0489             0679 *
0489 36 4C       0680 LABLER MVI   M,'L'  PUT IN THE L
048B 23          0681          INX   H
048C EB          0682          XCHG
048D 2A 40 05    0683          LHLD  POSLOC  GET POSITION
0490 EB          0684          XCHG
0491 7A          0685          MOV   A,D
0492 CD 43 04    0686          CALL  ASSGET  CONVERT THE NUMBER
0495 70          0687          MOV   M,B
0496 23          0688          INX   H
0497 71          0689          MOV   M,C
0498 7B          0690          MOV   A,E
0499 CD 43 04    0691          CALL  ASSGET  NONW THE OTHER HALF
049C 23          0692          INX   H
049D 70          0693          MOV   M,B
049E 23          0694          INX   H
049F 71          0695          MOV   M,C
04A0 23          0696          INX   H
04A1 36 48       0697          MVI   M,'H'  PUT AN H ON THE END
04A3 23          0698          INX   H
04A4 36 20       0699          MVI   M,' '
04A6 C9          0700          RET
04A7             0701 *
04A7             0702 * ROUTINE TO SEE IF ADDRESS IS IN OBJECT
04A7             0703 *
04A7 E5          0704 CHEKR    PUSH  H
04A8 2A 40 05    0705          LHLD  POSLOC  GET POSITION
04AB 23          0706          INX   H
04AC 5E          0707          MOV   E,M
04AD 23          0708          INX   H
04AE 56          0709          MOV   D,M
04AF 2A 36 05    0710          LHLD  END     GET ADDR. OF END
04B2 7C          0711          MOV   A,H
04B3 BA          0712          CMP   D
04B4 CA BD 04    0713          JZ    FURTH   CHECK FURTHER
04B7 D2 C5 04    0714          JNC   LESS    INSIDE
04BA C3 D8 04    0715          JMP   OUT     OUTSIDE PROG
04BD 7D          0716 FURTH    MOV   A,L
04BE BB          0717          CMP   E
04BF CA D8 04    0718          JZ    OUT
04C2 DA D8 04    0719          JC    OUT
04C5 2A 34 05    0720 LESS     LHLD  BGNLOC  NOW SEE IF LESS THAN BEGINNING
04C8 7C          0721          MOV   A,H
04C9 BA          0722          CMP   D
04CA CA D0 04    0723          JZ    FUR1
04CD D2 D8 04    0724          JNC   OUT
04D0 7D          0725 FUR1     MOV   A,L     CHECK FURTHER
04D1 BB          0726          CMP   E
04D2 CA DA 04    0727          JZ    LTTN
04D5 DA DA 04    0728          JC    LTTN
04D8 E1          0729 OUT      POP   H       OUTSIDE, SO NO LOCAL LABEL
04D9 C9          0730          RET
04DA E1          0731 LTTN     POP   H       FINALLY FOUND ONE INSIDE
04DB 2B          0732          DCX   H
04DC 36 4C       0733          MVI   M,'L'  PUT ON THE "L" FOR LOCAL
04DE 23          0734          INX   H
04DF C9          0735          RET
04E0             0736 *
04E0             0737 * THIS ROUTINE SETS THE LINE LENGTH POINTER FOR ALS8
04E0             0738 *
04E0 1E 00       0739 FILES    MVI   E,0
04E2 2A 38 05    0740 FLIES    LHLD  BUFF+4  FIND BEGINNING OF FILE
04E5 2B          0741          DCX   H
04E6 44          0742          MOV   B,H     PUT HL IN BC
04E7 4D          0743          MOV   C,L
04E8 C3 F0 04    0744          JMP   FILIS   THE SETUP
04EB 44          0745 FILOS    MOV   B,H
04EC 4D          0746          MOV   C,L
04ED 03          0747          INX   B
04EE 23          0748          INX   H
04EF 23          0749          INX   H
04F0 16 01       0750 FILIS    MVI   D,1     TWO PASSES
04F2 7E          0751 FILIES   MOV   A,M
04F3 FE 0D       0752          CPI   0DH     END OF LINE ?
04F5 CA 0D 05    0753          JZ    FIL2
04F8 2B          0754          DCX   H
04F9 7E          0755          MOV   A,M
04FA FE 01       0756          CPI   1       EOF ?
04FC CA 13 05    0757          JZ    FIL3    YES
04FF 23          0758          INX   H
0500 7E          0759          MOV   A,M
0501 FE 20       0760          CPI   20H
```

```
0503 D2 08 05          0761          JNC     FIL1    <20H ?
0506 36 20             0762          MVI     M,' '   < 20H, SO PUT A SPACE
0508 14                0763 FIL1     INR     D
0509 23                0764          INX     H
050A C3 F2 04          0765          JMP     FILIES
050D 14                0766 FIL2     INR     D
050E 7A                0767          MOV     A,D
050F 02                0768          STAX    B       PUT THE LENGTH
0510 C3 EB 04          0769          JMP     FILOS
0513 AF                0770 FIL3     XRA     A
0514 BB                0771          CMP     E
0515 C2 60 E0          0772          JNZ     ALS8    DONE !!!!!!!!
0518 1C                0773          INR     E
0519 C3 E2 04          0774          JMP     FLIES
051C                   0775 *
051C                   0776 * <<<<   THIS IS THE TABLE AREA   >>>>
051C                   0777 *
051C 42 43 44 45       0778 TAB      ASC     'BCDEHLMA'  REGISTERS
     48 4C 4D 41
0524 08                0779 TAB3H    DB      8       ILLEGAL OP'S
0525 10                0780          DB      10H
0526 18                0781          DB      18H
0527 20                0782          DB      20H
0528 28                0783          DB      28H
0529 30                0784          DB      30H
052A 38                0785          DB      38H
052B CB                0786          DB      0CBH
052C D9                0787          DB      0D9H
052D DD                0788          DB      0DDH
052E ED                0789          DB      0EDH
052F FD                0790          DB      0FDH
0530 00                0791          DB      0
0531 00                0792 FLAG     DB      0       LABEL OR NO LABEL
0532                   0793 BUFLOC   DS      2       LOCATION IN BUFF
0534                   0794 BUFF     DS      6       BUFFER FOR ADDRESSES
053A                   0795 INSLOC   DS      2       INSTRUCTION LOCATION
053C                   0796 STABLOC  DS      2       LOCATION IN SYMBOL TABLE
053E 00                0797 DSTLOC   DB      0       DESTINATION REG.
053F 00                0798 LOCLOC   DB      0       # OF BYTES
0540 00 00             0799 POSLOC   DW      0       POSITION IN OBJECT
0542 00                0800 SRCLOC   DB      0       SOURCE REG.
0543 00                0801 TYPE     DB      0       TYPE (i.e SIMPLE,DEST,SOURCE)
0544 00                0802 OPLOC    DB      0       THE OP
0545 00 00             0803 MNLOC    DW      0       POSITION OF THE MNEMONIC
0547                   0804 PLOC     DS      2       POSITION IN FILE
0549                   0805 ALS8     EQU     0E060H  ALS8 RETURN POINT
0549                   0806 ERROR    EQU     ALS8    SAME FOR ERRORS
0549                   0807 BGNLOC   EQU     BUFF    BEGINNING OF OBJECT
0549                   0808 PLO      EQU     BUFF+4  BEGINNING OF SOURCE
0549                   0809 END      EQU     BUFF+2  END OF OBJECT
0549 00                0810 CLOCLOC  DB      0
054A 45 4E 54 45       0811 INST     ASC     'ENTER AS A 4 DIGIT HEX ADDRESS (i.e. 074F)'
     52 20 41 53
     20 41 20 34
     20 44 49 47
     49 54 20 48
     45 58 20 41
     44 44 52 45
     53 53 20 28
     69 2E 65 2E
     20 30 37 34
     46 29
0574 00                0812          DB      0
0575 42 45 47 49       0813          ASC     'BEGINNING ADDRESS OF OBJECT CODE ?'
     4E 4E 49 4E
     47 20 41 44
     44 52 45 53
     53 20 4F 46
     20 4F 42 4A
     45 43 54 20
     43 4F 44 45
     20 3F
0597 00                0814          DB      0
0598 45 4E 44 20       0815          ASC     'END ADDRESS OF OBJECT CODE ?'
     41 44 44 52
     45 53 53 20
     4F 46 20 4F
     42 4A 45 43
     54 20 43 4F
     44 45 20 3F
05B4 00                0816          DB      0
05B5 44 45 53 54       0817          ASC     'DESTINATION OF SOURCE ?'
     49 4E 41 54
     49 4F 4E 20
     4F 46 20 53
     4F 55 52 43
     45 20 3F
05CC 00                0818          DB      0
05CD 2A 2A 2A 2A       0819 PCHL     ASC     '******<<< WARNING >>>******'
     2A 2A 3C 3C
     3C 20 57 41
     52 4E 49 4E
     47 20 3E 3E
     3E 2A 2A 2A
     2A 2A 2A
05E8 00                0820          DB      0
05E9                   0821 *
05E9                   0822 * THIS IS THE OPCODE TABLE
05E9                   0823 *
05E9 00                0824 OPTAB    DB      0
05EA 3A                0825          DB      3AH
05EB 03                0826          DB      3
05EC 4C 44 41          0827          ASC     'LDA'
05EF 00                0828          DB      0
05F0 32                0829          DB      32H
05F1 03                0830          DB      3
05F2 53 54 41          0831          ASC     'STA'
05F5 00                0832          DB      0
05F6 2A                0833          DB      2AH
05F7 03                0834          DB      3
05F8 4C 48 4C 44       0835          ASC     'LHLD'
05FC 00                0836          DB      0
05FD 22                0837          DB      22H
05FE 03                0838          DB      3
05FF 53 48 4C 44       0839          ASC     'SHLD'
0603 00                0840          DB      0
0604 EB                0841          DB      0EBH
0605 01                0842          DB      1
0606 58 43 48 47       0843          ASC     'XCHG'
060A 00                0844          DB      0
060B C6                0845          DB      0C6H
060C 02                0846          DB      2
060D 41 44 49          0847          ASC     'ADI'
0610 00                0848          DB      0
0611 CE                0849          DB      0CEH
0612 02                0850          DB      2
0613 41 43 49          0851          ASC     'ACI'
0616 00                0852          DB      0
0617 D6                0853          DB      0D6H
0618 02                0854          DB      2
0619 53 55 49          0855          ASC     'SUI'
061C 00                0856          DB      0
061D DE                0857          DB      0DEH
061E 02                0858          DB      2
061F 53 42 49          0859          ASC     'SBI'
0622 00                0860          DB      0
0623 27                0861          DB      27H
0624 01                0862          DB      1
0625 44 41 41          0863          ASC     'DAA'
0628 00                0864          DB      0
0629 E6                0865          DB      0E6H
062A 02                0866          DB      2
062B 41 4E 49          0867          ASC     'ANI'
062E 00                0868          DB      0
062F EE                0869          DB      0EEH
0630 02                0870          DB      2
0631 58 52 49          0871          ASC     'XRI'
0634 00                0872          DB      0
0635 F6                0873          DB      0F6H
0636 02                0874          DB      2
0637 4F 52 49          0875          ASC     'ORI'
063A 00                0876          DB      0
063B FE                0877          DB      0FEH
063C 02                0878          DB      2
063D 43 50 49          0879          ASC     'CPI'
0640 00                0880          DB      0
0641 07                0881          DB      07H
0642 01                0882          DB      1
0643 52 4C 43          0883          ASC     'RLC'
0646 00                0884          DB      0
0647 0F                0885          DB      0FH
0648 01                0886          DB      1
0649 52 52 43          0887          ASC     'RRC'
064C 00                0888          DB      0
064D 17                0889          DB      17H
064E 01                0890          DB      1
064F 52 41 4C          0891          ASC     'RAL'
0652 00                0892          DB      0
0653 1F                0893          DB      1FH
```

```
0654 01             0894        DB    1
0655 52 41 52       0895        ASC   'RAR'
0658 00             0896        DB    0
0659 2F             0897        DB    2FH
065A 01             0898        DB    1
065B 43 4D 41       0899        ASC   'CMA'
065E 00             0900        DB    0
065F 3F             0901        DB    3FH
0660 01             0902        DB    1
0661 43 4D 43       0903        ASC   'CMC'
0664 00             0904        DB    0
0665 37             0905        DB    37H
0666 01             0906        DB    1
0667 53 54 43       0907        ASC   'STC'
066A 00             0908        DB    0
066B C3             0909        DB    0C3H
066C 03             0910        DB    3
066D 4A 4D 50       0911        ASC   'JMP'
0670 00             0912        DB    0
0671 CD             0913        DB    0CDH
0672 03             0914        DB    3
0673 43 41 4C 4C    0915        ASC   'CALL'
0677 00             0916        DB    0
0678 C9             0917        DB    0C9H
0679 01             0918        DB    1
067A 52 45 54       0919        ASC   'RET'
067D 00             0920        DB    0
067E E9             0921        DB    0E9H
067F 01             0922        DB    1
0680 50 43 48 4C    0923        ASC   'PCHL'
0684 00             0924        DB    0
0685 E3             0925        DB    0E3H
0686 01             0926        DB    1
0687 58 54 48 4C    0927        ASC   'XTHL'
068B 00             0928        DB    0
068C F9             0929        DB    0F9H
068D 01             0930        DB    1
068E 53 50 48 4C    0931        ASC   'SPHL'
0692 00             0932        DB    0
0693 DB             0933        DB    0DBH
0694 02             0934        DB    2
0695 49 4E          0935        ASC   'IN'
0697 00             0936        DB    0
0698 D3             0937        DB    0D3H
0699 02             0938        DB    2
069A 4F 55 54       0939        ASC   'OUT'
069D 00             0940        DB    0
069E FB             0941        DB    0FBH
069F 01             0942        DB    1
06A0 45 49          0943        ASC   'EI'
06A2 00             0944        DB    0
06A3 F3             0945        DB    0F3H
06A4 01             0946        DB    1
06A5 44 49          0947        ASC   'DI'
06A7 00             0948        DB    0
06A8 76             0949        DB    76H
06A9 01             0950        DB    1
06AA 48 4C 54       0951        ASC   'HLT'
06AD 00             0952        DB    0
06AE 00             0953        DB    0
06AF 01             0954        DB    1
06B0 4E 4F 50       0955        ASC   'NOP'
06B3 00             0956        DB    0
06B4 08             0957        DB    08H
06B5 00             0958        DB    0
06B6 0A             0959        DB    0AH
06B7 01             0960        DB    1
06B8 4C 44 41 58    0961        ASC   'LDAX'
06BC 00             0962        DB    0
06BD 02             0963        DB    2
06BE 01             0964        DB    1
06BF 53 54 41 58    0965        ASC   'STAX'
06C3 00             0966 OPTB   DB    0
06C4 01             0967        DB    1
06C5 03             0968        DB    3
06C6 4C 58 49       0969        ASC   'LXI'
06C9 00             0970        DB    0
06CA 03             0971        DB    3
06CB 01             0972        DB    1
06CC 49 4E 58       0973        ASC   'INX'
06CF 00             0974        DB    0
06D0 0B             0975        DB    0BH
06D1 01             0976        DB    1
06D2 44 43 58       0977        ASC   'DCX'
06D5 00             0978        DB    0
06D6 09             0979        DB    9H
06D7 01             0980        DB    1
06D8 44 41 44       0981        ASC   'DAD'
06DB 00             0982        DB    0
06DC C5             0983        DB    0C5H
06DD 01             0984        DB    1
06DE 50 55 53 48    0985        ASC   'PUSH'
06E2 00             0986        DB    0
06E3 C1             0987        DB    0C1H
06E4 01             0988        DB    1
06E5 50 4F 50       0989        ASC   'POP'
06E8 00             0990        DB    0
06E9 10             0991        DB    10H
06EA 00             0992        DB    0
06EB C4             0993        DB    0C4H
06EC 03             0994        DB    3
06ED 43             0995        ASC   'C'
06EE 00             0996        DB    0
06EF C2             0997        DB    0C2H
06F0 03             0998        DB    3
06F1 4A             0999        ASC   'J'
06F2 00             1000        DB    0
06F3 C0             1001        DB    0C0H
06F4 01             1002        DB    1
06F5 52             1003        ASC   'R'
06F6 00             1004        DB    0
06F7 C7             1005        DB    0C7H
06F8 01             1006        DB    1
06F9 52 53 54       1007        ASC   'RST'
06FC 00             1008        DB    0
06FD 05             1009        DB    5
06FE 01             1010        DB    1
06FF 44 43 52       1011        ASC   'DCR'
0702 00             1012        DB    0
0703 04             1013        DB    4
0704 01             1014        DB    1
0705 49 4E 52       1015        ASC   'INR'
0708 00             1016        DB    0
0709 06             1017        DB    6
070A 02             1018        DB    2
070B 4D 56 49       1019        ASC   'MVI'
070E 00             1020        DB    0
070F 18             1021        DB    18H
0710 00             1022        DB    0
0711 80             1023        DB    80H
0712 01             1024        DB    1
0713 41 44 44       1025        ASC   'ADD'
0716 00             1026        DB    0
0717 88             1027        DB    88H
0718 01             1028        DB    1
0719 41 44 43       1029        ASC   'ADC'
071C 00             1030        DB    0
071D 90             1031        DB    90H
071E 01             1032        DB    1
071F 53 55 42       1033        ASC   'SUB'
0722 00             1034        DB    0
0723 98             1035        DB    98H
0724 01             1036        DB    1
0725 53 42 42       1037        ASC   'SBB'
0728 00             1038        DB    0
0729 A0             1039        DB    0A0H
072A 01             1040        DB    1
072B 41 4E 41       1041        ASC   'ANA'
072E 00             1042        DB    0
072F A8             1043        DB    0A8H
0730 01             1044        DB    1
0731 58 52 41       1045        ASC   'XRA'
0734 00             1046        DB    0
0735 B0             1047        DB    0B0H
0736 01             1048        DB    1
0737 4F 52 41       1049        ASC   'ORA'
073A 00             1050        DB    0
073B B8             1051        DB    0B8H
073C 01             1052        DB    1
073D 43 4D 50       1053        ASC   'CMP'
0740 00             1054        DB    0
0741 20             1055        DB    20H
0742 01             1056        DB    1
0743 4D 4F 56       1057        ASC   'MOV'
0746 00             1058        DB    0
0747 28             1059        DB    28H
READY
```

Debug

```
0000                    0001 *
0000                    0002 *
0000                    0003 *
0000                    0004 *    PROGRAM TO REPAIR CRASHED ALS8 FILES
0000                    0005 *    WRITTEN BY DON VAN DYKE
0000                    0006 *
0000                    0007 *
0000                    0008 *
0000 1E 00              0009        MVI   E,0     # OF THE PASS
0002 2A 05 D0           0010 LOOP   LHLD  POINTER  GET THE BEGINNING OF THE FILE
0005 2B                 0011        DCX   H        MINUS 1
0006 44                 0012        MOV   B,H     PUT A COPY IN BC
0007 4D                 0013        MOV   C,L
0008 C3 10 00           0014        JMP   INIT    INITIALIZE THE PROGRAM
000B 44                 0015 LOOP2  MOV   B,H     COPY INTO BC
000C 4D                 0016        MOV   C,L
000D 03                 0017        INX   B       INCREMENT THE POINTER
000E 23                 0018        INX   H       AND THIS BY TWO
000F 23                 0019        INX   H
0010 16 01              0020 INIT   MVI   D,1
0012 7E                 0021 LABL3  MOV   A,M     GET THE BYTE
0013 FE 0D              0022        CPI   0DH     END OF THE LINE ?
0015 CA 2D 00           0023        JZ    ENDLN   YES
0018 2B                 0024        DCX   H       NO, BACK UP
0019 7E                 0025        MOV   A,M     GET THE BYTE
001A FE 01              0026        CPI   1       END OF FILE ?
001C CA 33 00           0027        JZ    EOF     YES, CHECK PASS NUMBER
001F 23                 0028        INX   H       OTHERWISE CONTINUE
0020 7E                 0029        MOV   A,M     GET THE BYTE
0021 FE 20              0030        CPI   020H    CONTROL CHARACTER ?
0023 D2 28 00           0031        JNC   OK      CHARACTER IS O.K.
0026 36 20              0032        MVI   M,' '   PUT A BLANK IN THE FILE
0028 14                 0033 OK     INR   D       INCREMENT LENGTH
0029 23                 0034        INX   H       POINT TO NEXT
002A C3 12 00           0035        JMP   LABL3   DO IT
002D 14                 0036 ENDLN  INR   D       INCREMENT LENGTH
002E 7A                 0037        MOV   A,D     GET THE LENGTH IN A
002F 02                 0038        STAX  B       STORE IT IN THE FILE
0030 C3 0B 00           0039        JMP   LOOP2   NEXT LINE
0033 AF                 0040 EOF    XRA   A       A=0
0034 BB                 0041        CMP   E       # OF PASS = 0 ?
0035 C2 D3 E6           0042        JNZ   FCHK    GO TO ALS8 FCHK ROUTINE
0038 1C                 0043        INR   E       PASS = PASS +1
0039 C3 0B 00           0044        JMP   LOOP2   DO THE SECOND PASS
003C                    0045 FCHK   EQU   0E6D3H
003C                    0046 POINTER EQU  0D005H
READY
```

Adump

```
0000                    0001 *
0000                    0002 *
0000                    0003 *
0000                    0004 *    ROUTINE TO DUMP ASCII EQUIVALENT OF MEMORY
0000                    0005 *    MODIFIED VERSION OF DUMP ROUTINE FROM SOLOS (TM)
0000                    0006 *    COPYRIGHT 1977 PROCESSOR TECHNOLOGY CORPORATION.
0000                    0007 *          MODIFIED BY DON VAN DYKE
0000                    0008 *
0000                    0009 *
0000                    0010 *
0000 CD 3A C3           0011 DUMP   CALL   SCONV  SCAN TO FIRST ADDRESS AND CONVERT IT
0003 E5                 0012        PUSH   H      SAVE THE VALUE
0004 CD 10 C3           0013        CALL   PSCAN  SEE IF SECOND WAS GIVEN
0007 D1                 0014        POP    D      GET BACK START
0008 EB                 0015        XCHG   .      HL HAS START, DE HAS END
0009 CD F9 C2           0016 DLOOP  CALL   CRLF
000C CD E8 C3           0017        CALL   ADOUT  OUTPUT ADDRESS
000F CD 06 C4           0018        CALL   BOUT   ANOTHER SPACE TO KEEP IT PRETTY
0012 0E 10              0019        MVI    C,16   VALUES PER LINE
0014 7E                 0020 DLP1   MOV    A,M    GET THE CHR
0015 C5                 0021        PUSH   B      SAVE VALUE COUNT
0016 FE 20              0022        CPI    20H    < 20H ?
0018 D2 1D 00           0023        JNC    DOWN
001B 3E 2E              0024        MVI    A,'.'  YES, MAKE IT A DOT
001D FE 7F              0025 DOWN   CPI    7FH    >= 7FH ?
001F DA 24 00           0026        JC     DN1    NO
0022 3E 2E              0027        MVI    A,'.'  YES, MAKE THIS A DOT
0024 47                 0028 DN1    MOV    B,A    PUT IT IN B
0025 CD 19 C0           0029        CALL   SOUT   PUT IT OUT
0028 06 20              0030        MVI    B,' '
002A CD 19 C0           0031        CALL   SOUT   PUT OUT A BLANK
002D 7D                 0032        MOV    A,L    COMPARE DE & HL
002E 93                 0033        SUB    E
002F 7C                 0034        MOV    A,H
0030 9A                 0035        SBB    D
0031 D2 C9 C1           0036        JNC    COMND  ALL DONE
0034 C1                 0037        POP    B      VALUES PER LINE
0035 23                 0038        INX    H
0036 0D                 0039        DCR    C      BUMP THE LINE COUNT
0037 C2 14 00           0040        JNZ    DLP1   NOT ZERO IF MORE FOR THIS LINE
003A C3 09 00           0041        JMP    DLOOP  DO A LFCR BEFOR THE NEXT
003D                    0042 SOUT   EQU    0C019H
003D                    0043 SCONV  EQU    0C33AH
003D                    0044 COMND  EQU    0C1C9H
003D                    0045 ADOUT  EQU    0C3E8H
003D                    0046 BOUT   EQU    0C406H
003D                    0047 PSCAN  EQU    0C310H
003D                    0048 CRLF   EQU    0C2F9H
READY
```

Exam

```
0000                    0001 *
0000                    0002 *
0000                    0003 *    PROGRAM TO DISPLAY ASCII OF MEMORY CONTENTS
0000                    0004 *    WRITTEN BY DON VAN DYKE
0000                    0005 *
0000                    0006 *
0000 AF                 0007        XRA    A       INITAILIZE SCREEN
0001 D3 FE              0008        OUT    0FEH    SET SOL
0003 D3 C8              0009        OUT    0C8H    SET VDM
0005 01 00 F8           0010        LXI    B,OFFSET  (-2048)
0008 11 00 00           0011        LXI    D,0     BEGINNING OF MEMORY
000B CD 39 00           0012 NEXT   CALL   DISP    DISPLAY 1K
000E CD 1F C0           0013 IN     CALL   SINP    GET CHARACTER FROM SOLOS/CUTER
0011 CA 0E 00           0014        JZ     IN      READY ?
0014 E6 7F              0015        ANI    7FH     STRIP PARITY
0016 FE 1B              0016        CPI    1BH     ESCAPE ?
0018 CA 04 C0           0017        JZ     RETRN   YES, RETURN TO SOLOS
001B FE 09              0018        CPI    9       TAB OR CTRL-I ?
001D CA 2D 00           0019        JZ     TAB     YES
0020 FE 17              0020        CPI    17H     UP ARROW OR CTRL-W ?
0022 CA 0B 00           0021        JZ     NEXT
0025 FE 1A              0022        CPI    1AH     DOWN ARROW OR CTRL-Z ?
0027 CA 36 00           0023        JZ     BACK
002A C3 0E 00           0024        JMP    IN      IF NONE OF THESE, THEN TRY ANOTHER CHARACTER
002D                    0025 *
002D                    0026 * ROUTINE TO SKIP 8K AND DISPLAY
002D                    0027 *
002D EB                 0028 TAB    XCHG
002E 11 00 20           0029        LXI    D,2000H  8K TO SKIP
0031 19                 0030        DAD    D       ADD THE OFFSET
0032 EB                 0031        XCHG
0033 C3 39 00           0032        JMP    DISP    DISPLAY IT
0036                    0033 *
0036 EB                 0034 BACK   XCHG
0037 09                 0035        DAD    B       SUBTRACT 2K
0038 EB                 0036        XCHG
0039                    0037 *
0039 21 00 CC           0038 DISP   LXI    H,VIDMEM  BEGINNING OF DISPLAY
003C 1A                 0039 AGAIN  LDAX   D       GET THE CHARACTER FROM MEMORY
003D 77                 0040        MOV    M,A     PUT IT ON THE SCREEN
003E 13                 0041        INX    D       NEXT MEMORY LOCATION
003F 23                 0042        INX    H       NEXT SCREEN LOCATION
0040 7C                 0043        MOV    A,H     GET HIGH ORDER ADDRESS
0041 FE D0              0044        CPI    0D0H    END OF SCREEN ?
0043 C2 3C 00           0045        JNZ    AGAIN   NO, DO IT AGAIN
0046 C3 0E 00           0046        JMP    IN      GET ANOTHER CHARACTER
0049                    0047 SINP   EQU    0C01FH
0049                    0048 VIDMEM EQU    0CC00H
0049                    0049 RETRN  EQU    0C004H
0049                    0050 OFFSET EQU    0F800H
READY
```

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 22 **February 1978** **Volume 3, Issue 2**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

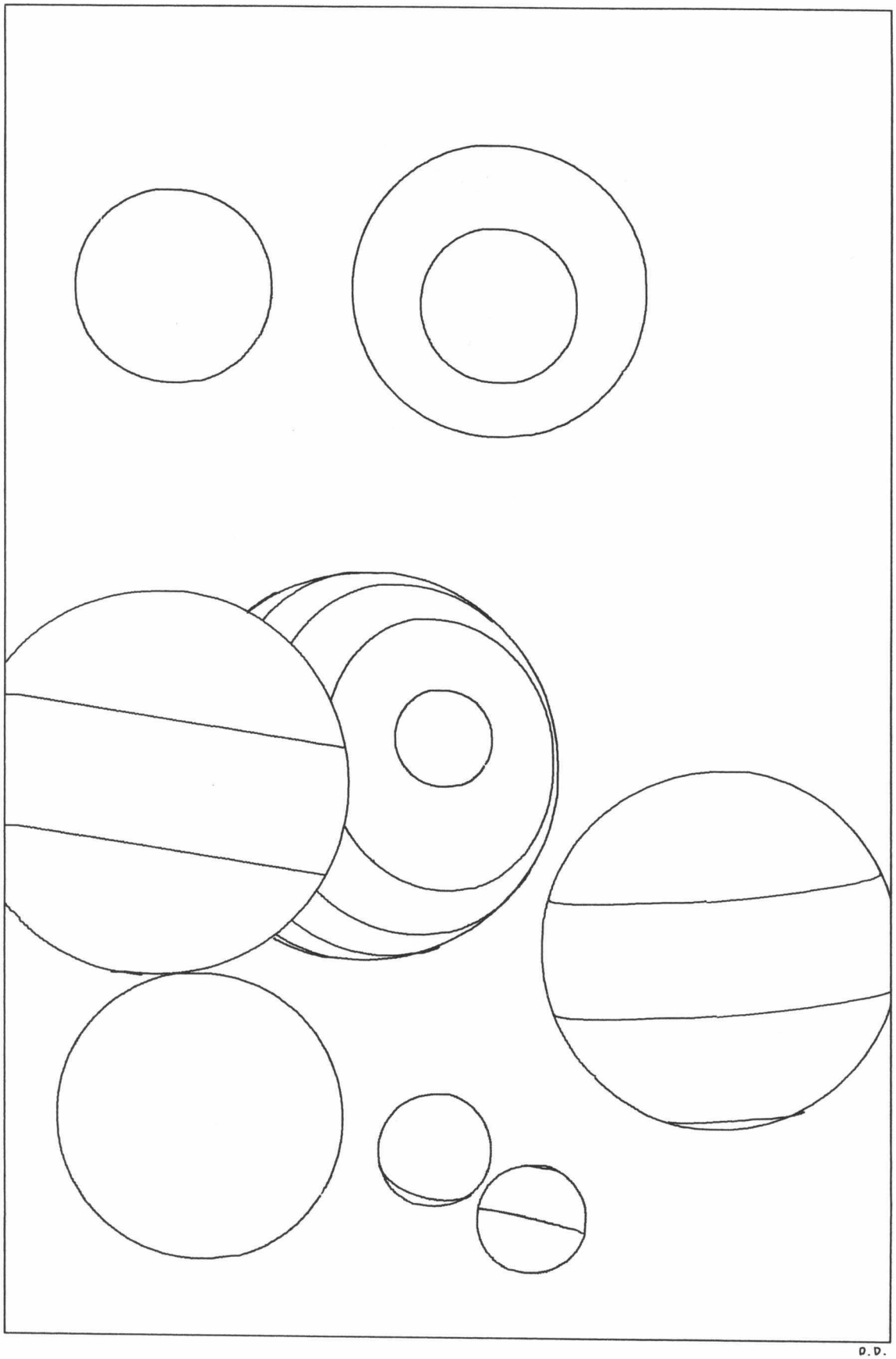
ET TU, KLATU?, Stanford News Service 4
Quasar Industries' robot called "a fraud" by John McCarthy
The Last Rehash of Beau Hex? *letters in the unending(?) hex/octal debate* 45

6502 SOFTWARE
Decoding Efficiency & Speed: Pros & Cons of Table Look-up, H.T. Gordon 5

8080 & Z-80 SOFTWARE
MITS 4.0/4.1 BASIC Searcher/Patcher, James Monagan 8
Z-80 RAM Tester, John MacDougall 42
Mini Editor for 8080 PILOT, Tim Scully 43

CODE RELOCATION
A Simple Technique for Static Relocation of Absolute Machine Code, Gary A. Kildall 10
the creator of PL/M and CP/M tells how
Relocation is Not Just Moving Programs, Richard L. Roth 14

TV DAZZLER SOFTWARE
PCC's Dazzler Contest Winners (*at long, long last*), Robert B. Jacobsen, Exec. Dir., PCC 21
Absolutely Dazzling Dazzleware, Gordon French & Gregory Yob 22
Dazzler Driver Subroutine and a "Chase", Phillip Simpson 30



D.D.

© David Dameron 1979

MORE GOOD ADVICE ON MAIL ORDERING

Gentlemen: *Received: 78 Jan 24*

In regard to your recent article on slow or non-existent delivery on the part of some merchandisers in the home computer field, I've devised some defensive techniques that work well.

Always write a letter in advance asking if the product is available for immediate delivery provided that a certified check or money order is sent.

If you receive no reply, don't send in an order. If you are really anxious to purchase the product, you might try a follow-up letter just in case the first one got lost somewhere along the way. If you still hear nothing, forget it.

Once you have received a letter indicating that the product is available, send in your order with certified check or money order. Keep a copy of the dated letter and also photocopy the check. In your cover letter clearly state that you wish shipment within 2 weeks (or whatever time you feel is reasonable). Otherwise you want your money refunded.

If you do not receive the product on time, send a follow-up letter. Courteously remind the merchandiser that shipment by a certain date had been promised, and refer to your previous correspondence by date. Keep a copy of this letter also.

If you do not receive a reply within two, weeks, send a not-so-courteous letter by certified mail, return receipt requested (costs about a dollar). Tell the supplier that you want an immediate refund or you will report the matter to the U.S. Postal Service, the Attorney General of your state, and to the magazine in which the advertisement appeared. Keep a number of copies of this letter, along with the signed delivery receipt.

Wait another two weeks, and, if you still have not received the product or a refund, proceed to report the matter to the U.S. Postal Service, Attorney General, etc. Send copies of correspondence with the merchandiser. Send the merchandiser copies of your complaint letters. You might also send copies to the Better Business Bureau in the merchandiser's home town.

If the above sounds like it was written by a paranoid, I can only claim long and bitter experience. My method works so long as the supplier is not an out and out fraud. Before sending a really substantial check, I would ask the supplier for credit references, if I was at all dubious.

It is also extremely effective to send complaints to the supplier's bank (appears on the back of the cancelled check if deposited) and to credit rating bureaus, as well as the wholesalers he deals with, if known. These are effective measures in drastic cases.

Sincerely,
John D. Hirsch

Berme Rd
Kerhonkson, NY 12446

RASKIN JOINS APPLE: LEAVES *DR. DOBB'S*

Announcement *Dated: 78 Jan 18*

Jef Raskin has been our itenerant product evaluator for some time. Part of the reason he could do that is that he was an independent (very) computer consultant. He has recently taken a position as Director of Publications with Apple Computers of Cupertino, California. As such, he can no longer be considered to be "independent." Therefore, he will no longer be submitting "product evaluations" to *DDJ*. It is likely, however, that we will–from time to time–publish "product opinions" authored by Jef.

O TEMPORA, O MORES!

Vector Graphic Inc.
790 Hampshire Road, A-B
Westlake Village, CA 91361

Dear Vector *Received: 78 Jan 30*

You are without a doubt the most sneaky, conniving company I have had to deal with in the microprocessor field. It is disgusting.

My letter to you, written (and clearly dated!) 22 January, 1978 (a Sunday) was not mailed until 23 January 1978. Yet you managed to somehow get ahold of it on the 24th and get your reply into my hands by 2 PM on the 25th, a gap of almost *4 whole days!* What's more, the material promised by you did not arrive until the 26th, 2 whole days later, which was *exactly as promised!*

Now my request was fairly reasonable and should have been simple to answer. But oh no! you have to set someone down who completely understood my request, have them answer my letter specifically point by point and then cause them to mail the answer without cramming the envelope with useless and *unrequested* literature.

Well let me tell you people up there at Vector that while I may have appeared to be a mild mannered computer hobbyist working for a Great Metropolitan Newspaper, I am in fact, a systems analyst with more than 20 years experience in all aspects of data processing. . . And I'm not going to stand still for conduct like yours!

If I can write to MITS, Micropolis, Digital Group and Dyna-Byte and never receive *any reply at all*, and if IMSAI and Poly Morphic and Space Byte need from thirty to ninety days to get a bale of unwanted literature to me instead of an answer to a specific question, then the obvious question to me is WHAT'S WRONG WITH YOU PEOPLE UP THERE?

Don't you people realize that a response like I received is like waving a yellow flag in front of a gorilla? You may think you have the world by the toe-nails and can afford to trifle with guys like me but you've made a serious mistake this time. I'm really going to give you the business!

A copy of this letter is going directly to Tom Williams at *Dr. Dobb's Journal*. There is no doubt in my mind that Tom will publish the letter and let everyone know just what kind of shenanigans you guys are up to!

Acryptically yours
Robert S. Hoover

First Heath returns phone calls, now Vector Graphic answers letters. What is *this world coming to? –TRW*

SOLUS PREPARES SOFTWARE DIRECTORY

News Release *78: Jan 29*

The SOL Users' Society (SOLUS) is preparing a directory of programs that are tailored to run under the SOLOS and CUTER operating systems. The list will be published in the *SOLUS Newsletter* (at no charge to the software vendors).

To qualify for listing in the directory, a program must be compatible with both SOLOS and CUTER and be available on CUTS format or Kansas City standard cassettes. It must be well-documented for the user, extensively tested and debugged, and supported by a warranty to repair or replace for at least 3 months. Programs written in higher-level languages will be accepted if the interpreter meets these specifications.

Software vendors should send a brief description and ordering information to SOLUS NEWS, Box 23471, San Jose, CA 95153.

ET TU, KLATU?

Stanford University News Release *Received: 77 Nov 14*

A company which claims it is ready to manufacture and sell truly intelligent household androids, or robots, within the next two years–to the tune of $4,000 apiece–has the nation's artificial intelligence experts up in arms. The experts unanimously state that the claims are fraudulent. The company, Quasar Industries located in Rutherford, NJ, is currently touring shopping centers around the country with a model of an android which they claim comprehends 4,000 words of vocabulary and can vacuum, wash dishes, and teach the kids French. Crowds in the sundry department stores are dazzled, according to numerous press reports. This robot is a fake, according to scientists from Carnegie-Mellon University in Pittsburgh, PA. The scientists recently undetook a first-hand investigation of the matter and found the robot to be a "radio-controlled puppet."

Quasar Industries sees it differently. "We programmed the robot to approximate human behavior," said Anthony Reichelt, president of the company. "The only limitations to what it can do is the boundary of imigination and the cost factor." Quasar Industries has no relationship with a brand of television set formerly manufactured by Motorola and now made by a Japanese company.

Numerous publications, including *Newsweek, Parade,* and *Newsday*, have in recent months printed articles about the robot–called Sam Strugglegear–in which the claims made by Quasar Industries are reported as fact. "It is a fraud that has gone on far too long," says John McCarthy, director of Stanford University's Artificial Intelligence Laboratory. McCarthy is one of the earliest pioneers in the field. The technology is nowhere near being able to manufacture a robot which can do all, let alone safely, what Strugglegear's owners say it can do, McCarthy said in an interview. The company has some sort of expensive, remote-controlled gimmick, he said, which in no way possesses the sort of intelligence claimed for it. "The press has been extraordinarily irresponsible in not bothering to ask the opinion of artificial intelligence experts on the matter. They don't have to believe us, but they should at least ask." On Oct. 24, according to Carnegie-Mellon graduate students Brian Reid and Mark Gos, a well-known department store in the heart of Pittsburgh advertised the appearance of a "domestic robot." "Knowing of CMU's pioneering work, particularly in the field of speech recognition, various friends called CMU to ask how this robot might be so much better at speech recognition than our talented and dedicated research teams," they wrote in a memorandum sent to colleagues at Stanford, MIT, and elsewhere. "Rising to the challenge, four courageous members of our department went downtown to investigate. They found a frightening sight: in the men's department, among the three-piece suits, was a 5' 2" image of an aerosol can on wheels, talking animatedly to the crowd."

Upon closer investigation, the CMU experts saw that the robot was surprisingly crude. "It didn't seem able to tell that an object was blocking its path. Covering the faceplate did not change its behavior at all. Since the robot seemed able to navigate around the room without hitting anything, we found it quite curious that it had no detectable sensory reactions. Feeling more dubious, we began to look around the room for evidence of remote control. Lo and behold, about 10 feet from the robot, standing in the crowd, we found a man in a blue suit with his hand held contemplatively to his mouth. After watching for a while we noticed that whenever the robot was talking, the man in the blue suit could be seen muttering into his hand."

At a second store, "they found a furtive-looking and rather disagreeable person loitering in the back of the room. He was carrying an airline flight bag with his hand stuck down in the bag." The investigators challenged the man, suspecting that he was controlling the movements of Sam Strugglegear. The man became "very excited and called for store officials to come get us away from him," the CMU scientists said. Quasar Industries still maintains they will be ready to market a domestic robot within 18 to 22 months. It will, they said, be able to perform 12 household tasks such as "serve your dinner, vacuum your rugs, babysit for your kids, and insult your enemies." Such claims amuse, annoy, and sometimes downright anger artificial intelligence experts.

Stanford graduate student Paul Martin, who is writing a thesis on the problems of machines understanding natural dialogue, believes much of the gullibility on behalf of the public, the press, and department store managers stems from recent technological innovations, such as hand-held calculators and space missions. But there is a huge difference, Martin said, between having a machine which can recognize separate, simple words (this is possible now) and one that can understand the flow of language as spoken naturally. "Everyone in the field would agree there are important scientific problems waiting to be solved before we can get to the real thing. On how long these problems will take to solve, you find a diversity of opinion. I tend to hedge by saying it will take between five and 500 years. If you ask me to center it, I would say 40 years. The basic problems are how to make computers recognize patterns in a noisy environment and how to go about representing common sense knowledge about the world, how to put that in a computer. In fact, what common sense knowledge about the world is there? How can a machine be taught to solve problems?

"One way these problems express themselves is in safety issues. A household robot has to be very reliable, mechanically and intellectually. It must distinguish between children and animals as objects. It cannot think the baby is part of a stack of pillows. It has to have a wide knowledge of different kinds of objects to be intelligent." In response to a reporter's question whether the robot could tell a cup from a saucer, Reichelt responded that it didn't have to because it didn't take them off the tray. McCarthy then argued that, in that case, the Quasar robot is too limited to be really useful. "Say it could carry a dinner tray from the kitchen to the dining room," McCarthy said. "Someone in the kitchen would have to prepare the tray. The robot would be on a tread, according to the company. But it still couldn't be done safely. It could trip on a rug. It could run over something. It could fall down the stairs."

Sam Strugglegear, also called KLATU in some news reports, is not yet out of the news. A few weeks ago, while performing at a shopping center in Philadelphia, the imitation robot reportedly frightened a 10-month-old child. The child's mother has filed suit for $100,000 in damages, saying that the robot "accosted, touched, assaulted, and terrorized" her child.

The CMU scientists were reminded by these recent events of another famous "intelligent" robot build by Baron Wolfgang van Kempeler in 1769. This robot, known as "the Turk," would trounce people at chess, all while rolling its eyes and nodding its head. Thousands of people, including Edgar Allen Poe and Napoleon, examined it thoroughly and could not figure out how it worked. It wasn't until more than 70 years later that the truth was discovered. The robot cleverly concealed a dwarf who was a talented chess player. Kempeler's description of his own robot, circa 1771, the CMU investigators said, is probably the best summary of Sam Strugglegear: "A mere bagatelle, not without merit in point of mechanism, but whose effects appear marvelous only from the boldness of conception and the clever choice of methods adopted for promoting the illusion."

Decoding Efficiency & Speed:

Pros & Cons of Table Look-Up

BY H. T. GORDON
College of Agricultural Sciences
University of California
Berkeley, CA 94720

Dear Jim,

Received: 78 Jan 15

While some of your readers may have become as weary as I am with my repeated attacks on the 650X legality-bytecount problem, this is in a way a programming guinea-pig for working-out optimization principles, relevant to all *problems. Even your non-650X readers may find something of value in the enclosed MS, as it touches on some universal problems: the chioce between decoding-logic or table-look-up programs, the extent to which subroutines should be designed to be non-destructive and purely-informative, and the uses of antibugging code. My ultimate goal is to develop scanning-debugging to a higher level. For this, the basic components of the debugger must be fast, efficient, and above al bug-free.*
Sincerely,
H. T. Gordon

The question of how to optimize a decoding subroutine, previously touched on in my notes in issues #17 and #19 of *DDJ* (1977) using decoding of the number of bytes required by 650X opcodes as an example, is re-opened by 2 new solutions: CNTBYT (written by Jim Butterfield and reproduced here with his permission) and BYTCNT (my own response to his challenge). Both optimize coding efficiency, and it is unlikely that a solution in fewer than 28 bytes exists. Both sort the 151 legal opcodes into the same 6 groups as my earlier subroutine BYTNUM. However, CNTBYT classes illegals of type X_0B as 3 byte codes while the others class them as 2 byte codes (which is how the 650X control unit executes them, but this is immaterial since all routines class illegals of type X2 as 2 byte codes although most of them are not executable).

The important differences are in structure and timing. CNTBYT uses an elegant and highly instructive linear sequence of bit-manipluations that cause each of 5 classes of opcodes in turn to become zero and exit, while the 6th class exits as non-zero. BYTCNT emulates BYTNUM by first sorting into subclasses X(0-7) and X(8-F) by a BIT test of bit 3, then shifts to destructive logic in each branch to attain higher code efficiency. Unlike BYTNUM, both new routines destroy the accumulator.

All routines have different path lengths (measured in clock cycles from entry to exit, including the 6 cycles for the main program JSR and 6 cycles for the RTS) for each of the 6 opcode classes. An index of relative speed (not weighted for difference in opcode usage frequency) is easily calculated by dividing Σ (N X T) by Σ N, where N is the number of legal opcodes in a class and T the path length in cycles for that class. BYTNUM has the fastest index, 30.6 cycles (range 28 to 37). BYTCNT is slower, 32.5 cycles (range 29 to 39). CNTBYT is slower still, 39.3 cycles (range 23 to 42). The delay is caused by its having to sort out 3 small classes (containing only 11 of the 151 opcodes) before it can work on the 3 large ones. The lesson to be drawn from this is that linear-sequence logic is *not inherently slower* (as I implied in my previous *DDJ* notes) and might even be *faster* if its first stages could sift out the majority of the items being sorted. However, the range of sorting times is necessarily wider. One additional point: to make the 2 new routines fully equivalent to BYTNUM, one would have to add a PHA to store and a PLA to restore the accumulator, at the cost of 2 more bytes and 7 more cycles. However, the main program might be able to restore the accumulator more efficiently.

One interesting point about all these "decoding" routines is that, although their branch instructions subdivide opcodes into 6 classes, they return with 3 byte-count classes *re-encoded* (as 01, 02, and 03) in the X register, so that a little more decoding logic is required to *use* this information. A second point, that I have emphasized before but that deserves reiteration, is that they all produce *additional* information as an incidental effect. E.g., CNTBYT and BYTCNT cause *only* 1 byte opcodes to set the Z flag, i.e., this type is *truly* decoded. It would be easy to recode BYTNUM to have the same effect, by omitting its inner RTS and relocating its TRICK sequence outside the subroutine. This would save 1 byte and very slightly reduce its speed. CNTBYT returns with the carry flag always equal to bit 7 of the opcode, and the sign flag always cleared. BYTCNT sets the carry flag equal to bit 4 for all 1 byte opcodes; it clears the carry for opcodes of type (0,1) (0,7) and sets it for all others. The BIT instruction in BYTCNT and BYTNUM sets the V flag equal to bit 6 for all opcodes; in BYTNUM it also causes all 1-byte opcodes to return with the N flag equal to bit 7.

I still prefer and recommend BYTNUM, although in my KIM system it takes 32 bytes (that would be 31 in the minor revision suggested above). In a well designed system, every one of the 256 possible bit patterns would be present in ROM (for use as numerical constants, BIT masks, etc.) and BYTNUM could be coded in 30 (or 29) bytes, a small price to pay for non-destructive logic and higher speed. Unfortunately (as I showed in a recent note in *Kim User Notes*) the KIM-ROM omits 81 of the possible patterns, including 39 of the 151 legal opcodes.

In the letter in which Jim Butterfield sent me his CNTBYT program, he also pointed out that a table-look-up operation would optimize timing. One way to do this is to move the opcode into an index register to get the content of a unique location in a 256 byte table. A content of 00 would identify an illegal (setting the Z flag), while contents of 01, 02, or 03 would give the byte-count of a legal. A 7 byte program insert (XINFOA, cf. listing) would do the work of *both* BYTNUM and OPLEGL at a cost of (7+M) main program bytes and (10+N) cycles. The values of M and N depend on whether the opcode needs to be restored in the accumulator (if not, both are zero) and on the kind of coding the main program is using to pick up a sequence of opcode bytes (most likely the LDA (INDIRECT), Y instruction with M = 2 and N = 5). Since the existing (not optimized) decoding logic version of OPLEGL needs 66 subroutine bytes and has a speed index of 37 cycles (range 24 to 63), the XINFOA insert would run more than 4 times faster than the OPLEGL + BYTNUM combination. To anyone enamored of speed (as I am) this is very attractive. An added superiority is that the main program can decide what to do with an illegal. If a program BREAK is desired, the branch offset ILLEG can be directed to the nearest 00 operand, otherwise to an illegality handling routine. One of several flaws in OPLEGL is that it forces a

BREAK for illegals. A well designed subroutine should return information and *not* make decisions.

Nothing comes free, and the price of the XINFOA speed is a very long table, with every opcode a *unique* class, vulnerable to bugs if located in RAM. Ideally an info table like TAB256 should be in ROM, but it uses only the 2 low-order bits and one hesitates to waste 75% of the bits in an expensive ROM. The solution would be to encode *other* information in the high-order bits, and AND them out to retrieve the legality/bytecount data. The dilemma is: *what* other information?

I briefly explored another approach, fully packing the infobit pairs of TAB256 into a 64 byte table. Retrieval needs a 36 byte program, too long for a main progarm insert, and with a speed index of 63 cycles when made into a subroutine. This is not good enough to be worth presenting here.

A more promising use of info tabling is my new legality-testing subroutine, HYLEGL, that has many advantages when compared with OPLEGL. It needs only 64 bytes, saves and restores the accumulator, and returns with the carry flag set by illegals and cleared by legals. Its speed index is 44 cycles (range 36 to 71). A revision of OPLEGL to make it fully equal to HYLEGL would be about 10 bytes longer and 2 cycles slower.

HYLEGL is a "hybrid" operation that retains the decoding logic of OPLEGL for the 128 "odd-number" opcodes because this is both fast and efficient. For the 128 "evens" it encodes legality as single infobits packed in a 16 byte table, INFTAB, a 1 bit meaning illegal and a 0 bit legal. A table-addressing index from 0 to F is generated by ANDing bits 4321 of the opcode, bit 0 (always 0 for evens) having been right-shifted out. The 16 infobytes correspond to the original opcode sequence: $X_0$0, $X_0$2,. . . .X_0E, $X_1$0. . . .X_1E. Since 7 of the infobytes contain no 1 bits, legality is proved by the setting of the Z flag at the load into the accumulator, so no retrieval is needed. For the remaining 9, the correct infobit is moved into the carry flag by a sequence of from 1 to 8 ASLs, controlled by analyzing the status of bits 765 of the original opcode (stored in the temporary zero-page location COPY).

Since INFTAB contains 128 unique classes, it is as vulnerable to bugs as any other table in RAM. Since it is short, it is much easier to check and also more likely to get itself fitted into a ROM. Decoding logic in RAM is of course not immune to bugs, but when a large class is involved the bug will show up very quickly. In a unique class, rarely use-tested (such as the illegals individually weeded out by OPLEGL) a bit-error bug may lurk for a long time before inflicting damage. Worse, the damage may be neither suspected nor detected; the output will look right but be wrong, and may even create a *new* hidden bug in other programs.

HYLEGL highlights some "gray areas" of subroutine design, where programming philosophies differ. The X register is used in its table look-up segment, but the pre-call X content is saved and restored at a cost of 4 bytes and 6 cycles. Its operations destroy the pre-call content of the accumulator, but this is saved and restored at a cost of 3 bytes and 7 cycles. It can be argued that only the main program really knows what needs to be saved and how and where to save it and when to restore it. By omitting the saves, HYLEGL would need only 57 bytes and 34 cycles, but greater precaution would be needed in the writing of the main program (perhaps not an evil). Should subroutines try to leave no traces of their work, other than the intended ones? Despite its saves, HYLEGL *does* leave traces, in the zero-page locations STORX and COPY and as incidental effects on the N, Z, and V flags. However, the PLP and PHP instructions of the 650X make status saving especially easy for the main program, and low zero-page locations are commonly viewed as temporary scratchpad area. My current view is that subroutines should be written to save on-chip registers, since it is easy for a user to remove any save instructions he does not want.

In order to make the high speed of the XINFOA/TAB256 approach immediately available, I am adding to the package subroutine MKTABL, that uses HYLEGL and BYTNUM to construct a *temporary* TAB256 in a RAM page. MKTABL is long (52 bytes) because it has an unusually high content of antibugging code and also serves as a testing/debugging program for the legality/bytecount subroutines. A large table in RAM is not only insecure but hazardous to generate. The data generating routines may be yielding some incorrect values. The write operation may be overwriting a RAM page other than the intended one, or trying to write in nonexistent RAM (that in systems not fully address-decoded may prove to be existent RAM with valuable stuff in it) or in write-protected RAM or in ROM.

Before calling MKTABL, a main program is expected to write into location BASEHI the number of the RAM page into which the table will be written. MKTABL compares this to the unique page number in its own coding; only if they are the same will table writing be permitted. It also sets the low base address for the table to 00, to ensure that the table will lie within the bounds of the one RAM page. This is part of the initial zeroing of 5 adjacent zero-page locations, that will contain useful information when MKTABL exits. The lowest one, COPY, is used by HYLEGL and will contain $FF at a correct exit; each run of HYLEGL stores the opcode it has worked on, unaltered except for the 72 "evens" that require infobit retrieval (that causes one ASL of COPY).

The next higher location (LEGCNT) contains $97 at a correct exit, since this counter is incremented before each call to BYTNUM, of which there will be 151 if HYLEGL is working properly. The next location (CHKLO) is a 1 byte checksum that contains $41 at a correct exit. The next location (TABGEN) indirectly counts the number of table-writes, containing either 00 (if no table was written) or $FF. This is a bit trickier than a simple count. After each write, the current value of the opcode is moved from the Y index into the accumulator, destroying the infobyte content so that the subsequent reloading of the infobyte from the table location is a test that the correct value was indeed stored there. The actual content of TABGEN is the current opcode value, from Y, that if all is well is also the count of writes.

Finally, the content of BASELO should always be 00, and BASEHI should contain the number of the unique permissible RAM page if the main program commanded a table-write, and some other number if not. A simple way to implement a no-table command would be an LDY #$06 followed by a JSR SETEX to bypass the first MKTABL instruction, since SETEX would zero BASEHI.

The terminal instructions of MKTABL control the 2 flags that will tell the main program what has happened. The BIT sets the V flag if a table has been written (TABGEN = $FF), and clears it if not. The crucial instructions compare the checksum in CHKLO with the correct value $41, setting the Z flag if all is well, clearing it if not. A great variety of errors, either in MKTABL itself or in the subroutines it calls, can cause a wrong checksum. Much information on the kind of error can be derived from the content of the 6 zero-page locations, of the accumulator (should be $41) and of the X and Y registers (both should be 00). If a table was written, very detailed information on operation will be found in the infobyte sequence.

The antibugging logic in MKTABL, although far from a total bugproofing, is clearly costly in memory and time. Programs that aim to optimize timing, such as the XINFOA/

TAB256 combination, can tolerate little or none. (An example of very light antibugging would be addition of an AND #$03, at a cost of 2 bytes and 2 cycles, to erase the 6 high-order bits of the infobyte.) The major insurace against error for such programs is a thorough testing by an antibug test program, both before and after each use. In service programs resident in RAM, like MKTABL, error-proofing can be given priority. It may be desirable to trade off more memory to attain higher speed, but one should never trade off reliability! On this subject, the chapters on antibugging, testing, and debugging in E. Yourdon's *Techniques of Program Structure and Design* (Prentice-Hall, 1975) make interesting reading.

To sum up, tables can indeed play useful roles beyond their classic one of defining arbitrary relations and rules (such as the metalogical ASCII decision that $41 represents the letter A). These other roles have been obscured by the fact that it is simpler to implement a *de novo* calculation of a math function than a look-up in a gigantic table. With tables of more modest size, table look-up can (as Jim Butterfield suggested) maximize speed, at an acceptable (though relatively high) memory cost. When decoding logic proves to be very intricate (many small or unique classes) the use of tables may also be more code-efficient. Furthermore, programs may be easier to write because similar retrieval logic may be applicable to a variety of tables. There is probably some trade-off point (more that 35 bytes of decoding logic, more than 10 classes decoded?) where the entabling of pre-decoded information becomes an optimal solution of the dimensions of the table can be kept within reasonable bounds (no more than 32 bytes?).

(coding for CNTBYT and BYTCNT)

```
Ø23Ø  A2 Ø1     CNTBYT LDX #$Ø1 (initialize X)
   2  49 2Ø            EOR #$2Ø (flip bit 5)
   4  FØ 13            BEQ THREE (only $2Ø)
   6  29 9F            AND #$9F (bits 5,6 out)
   8  FØ 11            BEQ ONE  (only (Ø,4,6)Ø)
   A  ØA               ASL A  (shift bit 7 out)
   B  49 12            EOR #$12 (flip bits 3,Ø)*
   D  FØ ØB            BEQ TWO  (only X09)
   F  29 1A            AND #$1A (bits 4,1 out)*
Ø241  49 Ø2            EOR #$Ø2 (flip bit Ø)*
   3  FØ Ø6            BEQ ONE  (X(8,A))
   5  29 1Ø            AND #$1Ø (all but flipped
                                 bit 3 out)*
   7  DØ Ø1            BNE TWO (resid. X(Ø-7))
   9  E8       THREE   INX      (resid. X(9-F))
   A  E8        TWO    INX
Ø24B  6Ø        ONE    RTS

 * comment refers to original opcode bits
   before left-shift at Ø23A

Ø23Ø  A2 Ø1     BYTCNT LDX #$Ø1
   2  2C 36 Ø2         BIT BYTCNT+6 (test bit 3)
   5  DØ Ø8            BNE HALFOP (all X(8-F))
   7  C9 2Ø            CMP #$2Ø
   9  FØ ØE            BEQ THREE (only $2Ø)
   B  29 9F            AND #$9F  (bits 5,6 out)
   D  DØ ØB            BNE TWO (all except
                                (Ø,4,6)Ø)
   F  29 15    HALFOP  AND #$15 (retains only
                                 bits Ø,2,4)
Ø241  C9 Ø1            CMP #$Ø1
   3  FØ Ø5            BEQ TWO  (X0(9,B))
   5  29 Ø5            AND #$Ø5 (bit 4 out)
   7  FØ Ø2            BEQ ONE  (X(8,A) and
                                (Ø,4,6)Ø)
   9  E8         THREE INX  (resid. X(9-F))
   A  E8          TWO   INX
Ø24B  6Ø          ONE   RTS
```

```
              XINFOA TAX
                     LDA TAB256, X
                     BEQ ILLEG
                     TAX

Ø26E  48       HYLEGL PHA (save opcode in stack)
   F  85 1Ø           STA COPY (store work-copy)
Ø271  4A              LSR A  (bit Ø to carry)
   2  9Ø ØA           BCC TABLOP (do evens/table)
   4  4A              LSR A  (bit 1 to carry)
   5  BØ Ø5           BCS ILLEX (all X(3,7,B,F))
   7  C9 22           CMP #$22  (LSRed $89)
   9  FØ Ø1           BEQ ILLEX  ($89 illegal)
   B  18              CLC  (legals clear carry)
   C  68        ILLEX PLA  (restore opcode in A)
   D  6Ø              RTS
   E  86 ØF    TABLOP STX STORX (save X register)
Ø28Ø  29 ØF           AND #$ØF (bits 7,6,5 out)
   2  AA              TAX  (4321 to X index)
   3  BD 9E Ø2        LDA INFTAB,X (load infobyte)
   6  FØ 12           BEQ EXIT (all Ø bits, legal)
   8  Ø6 1Ø           ASL COPY (bit 7 to carry)
   A  BØ Ø4           BCS STEPA (bit 7 = 1)
   C  ØA              ASL A (leftshift infobyte)
   D  ØA              ASL A
   E  ØA              ASL A
   F  ØA              ASL A
Ø29Ø  24 1Ø     STEPA BIT COPY (bits 6,5 to N,V)
   2  3Ø Ø2           BMI STEPB (bit 6 = 1)
   4  ØA              ASL A
   5  ØA              ASL A
   6  7Ø Ø1     STEPB BVS STEPC (bit 5 = 1)
   8  ØA              ASL A
   9  ØA        STEPC ASL A  (infobit to carry)
   A  A6 ØF     EXIT  LDX STORX (restore X)
   C  68              PLA  (restore opcode)
Ø29D  6Ø              RTS
```

INFTAB table for even opcodes

```
Ø29E  1Ø DF ØD ØØ ØØ ØØ Ø1 ØØ
Ø2A6  ØØ FF CF ØØ ØØ CF DF 1Ø
```

(coding for MKTABL)

```
Ø2AF  AØ Ø5     MKTABL LDY #$Ø5 (zero 5 loci)
Ø2B1  A2 ØØ      SETEX LDX #$ØØ (notable entry)
   3  96 ØF     ZEROIT STX COPY-1, Y
   5  88               DEY
   6  DØ FB           BNE ZEROIT
   8  D8               CLD  (clear decimal mode)
   9  98        OPBYTE TYA  (opcode from Y to A)
   A  2Ø 6E Ø2         JSR HYLEGL (test legality)
   D  A2 ØØ            LDX #$ØØ  (rezeroes X)
   F  BØ Ø5            BCS MOVXA (illegal bypass)
Ø2C1  E6 11            INC LEGCNT (count legals)
   3  2Ø 1Ø Ø2         JSR BYTNUM (bytecount to X)
   6  8A        MOVXA  TXA (bytecount from X to A)
   7  A2 XX            LDX #$XX (RAM table page #)
   9  E4 15            CPX BASEHI (compare page #
                                   set by program)
   B  DØ Ø7            BNE NOTABL (nowrite if ≠)
   D  91 14            STA (BASELO),Y  (write)
   F  98               TYA  (opcode from Y to A)
Ø2DØ  85 13            STA TABGEN (count writes)
   2  B1 14            LDA (BASELO),Y (read table)
   4  18        NOTABL CLC  (clear carry for add)
   5  65 12            ADC CHKLO (bytecount sum)
   7  85 12            STA CHKLO (store checksum)
   9  C8               INY   (next opcode)
   A  DØ DD            BNE OPBYTE
   C  24 13     EXIT   BIT TABGEN (sets V if table)
   E  A9 41            LDA #$41 (valid checksum)
Ø2EØ  C5 12            CMP CHKLO (compare actual)
Ø2E2  6Ø               RTS
```

BASIC SEARCHER/PATCHER

Dear Dr. Dobb's: *Received: 77 Dec 9*

The enclosed program should be of use to anyone seeking to modify MITS 4.0 or 4.1 BASIC. It should be used with an 8080 disassembler such as the one published in the March 1977 issue of *Dr. Dobb's Journal*.

The program has two functions; viz., searching and patching. The code to be searched or patched may reside anywhere in memory or on the reserved areas of the disk. These areas comprise the first five tracks and are occupied by the bootstrap/loader (sectors # 0,2,4,6) and by BASIC itself (remaining 156 sectors).

The search function accepts any non-empty string of characters or hexadecimal digits. A search is then performed over the specified address range. Wherever a matching string is found, the hex address is displayed or printed and the search continued.

Two searching examples are given. In the first, the memory resident BASIC code is searched for all occurrences of the hex string 'DB08' (read disk status command). These indicate where the disk I/O routines are located. In the second example, a search is made through part of the BASIC code on the disk for the string "RANDOM". This was found only once in the specified address range.

The patch function accepts a starting address, which is then displayed together with the hexadecimal, decimal, and ASCII equivalents of its contents. If the byte is to be changed then a single character or two hex digits are entered. Otherwise only a carriage return is entered. The patching address is then incremented, decremented, or left unchanged, as selected.

Two patching examples are shown. In the first, the string "RANDOM" beginning at address '069D' is changed to "RANCID". In the second, the command 'DB08' (IN 08) at '2E86' is changed to D309' (OUT 09). Since these changes were made only to memory, they are not permanent.

James Monagan
806 Clark St.
Iowa City, IA 52440

```
10 PRINT"'PATCHER'"  '12/05/77
20 PRINT"PATCHES ANY BYTE IN MEMORY OR ON THE DISK COPIES OF MITS' 4.0 B
ASIC AND  BASIC BOOTSTRAP/LOADER."
30 PRINT"ALSO SEARCHES THROUGH ANY OF THE ABOVE TO LOCATE A GIVEN ASCII
OR BINARY STRING."
40 CLEAR500
50 TRK=-1
60 '
70 PRINT:INPUT"1) SEARCH   2) PATCH";FC:IFFC<10RFC>2THEN70
80 PRINT:MEM=1:INPUT"1) MEMORY   2) BOOTSTRAP   3) BASIC";MEM:IFMEM<10RM
EM>3THEN80
90 IFFC=2THEN 980
100 '
110 'SEARCH ROUTINES
120 PRINT:INPUT"DO YOU WANT HARD COPY";ZZ$:PRINT
130 IF LEFT$(ZZ$,1)="Y" THEN HDCPY=1ELSEHDCPY=0
140 '
150 GOSUB400  ' GET START ADDRESS
160 PC=X
170 PROMT$="END":GOSUB420  ' GET END ADDRESS
180 HLTADR=X
190 GOSUB490  ' CHECK IT
200 '
210 PRINT:I=3:INPUT"1) CHARACTER-  2) HEX-STRING SEARCH   3) EXIT";I:IFI
=3THEN70
220 LINEINPUT"SEARCH STRING = ";SRCH$:IFHDCPYTHENGOSUB65520
230 PRINT:PRINT"SEARCH STRING = ";:IFI=2THENPRINT"HEX: ";
240 PRINT"'"SRCH$"'"
250 PRINT"ADDRESS"
260 IFI=2THENGOSUB870  'CONVERT STRING IF HEX
270 SLG=LEN(SRCH$):IFSLG=0THEN GOSUB65525:GOTO210   'INVALID STRING?
280 BC%=1:SS=ASC(SRCH$)  '1ST BYTE 1ST
290 '
300 'SEARCH FOR STRING
310 IFMEM=1THENX=PEEK(PC)ELSEGOSUB690  ' FETCH NEXT BYTE
320 PC=PC+1:IFX<>SSTHEN350  'MATCH ?
330 IFBC%<SLGTHEN BC%=BC%+1:SS=ASC(MID$(SRCH$,BC%)):GOTO360 '  COMPARE F
OLLOWING BYTE, IF ANY
340 X=PC-SLG:GOSUB610:PRINT '  FOUND, PRINT ADDRESS
350 IFBC%>1THENPC=PC-BC%+1:BC%=1:SS=ASC(SRCH$)
360 IFPC<HLTADRTHEN310
370 GOSUB65525
380 PRINT"HALT ADDRESS REACHED!":GOTO70
390 '
400 'GET HEX ADDRESS
410 PROMT$="START"
420 SRCH$="":PRINTPROMPT$; :INPUT" ADDRESS (HEX) = ";SRCH$
430 IFLEN(SRCH$)<>4THEN420
440 GOSUB870   'CONVERT TO BINARY
450 IFLEN(SRCH$)<>2THEN420
460 X=256*ASC(SRCH$)+ASC(MID$(SRCH$,2))
470 RETURN
480 '
490 'CHECK HALT ADDRESS
500 IF MEM=1AND HLTADR>65526! THEN HLTADR=65526!
510 IF MEM=2 AND HLTADR>512 THEN HLTADR=512
520 IF MEM=3 AND HLTADR>23552 THEN HLTADR=23552
530 RETURN
540 '
550 ' FETCH THE NEXT BYTE
560 IFMEM=1THENX=PEEK(PC)ELSEGOSUB690
570 RETURN
580 '
590 ' PRINT X AS A 2-DIGIT HEX NUMBER
600 PRINT RIGHT$("00"+HEX$(X),2);:RETURN
610 ' PRINT X AS A 4-DIGIT HEX NUMBER
620 PRINTRIGHT$("0000"+HEX$(X),4);:RETURN
630 '
640 'PRINT HEX, DECIMAL, & ASCII STRINGS
650 PRINTTAB(10):GOSUB590:PRINT"   ";RIGHT$("000"+MID$(STR$(X),2),3);"
";
660 IFX>32ANDX<91THENPRINTCHR$(X);
670 RETURN
680 '
690 'FETCH BYTE FROM DISK BUFFER
700 IFMEM=2THEN XPC=PCELSE XPC=PC+512   'BOOTSTRAP'S ON 1ST 4 SECTORS
710 ADR=4+(XPCAND127):TK=INT(XPC/4096):SC=2*((XPC/128)AND31):IFSC>30THEN
SC=SC-31
720 '
730 'GET TRACK IF NECESSARY
740 IFTRK>=0THEN770
750 TRK=0:SEC%=-1
760 IF(INP(8)AND64)<>0THENWAIT8,2,2:OUT9,2:GOTO760 '   GET TRACK #0
770 IFTRK=TKTHEN810
780 WAIT8,2,2:IFTRK>TKTHENOUT9,2:TRK=TRK-1ELSEOUT9,1:TRK=TRK+1   'MOVE H
EAD TOWARDS DESIRED TRACK
790 SEC%=-1:GOTO770
800 'GET SECTOR IF NECESSARY
810 IFSEC%=SCTHEN840ELSESEC%=SC:ECNT=10
820 SEC$=DSKI$(SEC%):TST$=DSKI$(SEC%)   'READ SECTOR
830 IFTST$<>SEC$THENECNT=ECNT-1:IFECNT>0THEN820 ELSE GOSUB65525:PRINT"DI
SK READ ERROR":STOP
840 X=ASC(MID$(SEC$,ADR))
850 RETURN
860 '
870 ' CONVERT HEX STRING TO BINARY STRING
880 J=-1:S$="":L=LEN(SRCH$)
890 FORI=1TOL
900 C$=MID$(SRCH$,I,1):IFC$=" "THEN950
910 IFC$>="0"ANDC$<="9"THEND=ASC(C$)-ASC("0"):GOTO940
920 IFC$>="A"ANDC$<="F"THEND=10+ASC(C$)-ASC("A"):GOTO940
930 S$="":I=L:GOTO950
940 J=(J+1)AND1:IFJ=0THENT=DELSET=D+16*T:S$=S$+CHR$(T)
950 NEXT
960 SRCH$=S$:RETURN
970 '
980 '   PATCH ROUTINES
990 GOSUB400:PC=X   ' GET ADDRESS
1000 PRINT:PRINT"X OR 'X' = ASCII,  XX = HEX, XXX = DECIMAL, RETURN = NO
 CHANGE":PRINT
1010 IF(MEM=2ANDPC>511)OR(MEM=3ANDPC>23551)THEN 1080
1020 CHNG=0:ADR=0:GOSUB560:SX=X ' FETCH BYTE
1030 X=PC:GOSUB610:PRINT";  ";   ' PRINT ADDRESS
1040 X=SX:GOSUB650:PRINTTAB(25)  'PRINT OLD VALUE
1050 SRCH$="":INPUTSRCH$:L=LEN(SRCH$)   'PRINT NEW VALUE
1060 IFL=0THEN 1140  ' JUMP IF NO CHANGE
1070 IFL=3ANDASC(SRCH$)=39THENSRCH$=MID$(SRCH$,2,1)  ' STRIP OFF QUOTES
1080 IFL=3THENIFSRCH$>="000"ANDSRCH$<="255"THENSRCH$=CHR$(VAL(SRCH$))  '
CONVERT DECIMAL
1090 IFL=2THENGOSUB870  ' CONVERT HEX
1100 IFLEN(SRCH$)<>1THENPRINT CHR$(7)"1 BYTE AT A TIME":GOTO1030
1110 '
1120 'MAKE PATCH
1130 SX=ASC(SRCH$):IFMEM=1THENPOKEPC,SXELSEMID$(SEC$,ADR,1)=SRCH$:CHNG=1
1140 FA=1:INPUT"PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT";FA
:IFFA<10RFA>4THEN1140
1150 IFFA=2THEN1030
1160 IFFA=4THENGOSUB1220:GOTO70
1170 IFFA=1THENPC=PC+1:ADR=ADR+1ELSEPC=PC-1:ADR=ADR-1
1180 IFMEM=1THEN1010
1190 IFADR>=4ANDADR<=131THENGOSUB840:SX=X:GOTO1030
1200 GOSUB1220:GOTO1010
1210 '
1220 ' OUTPUT SECTOR IF CHANGES MADE
1230 IFMEM=1ORCHNG=0THENRETURN
1240 CS=0:FORADR=4TO131:CS=(CS+ASC(MID$(SEC$,ADR)))AND255:NEXT
1250 MID$(SEC$,133,1)=CHR$(CS)   'UPDATE CHECKSUM
1260 ECNT=10
1270 DSKO$ SEC$,SEC%:TST$=DSKI$(SEC%) 'WRITE SECTOR
1280 IFTST$<>SEC$THENECNT=ECNT-1:IFECNT>0THEN1270 ELSE GOSUB65525:PRINT"
DISK WRITE ERROR":STOP
1290 RETURN
65509 '
65510 'SUBSTITUTE CENTRONIC PRINTER'S PORT # FOR CRT'S
65520 IFTRM%=0ANDINP(33)=0THENTRM%=PEEK(4561):OUT32,0:OUT34,0:OUT33,0:OU
T35,255:OUT32,44:OUT34,44:POKE4561,33:POKE4562,246:POKE4563,0:POKE4564,1
94:POKE4569,35
65521 RETURN
65522 '
65523 'RESTORE THE CRT AS OUTPUT PORT
65525 IFTRM%>0THENPOKE4561,TRM%:POKE4562,230:POKE4563,2:POKE4564,202:POK
E4569,TRM%+1:OUTTRM%,3:OUTTPM%,17:OUTTRM%,1
65526 TRM%=0:RETURN
OK
```

FEBRUARY SHOW IN PHILLY

News Release *Received:77 Dec 28*

1978 Business and Personal Computer Sale & Exposition

Location: City Line Holiday Inn, Philadelphia, Pa.
Date: February 27th to March 1st, 1978
Hours: 10:00 A.M. to 8:00 P.M.

For further information contact: Produx 2000, Roosevelt Blvd and Mascher St, Philadelphia, PA 19120 (215) 722-4837

GERMAN COMPUTER CLUB HOPES

Received: 77 Oct 10

"Call for interested individuals in (northern) Germany to form a Hobbyist Computer Club. Please contact Ilja Burkoff, Parkallee 105, D-28 Bremen. Tel: (0421) 34 71 79."

```
 'PATCHER'
PATCHES ANY BYTE IN MEMORY OR ON THE DISK COPIES OF MITS' 4.0 BASIC AND
 BASIC BOOTSTRAP/LOADER.
ALSO SEARCHES THROUGH ANY OF THE ABOVE TO LOCATE A GIVEN ASCII OR BINARY
 STRING.

1) SEARCH   2) PATCH? 1

1) MEMORY   2) BOOTSTRAP   3) BASIC? 1

DO YOU WANT HARD COPY? N

START ADDRESS (HEX) = ? 0000
END ADDRESS (HEX) = ? 5C00

1) CHARACTER-  2) HEX-STRING SEARCH   3) EXIT? 2
SEARCH STRING = DB08

SEARCH STRING = HEX: 'DB08'
ADDRESS
2E86
2E96
2EFD
2F08
303D
3057
3B52
3B69
3B8F
3BC7
HALT ADDRESS REACHED!

1) SEARCH   2) PATCH? 1

1) MEMORY   2) BOOTSTRAP   3) BASIC? 3

DO YOU WANT HARD COPY? N

START ADDRESS (HEX) = ? 0000
END ADDRESS (HEX) = ? 1000

1) CHARACTER-  2) HEX-STRING SEARCH   3) EXIT? 1
SEARCH STRING = RANDOM

SEARCH STRING = 'RANDOM'
ADDRESS
069D
HALT ADDRESS REACHED!

1) SEARCH   2) PATCH? 2

1) MEMORY   2) BOOTSTRAP   3) BASIC? 1
START ADDRESS (HEX) = ? 069D

X OR 'X' = ASCII,  XX = HEX,  XXX = DECIMAL,  RETURN = NO CHANGE

069D:      52    082   R    ?
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
069E:      41    065   A    ?
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
069F:      4E    078   N    ?
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
06A0:      44    068   D    ? C
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
06A1:      4F    079   O    ? I
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
06A2:      4D    077   M    ? D
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT? 4

1) SEARCH   2) PATCH? 2

1) MEMORY   2) BOOTSTRAP   3) BASIC? 1
START ADDRESS (HEX) = ? 2E86

X OR 'X' = ASCII,  XX = HEX,  XXX = DECIMAL,  RETURN = NO CHANGE

2E86:      DB    219         ? D3
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT?
2E87:      08    008         ? 09
PATCH 1) NEXT   2) THIS   3) PREVIOUS BYTE   4) EXIT? 4
```

A Simple Technique for Static Relocation of Absolute Machine Code

BY GARY A KILDALL
Digital Research
Pacific Grove, CA 93950

One principal difficulty with newly evolving computer technology is that software generation tools generally lag corresponding hardware facilities, thus forcing the software engineer to resort to outmoded techniques to produce software systems.

The purpose here is to present one area of difficulty—that of a static program relocation—and to offer a simple solution which can be applied to nearly any microcomputer software environment where relocation is not supported by the manufacturer.

The need for static relocation arises most often in a situation where the software systems must be reconfigured in the field. For example, data entry equipment manufacturers often provide a range of optional peripherals which can be attached to user's equipment as processing requirements change. Each peripheral usually requires a software "driver" which is device-specific, and interfaces the device to the operation environment.

A common approach to software reconfiguration is to arrange the individual translated peripheral drivers into distinct machine code modules which can be selectively brought together to form an integral system at the customer site. In order to perform the field reconfiguration, each module is translated so that it originates at location 0 in memory and, when it is brought together with other modules, it is placed at the next available memory location as the system is being constructed. All machine code elements which are location dependent must, of course, be altered to reflect the actual locations that the driver occupies. Generally, the elements which are affected are the addresses of branch destinations and data addresses. If the locations of the affected addresses in each module are known ahead of the system reconfiguration, the module can be placed anywhere in the final memory image.

Simple Static Relocation

The process of constructing an executable memory image from a set of relocatable modules, as described above, is called static relocation. Unfortunately, very few microcomputer manufacturers produce the address information with their translator output which is required for the relocation process. The method described below, however, can be applied to the output of most manufacturers' absolute translators to derive the necessary relocation information. In order to be specific, the Intel 8080 microcomputer is used in the discussion with the understanding that the concepts can be easily extended to differing architectures.

The Intel 8080 microcomputer has a 64K (65536 bytes) memory space which can be thought of as 256 "pages" of 256 bytes per page. Data and instructions are intermixed in this memory space, and are addressed with a 16-bit address operand which can be divided into an 8-bit (high-order) page address (0-255), and an 8-bit (low-order) address within a page. Typical 8080 instructions which can use these address operands are shown in Figure 1, where PA denotes the page address, and AWP denotes the address within a page. In general, a machine code memory image consists of instructions, instruction addresses, and data items. The instructions and data items are independent of the actual location at which the module finally resides. Further, only a subset of the instruction addresses are dependent upon the module location. That is, a load instruction may reference a buffer address which is fixed outside the relocatable module, in which case it does not change when the module is moved into position. If the address references a branch location or data item within the module, then the value of PA, AWP, or both, must be biased by fixed values, dependent upon the final position of the module in the resulting configuration.

MVI	A,	PA		Move immediate to A
MVI	C,	AWP		Move immediate to C
LXI	D,	AWP	PA	Load DE with address
JMP		AWP	PA	Jump to address

Figure 1. Typical 8Ø8Ø Instructions

A simpler form of relocation, called "page boundary relocation," is usually sufficient for field reconfiguration. In this case, the module is relocated to a page boundary so that only the page address (PA) need be changed to perform the relocation, since the address within a page (AWP) remains constant.

Page Boundary Relocation

In its simplest form, page boundary relocation can be accomplished by constructing two parallel memory images for each module. The first, called the "relative-0" image is created by translating the module for execution at location 0. The second, called the "relative-1" image is produced by translating the module for execution at page 1 (address 256). The relative-0 and relative-1 memory images can then be compared to determine the high-order address elements which must change when the module is moved to its final page boundary location. Figures 2a and 2b show a simple program segment assembled as relative-0 and relative-1 images. The differences in the machine code images are circled, and are thus the high-order addresses which must be biased when the module is moved. Figure 2c shows the same program segment assembled at page 5. Note that if the circled address fields in the ralative-0 image are biased by an amount 5 (corresponding to page boundary 5), they result in the proper values for the relocated program.

```
0000                    org     000h  ;relative 0 assembly
0000 3E00       start : mvi     a,d1 shr 8  ;page address to a
0002 0E0A               mvi     c,d1 and 0ffh ;address in page
0004 110A00             lxi     d,d1  ;full address to d, e
0007 C30000             jmp     start
                ;       data area
000A            d1:     ds      2     ;two unfilled
000C 00                 db      0     ;one filled element
000D                    end
```

Relative-0 Hex File:

```
:0A0000003E000E0A110A00C30000C2
:01000C0000F3
:00000000
```

Figure 2a. Relative-0 Assembly

```
0100                    org     100h  ;relative 1 assembly
0100 3E01       start:  mvi     a,d1 shr 8  ;page address to a
0102 0E0A               mvi     c,d1 and 0ffh ;address in page
0104 110A01             lxi     d,d1  ;full address to d,e
0107 C30001             jmp     start
                ;       data area
010A            d1:     ds      2     ;two unfilled
010C 00                 db      0     ;one filled element
010D                    end
```

Relative-1 Hex File:

```
:0A0100003E010E0A110A01C30001BE
:01010C0000F2
:00000000
```

Figure 2b. Relative-1 Assembly

```
0500                    org     500h  ;assembly at page 5
0500 3E05       start:  mvi     a,d1 shr 8  ;page address to a
0502 0E0A               mvi     c,d1 and 0ffh ;address in page
0504 110A05             lxi     d,d1  ;full address to d,e
0507 C30005             jmp     start
                ;       data area
                d1:     ds      2     ;two unfilled
050C 00                 db      0     ;one filled element
050D                    end
```

```
:0A0500003E050E0A110A05C30005AE
:01050C0000EE
:00000000
```

Figure 2c. Assembly at Page 5

The program which actually performs the relocation process is a simple modification of an absolute loader. The translator output for an 8080 microcomputer is a "hex format" file, containing a sequence of absolute records which give a load address and byte values to be stored starting at the load address. The exact format of each record, shown in Figure 3, begins with a colon (:) followed immediately by a two digit record length (RL) and 4-digit load address (LA). The 2-digit record type (RT) is always zero for absolute records, and is followed by RL pairs of hexadecimal digits to be placed at LA through LA+RL–1 in memory. The record terminates with a pair of checksum digits: if the byte values (hexadecimal digit pairs) are summed, starting immediately after the colon, and, continuing through the end of the record, including the checksum byte, then the sum should be zero when computed with an 8-bit counter. The checksum byte is included as an error detection mechanism. The last record of a hex file is denoted by a record length of 00.

```
:nnaaaattd1d2. . .dncc
```

nn	–	record length 01-FF
aaaa	–	load address 0000-FFFF
tt	–	record type = 00
d1	–	data byte 1
d2	–	data byte 2
. . .		
dn	–	data byte nn
cc	–	checksum byte

Figure 3. Hex File Format

An absolute loader reads each record of the hex file, and loads the byte values at the load address specified by LA for the next RL bytes, as shown in the algorithm of Figure 4. The notation used in this algorithm is that of Knuth [Kn.], where each step is labeled with a step name (A1. . . A16), followed by a comment describing the action of the step. The action itself is a series of assignments of expressions to variables, and conditional control transfers. The algorithm begins at step A1, and scans for the beginning colon for each record. When found, the algorithm reads the record length and, if zero, terminates the load operation. If the recrod length is not zero, the load address is read followed by the record type (which should be zero). The algorithm then loops between steps A6 and A12, reading successive bytes to memory while computing the checksum. When the entire record has been loaded, the final checksum byte is added, which should result in a zero value. Upon completion of the algorithm of Figure 4, the entire hex file has been read and loaded to an absolute location in memory.

The algorithm of Figure 5 is a simple extension of the previous absolute loader, which reads two successive hex files. The first hex file is the relative-0 machine code, produced by translating the module for execution at location 0. The second hex file is the relative-1 machine code, resulting from the module translation when originated at location 256 (100 in hexadecimal). The first part of the algorithm, given by steps A1 through A16 is similar to that of Figure 4, except that the data is loaded to address LA+PG*256 which effectively moves the module to the page boundary given by PG rather than absolute address LA.

Note: nextchar reads the next ASCII character
nextbyte reads the next pair of digits
nextaddr reads the next pair of bytes
CS is the checksum accumulator (8-bits)
RL is the record length (8-bits)
LA is the load address (16-bits)
M[x] is memory location x (8-bits)

A1	[scan for :]	if nextchar ≠ ":" go to A1
A2	[set checksum]	CS : = 0
A3	[get length]	RL : = nextbyte
A4	[last record?]	if RL = 0 go to A16
A5	[set address]	LA : = nextaddr
A6	[set type]	RT : = nextbyte
A7	[load bytes]	if RL = 0 go to A13
A8	[get byte]	b : = nextbyte
A9	[store byte]	M [LA] : = b
A10	[checksum]	CS : = CS + b
A11	[next addr]	LA : = LA + 1
A12	[count length]	RL : = RL – 1, go to A7
A13	[checksum]	CS : = CS + nextbyte
A14	[total ok?]	if CS = 0 go to A1
A15	[check error]	halt, "checksum error"
A16	[normal end]	halt, "tape read ok"

Figure 4. Absolute Loader Algorithm

Upon reaching step A16, the module has been loaded into memory at page PG but is translated for execution at location 0 and thus would (most likely) execute improperly, since the high order branch and data addresses must be biased by an amount PG. Thus, steps R1 through R19 read the relative-1 hex file to determine the addresses which must change. These steps are similar to A1 through A16, except the input data is compared with memory for differences, rather than actually placed in memory. In step R5, the load address is read as before but, since the relative -1 machine code is biased by one page, the effective address must be reduced by 256 bytes. Step R9 compared the data loaded in the first phase with the data read in the second phase: if the data is the same, then the element is invariant in the relocation process. If the data differs, then it must have been due to the difference in the relative-0 and the relative-1 memory images. Further, this difference must be exactly 1 since differences occur only in the high-order address fields; otherwise an error occurs, and the module cannot be relocated. When a relocatable element is found, the original value loaded and relocated in phase 1 must be biased by an amount PG in step R11. Upon completion of the second phase, the algorithm halts at step R19 with the high order addresses altered by the proper amount in the relocated module. Note that the algorithm given in Figure 5, when applied to the relative-0 file of Figure 2a, followed by the relative-1 file of Figure 2b, produces the relocated machine code of Figure 2c, when page boundary PG=5 is used.

Note: nextchar, nextbyte, nextaddr, CS, RL, LA, and M are identical to Figure 4. PG is the page number where the relocated module will reside.

```
A1  [scan for :]        if nextchar ≠ ":" go to A1
A2  [set checksum]      CS  := 0
A3  [get length]        RL  := nextbyte
A4  [last record?]      if RL = 0 go to A16
A5  [set address]       LA  := nextaddr
A6  [set type]          RT  := nextbyte
A7  [load bytes]        if RL = 0 go to A13
A8  [get byte]          b   := nextbyte
A9  [store byte]        M[LA+PG*256] := b
A10 [checksum]          CS  := CS + b
A11 [next addr]         LA  := LA + 1
A12 [count length]      RL  := RL - 1, go to A7
A13 [checksum]          CS  := CS + nextbyte
A14 [total ok? ]        if CS = 0 go to A1
A15 [check error]       halt, "checksum error"
A16 [end rel-0]         go to R1

R1  [scan for :]        if nextchar ≠ ":" go to R1
R2  [set checksum]      CS  := 0
R3  [get length]        RL  := nextbyte
R4  [last record?]      if RL = 0 go to R19
R5  [set address]       LA  := nextaddr + 256 x (PG - 1)
R6  [set type]          RT  := nextbyte
R7  [record done?]      if RL = 0 go to R15
R8  [compare data]      b   := nextbyte
R9  [data same?]        if b = M[LA] go to R12
R10 [page diff 1?]      if b ≠ M[LA] + 1 go to R18
R11 [relocate]          M[LA] := M[LA] + PG
R12 [checksum]          CS  := CS + b
R13 [next address]      LA  := LA + 1
R14 [count length]      RL  := RL - 1, go to R7
R15 [checksum]          CS  := CS + nextbyte
R16 [total ok?]         if CS = 0 go to R1
R17 [check error]       halt, "checksum error"
R18 [reloc error]       halt, "relocation error"
R19 [end rel-1]         halt, "relocation done"
```

Figure 5. Relocating Loader Algorithm

The algorithm of Figure 5 can be easily translated to an appropriate assembly or high-level language program to perform this relocation process.

The processing of Figure 5 can be used to produce a more compact form of the relocatable module by building a "bit vector" which tabulates the addresses which require relocation, rather than actually performing the relocation process. That is, in step R11 the address LA must be biased by an amount PG for proper execution when the module originates at address PG*256. Thus, on the first pass, the data can be read to memory and, upon completion of the pass, a bit vector is constructed which has one bit position for each address within the module. Before starting step R1, the entire bit vector is zeroed to indicate that no addresses require relocation. As the second phase processing proceeds, each relocation address determined in step R11 can be "marked" by setting the corresponding position of the bit vector. Upon completion of the algorithm, the bit vector contains zeroes in the positions corresponding to addresses which are invariant over the relocation, and ones in the positions which require biasing by an amount PG. The entire relocatable module can then be saved for later static relocation.

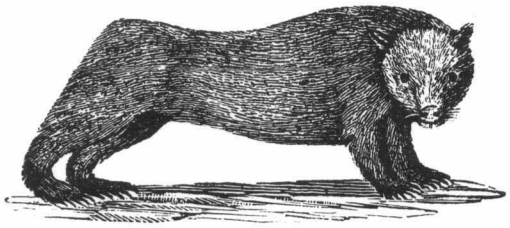
Given that the relative-0 memory image has been saved along with the relocation bit vector, the page boundary relocation can be simply accomplished by reading the memory image to its relocated page address PG. The bit vector is then read and processed: for each bit position which is set, the value PG must be added to the corresponding element which was previously loaded. Note that this extension to the basic algorithm of Figure 5 is included only for compact representation, and produces exactly the same memory image as the original algorithm.

A Case In Point

The following situation shows a case where page boundary relocation is useful. The CP/M operating system [Ki] is a simple small computer diskette based software system, which implements a file management and program loading facility for microcomputer development. The operating system is arranged as a set of modules which are loaded into memory when the computer system is started. User programs are then loaded into memory from the diskette and, because of memory constraints, must overlay non-essential portions of the CP/M system to reclaim storage for program and data areas. In order to allow these areas of memory to be reclaimed, the CP/M system is loaded into the high addresses of the memory space, and the user programs are loaded into the low addresses. Thus, the user programs can overlay the high addresses of memory when necessary and, upon completion, cause the CP/M system to be brought back from the diskette for the next operation.

Given that relocation is not supported by the manufacturer, this memory organization presents a fundamental difficulty: each CP/M operating system must be tied directly to the memory size. If the user of CP/M owns a computer system with 16K bytes of memory then a 16K version of CP/M must be supplied. If the user adds memory to enhance system capabilities, a different version of CP/M must be supplied to support the larger memory space.

In order to overcome this difficulty, the CP/M system can be reconfigured in the field to accomodate the increased memory using the page boundary relocation technique described above. In particular, each user receives a 16K version of CP/M (the smallest amount of memory which is useful for CP/M operation), along with a program which implements the reconfiguration. The user may optionally execute the program which rebuilds the CP/M system, according to the existing memory size, and places the relocated memory image back on the diskette, ready for subsequent loading.



The CP/M debugger program, called the Dynamic Debugging Tool (DDT), also resides in the upper portions of memory so that it can co-exist with the programs under test. Again, the area in which DDT is loaded depends upon the current memory configuration, and thus page boundary relocation is performed each time the DDT program is brought into memory. The increased elapsed time for relocation of DDT is negligible when compared to an absolute load, as long as the bit vector technique of the previous section is used.

Restrictions

It should be noted that the technique described here is by no means a complete linking loader: no address resolution is provided between modules, and no load-time address arithmetic is allowed. Sets of modules which co-exist in an integral system must communicate through instruction and data addresses. Using the technique presented here, the communication must be performed through dedicated absolute addresses for data items. Further, instruction addresses must be established through a "root module" which contains a jump vector with vector elements for each possible module which could be configured in a final system.

Address arithmetic is often useful when combining modules. In the simple page boundary relocation described above, all address arithmetic must be performed at assembly or compile time, and must consist only of simple operations which involve a fixed positive or negative offset from a base address, or a shift or logical and operation which extracts the 8-bit page address of a full 16-bit address. A relocation error will occur, for example, if an 8-bit immediate operand instruction is obtained from a 7-bit right shift rather than an 8-bit right shift of an address quantity.

In spite of these shortcomings, the technique has particular advantages in being independent of a manufacturer's capabilities, whims, and fancies. All language processors must eventually produce an absolute memory image for execution on the target machine, and thus the relocation process presented here will continue to operate when new software tools are introduced.

Acknowledgements

The author would like to point out that the techniques presented here, although useful, are most likely not original or particularly inventive. In fact, at least one individual, Bruce VanNatta of IMS Associates, San Leandro, California, has independently applied the methods to produce relocatable PL/M programs. There are most certainly many other software designers who have approached the relocation problem in a similar fashion.

References

Kildall, G., *Microcomputer Software Design: A Checkpoint*. Proceedings of the Fall Joint Computer Conference, 1975.

Knuth, D., *The Art of Computer Programming. Volume I: Fundamental Algorithms*. Addison-Wesley, Reading, MA, 1975.

FASTER DATA ENTRY FOR SC/MP

Dear Dr. Dobbs, *Received: 78 Jan 16*

Using the SC/MP kit (actually it is called Introkit this side of the Big Lake) with the keyboard addition I often cursed the program entry method as being somewhat slow. You have to press MEM, TERM, then enter your data, and finally press TERM again. That makes 5 keystrokes per byte where most systems only use 3, i.e. two for the hex digits and one carriage return. The enclosed program should solve this problem. All should be self-explanatory except for a few details:

1. The initial data display is always 00, the following (after first TERM) shows true data read from memory.
2. Assembler syntax is not standard but from a home-brewed 'cross' on an HP 9825A. The main difference is in hex notation – I use xxH where National uses X' or leading zero.
3. The program is relocatable as only PC-relative jumps are used (besides using SCMPKB which is assumed to start in 0000).

Best of all,

Erik Skovgaard
Nordlundsvej 10
DK-2650 Hvidovre
DENMARK

```
***** FSTKEY SCMPKB FAST DATA ENTRY ROUTINE *****

$$$$      Track 1  File #  6

   1                          ;ROUTINE FOR FAST ENTRY OF
   2                          ;PROGRAMS USING SCMPKB SYSTEM
   3                          ;JUST KEY IN BYTE AND PRESS TERM
   4                          ;ADDRESS IS THEN AUTO INCREMENTED
   5                          ;AFTER STORING THE BYTE.
   6                          ;ABORT KEY RETURNS CONTROL TO SCMPKB
   7                          ;PROGRAM ENTRY ADDR IS GIVEN IN LOC
   8                          ;0F15 (UPPER) AND 0F16 (LOWER)
   9                          ;
  10 0F30 C40F           LDI 0FH         ;P2=0F00
  11 0F32 36             XPAH P2
  12 0F33 C400           LDI 0
  13 0F35 32             XPAL P2
  14 0F36 C215           LD 15H(2)       ;GET ADDRESS
  15 0F38 CA0E           ST 0EH(2)
  16 0F3A C216           LD 16H(2)
  17 0F3C CA0C           ST 0CH(2)
  18 0F3E C400           LDI 0           ;CLEAR DATA
  19 0F40 CA0D           ST 0DH(2)
  20 0F42 900B           JMP "GODIS"
  21 0F44 C20D     NUM:  LD 0DH(2)
  22 0F46 1E             RR              ;SHIFT DIGIT
  23 0F47 1E             RR
  24 0F48 1E             RR
  25 0F49 1E             RR
  26 0F4A D4F0           ANI F0H         ;AND APPEND
  27 0F4C 58             ORE             ;NEW DIGIT
  28 0F4D CA0D           ST 0DH(2)       ;SAVE RESULT
  29 0F4F C401   GODIS:  LDI 1
  30 0F51 37             XPAH P3         ;P3=DISP ROUTINE
  31 0F52 C43F           LDI 3FH
  32 0F54 33             XPAL P3
  33 0F55 3F             XPPC P3         ;DISPLAY
  34 0F56 9002           JMP "CMD"       ;COMMAND RETURN
  35 0F58 90EA           JMP "NUM"       ;NUMBER RETURN
  36                          ;
  37                          ;COMMAND PROCESSING
  38                          ;ANY CMD EXCEPT ABT IS OK.
  39                          ;
  40 0F5A C20E     CMD:  LD 0EH(2)       ;P3=NEW ADDR
  41 0F5C 37             XPAH P3
  42 0F5D C20C           LD 0CH(2)
  43 0F5F 33             XPAL P3
  44 0F60 C20D           LD 0DH(2)
  45 0F62 CF01           ST @1(3)        ;STORE BYTE, INCR
  46 0F64 C300           LD 0(3)         ;READ NEXT BYTE
  47 0F66 CA0D           ST 0DH(2)
  48 0F68 37             XPAH P3         ;STORE NEW ADDR
  49 0F69 CA0E           ST 0EH(2)
  50 0F6B 33             XPAL P3
  51 0F6C CA0C           ST 0CH(2)
  52 0F6E 90DF           JMP "GODIS"     ;DISP NEW VALUES


 NUM = 0F44
GODIS = 0F4F
 CMD = 0F5A
```

Relocation Is Not Just Moving Programs

BY RICHARD L. ROTH
5 North Salem Rd.
Ridgefield, CA 06877
(203) 438-3954
© RLR 1977

Once you start writing assembly language programs you will find every program requires certain functions. Some functions are so common, such as getting a character from the console terminal, that you can build them into a system monitor. This monitor can be put in PROM or loaded when you turn your machine on. But the large majority of common functions are only used for a class of programs. It's wasteful to always have them around so you only have them in memory when you need them. During the stage of small programs and hand assembling, each time you want any of the common functions you must move it to the appropriate place and you must 'relocate' all absolute addresses, usually jumps and data. But as programs get bigger and as the number and kind of common modules grow, it behooves one to let the computer do the busy work of moving those modules. At the low level, if you separate your code from data, you might find a relocator that can move absolute code–fine for a 10 or 256 byte program. Not so fine for 4000 bytes, esp., since you must watch carefully to see if all the constants that look like addresses are not changed. The next step up is to have an assembler do the dirty work and start each program with a common set of routines, in source. At around 6-8K of program code, even this becomes unmanageable since just handling a source file that is 60K or 1000K can take 15 minutes just to read it off a floppy disk. Keeping track of a program becomes harder and harder as it gets bigger, a 2K program can have a 30 page listing if it's well commented. The final answer is to produce an intermediate phase where the computer has converted all the code to internal form, i.e., the machine instructions are directly in hex or octal, but is still keeping track of jumps and other location dependent information.

Relocation Schemes

There are two general categories of relocation information: dynamic and static. Dynamic relocation refers to information still present at run-time which allows code to be moved. The relative jump that the Z-80, 6800 and 6502, as well as most large computers have is a good example. It runs the same no matter where it is in memory. Minicomputers like the PDP-11 and NOVA have enough relative-type instructions that properly written code can almost be moved between instructions and still run properly. Large computers use a base register which points to the start of the program and do all addressing relative to that. Unfortunately, most micro's either are not this flexible or require very careful, expensive coding to do so.
to keep the relocation information around until all the modules can be put together to form a final program. But in forming the final program, all addresses are resolved into absolute addresses.

A Little Linking With Your Loader?

The two things an advanced assembler/linker scheme must do are: move the code to the proper memory position, (relocation) and allow modules to call each other by name, (linking). A primitive form of linking is done in most large absolute programs by providing branch vectors. By previous agreement, a set of jumps are set-up directly after each other, each jump going to some internal module start. So if a program knows where the jump vector is, it can get to any specific routine. A full linking loader actually keeps the symbolic name assigned to the routine or data location, and matches it with the module that references the name as an externally defined symbol. Some machines (such as 6502 and 6800) have a special section of memory treated differently, called page zero or base page. A good linker should help the programmer manage page zero, as usually it's limited to 256 bytes or so, but is very important.

The Process

The final process consists of two or three steps. Each module is assembled into a 'relocatable object' module. Then all the object modules are combined by the linker into a single object module, which is still relocatable. Finally the loader converts the relocatable module into an executable load module. Very often the process is simplified by combining the linking and loading, so one program takes all the separate relocatable object modules and produces an executable load module.

In addition to the general simplification in managing smaller modules and the ability to just keep object modules for common functions, a whole new capability enters the picture. Any program which generates a similar object module can be accepted by the linker/loader. This means FORTRAN, Assembler, even BASIC modules can be combined into one program, providing the object form is correct. (NOTE–most BASIC's in use today do not fit this scheme because they do not actually generate machine code. But if the BASIC interpreter has a simplified loader built-in, it can use those other modules.)

With Reference to a Relocatable Object Format

I have now reviewed a dozen or so relocatable formats, in addition to implementing my own for a number of machines. Of those specifically for microcomputers, there are two roads of thought: binary and ASCII. The binary direction tends to be one of storage optimization, to the extreme of a pure bit stream. The ASCII direction relates to interchangeability and manageability, as ASCII is generally easier to move and manage than binary. This is partially due to the human readability. For 8080/Z-80 processors, the INTEL absolute hex format is often the starting point for building a full format.

When I started writing my relocatable, linking assembler in early 1977, I reviewed all the available formats and decided on the ASCII direction because of manageability. As I developed the scheme, I kept building with a key goal being human interaction. My format can be directly read, once the basic rules are learned.

My orientation is 8080 byte machines with disk level capability. Common address is a 16-bit value. 8-bit values generally occur as parameters that may be handled on the source level by 'Including' a parameter file at assembly time. Module linkage is handled by allowing externals between modules. Again, 16-bit values are sufficient. It is occasionally desirable to add or subtract an offset from an external or do arithmetic with externals. However, most often these practices lead to

bad programs so I decided not to allow other than simple references.

As I developed my capabilities, simple relocation was not enough. I needed the capability to separate code and data into separate segments, to ease writing PROM-able software. Hence to the normal relocatable segment (ROG), I added a RAM segment. At the same time, I added further capability to handle FORTRAN-like common blocks.

With the assembler and linker as usable tools, I built up a library of often used functions. The most useful is a module that allows character I/O to be done with disk, console, printer, reader, or punch. My selection of an ASCII format greatly simplified this library development. As to building a library, all that has to be done is to concatenate the appropriate individual modules into one file.

RLR Relocatable Object Format Specification Rev 1.0

Version 1.1 12/1/77

This is the specification of a relocatable object format for micro-processors. It is copyright (RLR 1977), but released for general use providing that acknowledgement is made to the source and any modifications are specified and returned to the source for comment.

RLR Standard Relocatable ASCII Format

a) The 'Assembler' or 'Compiler' produces an 'Object' module of ASCII records.
b) If the module contains relocating and/or linking information, it is a 'Relocatable object' module.
c) The 'Linking Loader' produces an ASCII 'Object' module or a binary 'Load' module.
d) An 'Entry' symbol is a symbol defined locally and known to other programs (alias public or internal symbol).
e) An 'External' symbol is a symbol used locally, but defined in another program.
f) A 'Local' symbol is only known within a program.
g) An 'Object' module consists strictly of ASCII characters; parity may be supplied. Each record begins with a delimiter and ends with a return and line feed. The delimiter is '=' (3DH), although data records of absolute data may begin with the Intel delimiter ':' (3AH). Numeric information is presented as hexidecimal ASCII with optional encoding. Once the encoding is removed, hex digits (0. . 9, A. . F) will remain. The encoding is used to specify relocatability or externality.
h) A 'Symbol' is up to 6 printable characters, excluding '*'. Symbols with a length of less than 6 are right filled with '*'. The system may recognize certain special symbols:
   'STACK' – top of program stack
   'MEMORY' – first location of free memory.
   '?' – Blank common (for FORTRAN)
i) All relocation is of 16-bit values, although optional formats provide use of high or low byte only. Relocation is handled by declaring program or data 'Segments.' A segment is an independently locatable section of data or code. Three standard segments are provided: ROG (for normal use or code), RAM (for data) or COM (for common data blocks, per FORTRAN). ROG and RAM segments are handled similarly as independent sections. COM is mapped so that all COM symbols in different modules refer to the same block of memory. Each segment allows for 64K addressability, unless an optional format is specified.
j) 'Mode encoded ASCII' is used to indicate relocatability or externality of data. All encoded data are 16-bit values, most significant byte first. The first character is encoded based on the meaning of the data, the remaining 3 characters are hex ASCII. All encoded data is converted into 16-bit data by resolving the relocation or other meaning and then output as 2 absolute data bytes. If appropriate (i.e., 8080/Z-80 type processors) then data is output least significant byte first. If an optional format is specified, encoding may indicate only the high or low byte is to be output. Normally, a direct mapping may be made from 2 bytes of input data, to 2 bytes of output data.
k) An 'Object file' is a file consisting of one or more object modules.

Record Formats

character	meaning
0	record mark (':' or '=')
1. . 2	record length (0 if eof record)
3. . 6	load address (may be mode encoded)
7. . 8	record type
9. . N – 2	data
N – 1. . N	checksum

Record length:

Normally contains number of data bytes in record. This implies length*2 characters of data. Symbols records specify length as number of symbols. Any record with length of zero is recognized as end of module record, regardless of record type.

Record type:

0	data record
1	symbol record or end of file record
2	name record
3	library mark
4	option record
5	data count record
6	fix-up record

Checksum:

The checksum is a total of all numeric data bytes in the record. Symbols are not added nor is the encoding information. The checksum number is the negative of the 8-bit total of all data bytes, discarding any carry. So that if all data including the checksum are added, the overall total should be zero. This is the standard Intel checksum.

Specific Record Types

Data Record (type 0)
- 0 ':' or '='
- 1. . 2 record length (data bytes) if 0, eof record with load address meaning starting adr.
- 3. . 6 load address, (MSBy first) can be mode encoded, relocatable
- 7. . 8 '00' (record type)
- 9. . (8+len*2) data, can be mode encoded. Mode encoded data never folds over record boundary
- (9+2*len). . (10+2*len) checksum

Symbol record (type 1)
- 0 '='
- 1. . 2 number of symbols
- 3. . 8 '000001' load adr & record type
- 9. . (8+12*len) symbol entries (number specified)
- (9+12*len). . (10+12*len) checksum

Each symbol entry is 12 characters:
- 0. . 5 name (6 characters, right filled with '*')
- 6. . 9 value (LSBy first, absolute hex)
- 10. . 11 type (absolute hex)

Symbol type bit assignment:
(lsb)

0	mode (0–absolute, 1–relocatable)
1	defined (1–true)
2. . 3	scope (00B–local, 1B–entry, 10B–external, 11B–common)
4	equivalance (defined via EQU in asm program)
5. . 6	segment (00B–ROG, 01B–COM, 11B, system)
7	undefined

(msb)

Value's meaning depends on symbol scope. If external, then the value is a serial number of the external. Serial numbers start at zero (0). This will be referenced when a data item is encloded as external. If common, then the value is the length of the common block which the symbol defines. Otherwise, local or entry, the value is the relative address of the symbol within the appropriate segment.

Standard type bit settings: ([ ] indicates optional values)
absolute symbol: [entry scope] + [equate] + [defined]
relocatable symbol: [entry scope] + segment + relocatable + [equate] + defined
external symbol: external scope
common symbol: common scope + defined

Name Record (type 2)

0. . 2	'=04'
3. . 6	size of ROG segment (MSBy first, absolute hex)
7. . 8	'02' (record type)
9. . 14	module name (right filled with '*')
15. .18	size of RAM segment
19. .22	size of COM segment
23. .24	checksum

Module name is used for identification and selective load processing. If an unnamed module is selectively loaded or duplicate names exist, the loader should indicate a state of confusion.

Library Mark (type 3)

0. . 2	'=01'
3. . 6	function
7. . 8	'03' (record type)
9. . 10	checksum

The function indicates that all modules to follow in the current file are either to be forced loaded or selectively loaded. The loader starts loading a file in forced load mode. If switched to selective load mode, only modules that fill unresolved externals will be loaded. However, as the module name is used to catalog modules to be loaded, no duplicate module name must exist for any module that is considered for selective loading. The same file can switch between forced and selective load.

Image Record Examples

data:
=NNAAAA00<DD*>CK<CR/LF>
N–number of data bytes
A–load address
D–data

symbol:
=NN000001<SSSSSSVVVVTT*N>CK<CR/LF>
N–number of symbols
S–symbol name
V–symbol value
T–symbol type

name:
=04RRRR02NNNNNNSSSSCCCCCK
R–ROG size
N–name
S–RAM size
C–COM size

library mark:
=01000L03CK
L–1 = selective load on
0 = selective load off
there are only 2 possibilities:
=01000003FC – off
=01000103FB – on

Mode Encoded ASCII

Mode encoded ASCII is a schema for using the rest of the information storage capacity of an ASCII character while storing hexidecimal information. Normal hex coding uses '0'. . '9', 'A'. . 'F' to represent the 16 hex digits. By selecting other character ranges of 16 characters, each range can be defined to have a specific meaning. The underlying character, after the encoding is removed, is the first digit of the value corresponding to the catagory the encoding specifics. The remaining characters of the value use the normal representation of hex data. For simplicity and human readability, all mode encoded values are 16-bit (2 bytes, 4 characters) with most significant byte first. Also, no mode encoded data is to fold between two data records. To get a full range of relocation counters the full ASCII character set must be used. If only upper case characters are used then ROG allows 64K and RAM allows 36K, but no COM is allowed.

Encodings

Externals:
First character is '@', next 3 characters are hex numbers from 000H. . FFFH. This allows a meximum of 4096 externals in any one module, but does not allow for arithmetic with externals.

Relocatable values:
First character is encoding value plus most significant digit of data value. Next 3 characters are low 3 hex digits of data value.

relocation	encoding value
ROG	'G' (47H)
RAM	'W' (57H)
COM	'g' (67H)

This table shows the first hex digit as encoded as each relocation type:

Data	Abs. Hex	ROG	RAM	COM
0	'0' (30H)	'G' (47H)	'W' (57H)	'g' (67H)
1	'1' (31H)	'H' (48H)	'X' (58H)	'i' (68H)
2	'2' (32H)	'I' (49H)	'Y' (59H)	'i' (69H)
3	'3' (33H)	'J' (4AH)	'Z' (5AH)	'j' (6AH)
4	'4' (34H)	'K' (4BH)	'[' (5BH)	'k' (6BH)
5	'5' (35H)	'L' (4CH)	'\' (5CH)	'l' (6CH)
6	'6' (36H)	'M' (4DH)	']' (5DH)	'm' (6DH)
7	'7' (37H)	'N' (4EH)	'^' (5EH)	'n' (6EH)
8	'8' (38H)	'O' 4FH)	'_' (5FH)	'o' (6FH)
9	'9' (39H)	'P' (50H)	'`' (60H)	'p' (70H)
A	'A' (41H)	'Q' (51H)	'a' (61H)	'q' (71H)
B	'B' (42H)	'R' (52H)	'b' (62H)	'r' (72H)
C	'C' (43H)	'S' (53H)	'c' (63H)	's' (73H)
D	'D' (44H)	'T' (54H)	'd' (64H)	't' (74H)
E	'E' (45H)	'U' (55H)	'e' (65H)	'u' (75H)
F	'F' (46H)	'V' (56H)	'f' (66H)	'v' (76H)

Common Relocation Resolution

ca) COM relocation is handled by a mapping process, where each module's COM segment is distributed over the actual COM blocks for the whole program. This is distinctly different from ROG or RAM segments where each byte allocated gets a different memory location. In common blocks, all block/segments with the same name get placed in the same memory location.

cb) The only entry symbol related to a COM block, is the COM symbol. Its value is the length of the COM block. No other entry may exist for any COM block.

cc) Since COM is a relocation type, program references may specify offsets into COM. Any symbols within COM are local symbols and are mapped at load time, as is any COM data.

cd) Blank common, if it exists, must be the last common label in the symbol table. The predefined name '?*****' is generated for blank common.

ce) The COM size specified in the name record is the highest COM relocation value used in a program (including space reservation by DS). The size of blank common, if any, is determined by subtracting the cumulative size of all labeled common from the size in the name record. In other words, the size in the name record 'hangs' over the end of labeled common by the length of blank common. The actual value of the bcom symbol ('?*****') is ignored.

cf) Labeled common blocks must all be defined lengths and must be the same size in all modules.

cg) Blank common may vary in size. The loader will allocate a block corresponding to the largest blank common block specified.

ch) Common module relative is mapped from one relative segment, in a module, to appropriate memory addresses based on order and size of common blocks. Each common relative reference is given a COM value based on the offset from the beginning of its common block and the total size of all preceding common blocks. At load time the process is reversed to yield an offset relative to the start of its common block. This offset is then added to the actual start of the block in memory. This way all references to any common block, even in different modules, get mapped into the same absolute memory address.

ci) To resolve a common reference: first determine which common block it refers to then resolved value: = common block absolute start + (reference-reloc. com blk start)

Example of COM mapping

module	symbol	length	COM rel	absolute address
module 1:	COM1	10H	0	1000H. .100FH
	COM2	20H	10H	1010H. .102FH
	COM3	15H	30H	1030H. .1044H
	(bcom)	20H	45H	105CH. .107BH
	————	———		
	nsize	65H	(size in name record)	
module 2:	COM2	20H	0	1010H. .102FH
	COM4	18H	20H	1045H. .105CH
	bcom	12H	38H	105CH. .106DH
	————	———		
	nsize	4AH		

MEMORY —>> 107CH (next available location)

Notice both modules' blank common are different sized. After all modules are loaded the largest blank common size is used. The next available byte is defined as 'MEMORY'. Note the way COM2 has different common relative addresses in each module, yet both are mapped into the same absolute address. Also note the gap between COM3 and blank common where the loader allocated COM4.

Optional Formats

There are a number of features that might be nice in a relocatable format but are not included here. Some are necessary for machines other than the 8080 line, for which this format was designed Others are a convenience to programmers that lead toward bad programming practice (specifically externals plus offsets). To allow for future flexibility, it was decided to specify how such additions would be added to this format and a way of telling the loader to expect same, or screan foul. The 'option select' record is used to indicate target machine and active options. For one pass compilers and others, a general fix-up record is specified. *These are suggested options and are not final, please contact the author with suggestions!*

The major deficiency of this format is that all relocatable and external data are assumed to be 16-bit values. The Z-80 has an indexed mode which can benefit from 8-bit externals and machines like 6800 & 6502 require 8-bit values for indexing and page zero addressing. As such, three schemes are provided to handle these: using the fix-up record, using the high 32K of each relocation range encoding and adding an extra byte on 8-bit references. The most desirable is using the high 32K because the consistency of the format is maintained–each input byte translates to one output byte. However only 128 8-bit externals and a split page zero relative encoding result if more than 128 bytes relative are required. If a flag character is used (such as the '@' on a 16-bit external) then no such problems occur, but the format becomes less humanly readable because the flag byte gets totally thrown away and not passed to the output. The fix-up scheme used on large computers totally destroys the primary goal that the relocatable format will be directly humanly readable, and one might as well use binary.

Option select (type 4)

0. .2	'=01'
3. .6	bits specification of options selected
	(lsb) 0–non 8080 machine (default is Z80 and 8080)
	1–use of high encoding for 8-bit data (max 128 byte page zero relative)
	2–full high encoding (256 byte PO relative)
	3–fix-up record allowed
	4–record count record allowed
	5–flag scheme for 8-bit data allowed
7. .8	'04' (record type)
9. .14	machine type (ASCII name, right filled with '*') 8080, Z80, 6800, 6502, 8085 if other use, standard name of machine
15. .16	checksum

Record count record (type 5)

0	'='
1. .2	number of counters
3. .6	overall checksum of all record checksums
7. .8	'05' (record type)
9. .10	number of type 0 records (data)
11. .12	" " 1 " (symbols+eof)
13. .14	2 (name)
15. .16	3 (library mark)
17. .18	4 (option select)
19. .20	5 (record count)
21. .22	6 (fix-up)
23. .24	checksum

The record count record is allowed to add to the security of the system. It allows checking that no previous record has been lost because of errors. Each count is an 8-bit, carry ignored checksum of all previous record checksums. No attempt has been made at error recovery, just error detection. One implicit feature of this format is that intuitive error correct can be made, if necessary, by the human operator.

Fixup Records (type 6)

0 '='
1..2 number of fixup items
3..8 '000006' (address and record type)
(9+len*5)..(10+len*5) fixup items:
<AAAA><DDDD><TT>
where: A—fixup address
D—data value
T—fixup type: 0=16-bit value
1=chain resolution
2=8-bit (hi byte)
3=8-bit (lo byte)

Both address (A) and data (D) are encoded ASCII (MSBy first). The type indicates fixup required. A chain indicates the target address is to be filled in with the data vluae, but first the contents of that address are to be kept to point to the next address in the chain. A value of zero indicates the end of the chain. Otherwise, the data value is added to the current contents of the target location. This also allows arithmetic with externals, as the data value may be an external. On 8-bit data, the arithmetic is done with the 8-bit contents of the location and the requested byte is substituted in.

Optional Encodings

The high 32K of encoding is reassigned to 8-bit values, on the assumption that no modules that big will be generated. This is largely valid, however, it is conceivable that a relocatable object module produced by combining other smaller modules could get that big.

Reassignments:

O (4FH)..V (56H) – page Zero relocatable (00H..7FH)
W (57H)..(5FH) – 1 byte externals (must be first in serial sequence too)
o (6FH)..v(76H) – page Zero relocatable (80H..0FFH)

If 2 byte values describing 8-bit data items are to be allowed, then the first character of the 4 character item is a flag. The remaining 3 then contain the hex value desired.

Flags:

'!' (21H) – page Zero relocatable
'"' (22H) – 1 byte external (lo byte)
'#' (23H) – 1 byte external (hi byte)
'$' (24H) – 1 byte common segment reference

```
0000'                   ;
0000'                   ;   THIS IS A MINI-LOADER
0000'                   ;   FOR THE RLR RELOCATABLE FORMAT
0000'                   ;
0000'                   ;       REV 0.0 12/1/77
0000'                   ;
0000'                   ;   IT IS NOT DESIGNED TO BE EXTENSIVE
0000'                   ;   ERROR CHECKING NOR PROVIDE VERY FLEXIBLE
0000'                   ;   LOAD CAPABILITIES.  IT DEMONSTRATES RESOLVING
0000'                   ;   MODE ENCODED ASCII AND IS A USEFUL LOADER
0000'                   ;
0000'                   ;   GIVEN:  STARTING LOAD ADDRESS FOR ROG SEGMENT
0000'                   ;           STARTING LOAD ADDRESS FOR RAM SEGMENT
0000'                   ;           NUMBER OF EXTERNALS
0000'                   ;           TABLE OF EXTERNALS
0000'                   ;
0000'                   ;   ACTIONS:
0000'                   ;       ONLY PROCESS DATA RECORD & EOF RECORD
0000'                   ;       IGNORE ALL OTHER RECORD TYPES
0000'                   ;
0000'                   ;   ON DATA RECORD:
0000'                   ;       PROGRAM COUNTER (PC):=RELOCATE(PC IN RECORD)
0000'                   ;       FOR EACH DATA BYTE
0000'                   ;           IF ABSOLUTE
0000'                   ;               THEN [@PC:=VALUE ; PC:=PC+1 ]
0000'                   ;           ELSE IF RELOC
0000'                   ;               THEN [@PC:=RELOCATE(VALUE),PC:=PC+2]
0000'                   ;           ELSE (*EXTERNAL*)
0000'                   ;               [ IF VALUE > NUMBER OF EXTERNALS
0000'                   ;                   THEN VALUE:=ILLEGAL EXTERNAL
0000'                   ;                   ELSE VALUE:=EXTERNAL[VALUE];
0000'                   ;                 @PC:=VALUE ; PC:=PC+2  ]
0000'                   ;   ON EOF RECORD:
0000'                   ;       RETURN (HL:=RELOCATE(ADR IN RECORD)
0000'                   ;
0000'                   ;       USER ENTRY:  RLOAD
0000'                   ;
0000'                   ;       ON ENTRY:
0000'                   ;           A=MAX. EXTERNALS (N)
0000'                   ;           BC=ROG ORIGIN
0000'                   ;           DE=RAM ORIGIN
0000'                   ;           HL=ADDRESS OF EXTERNAL TABLE
0000'                   ;
0000'                   ;       EXTERNAL TABLE FORMAT:
0000'                   ;           (2 BYTES EACH, LSBY FIRST)
0000'                   ;           ILLEGAL EXT.
0000'                   ;           EXT 0
0000'                   ;           EXT 1
0000'                   ;           EXT 2
0000'                   ;           ...
0000'                   ;           EXT 'N'
0000'                   ;
0000'                   ;   THE 3 BYTES BEFORE THE ENTRY IS A JUMP TO THE
0000'                   ;   USER GET CHAR ROUTINE
0000'                   ;   THIS IS TO ALLOW BOOTSTRAPING TO A RELOCATABLE SYSTEM
0000'                   ;   THE ROUTINE RETURNS A CHARACTER IN 'A' REG, NO PARITY
0000'                   ;       IT MAY DISTROY ALL REGISTERS
0000'                   ;
0000'                   ;           EXT     GCHRX
0000'                   ;
0000'C3 0000            ;   GCHR:   JMP     GCHRX
0003'                   ;
0003'                   ;           NAM     RLOAD
0003'                   ;           ENT     RLOAD
0003'                   ;
0003'                   ;   RLOAD:
0003'                   ;
0003'23                 ;           INX     H
0004'23                 ;           INX     H               ; SKIP ILLEGAL EXT
0005'22 00'F9           ;           SHLD    EXTTAB          ; EXT. TABLE
0008'EB                 ;           XCHG
0009'22 00'FD           ;           SHLD    REL1            ; RAM ORIGIN
000C'60                 ;           MOV     H,B
000D'69                 ;           MOV     L,C
000E'22 00'FB           ;           SHLD    RELO            ; ROG ORIGIN
0011'32 00'FF           ;           STA     MAXEXT
0014'                   ;
0014'                   ;   IGNORES ALL ERRORS UNLESS USER
```

```
0014'                   ; PATCHES TO USER ERROR JUMPS
0014'                   ;
0014'                   ; NEXT RECORD
0014'                   ;
0014'CD 00'00           NEXTRC: CALL    GCHR    ; LOOP
0017'FE 3D                      CPI     '='
0019'C2 00'14                   JNZ     NEXTRC  ; UNTIL RECORD MARK (=)
001C'                   ;
001C'AF                         XRA     A
001D'32 01'01                   STA     CKSUM   ; CKSUM:=0
0020'                   ;
0020'CD 00'61                   CALL    GHEX    ; GET LENGTH
0023'79                         MOV     A,C
0024'32 01'00                   STA     RECLEN
0027'                   ; BC:=ADDRESS (RESOLVED)
0027'CD 00'61                   CALL    GHEX
002A'B7                         ORA     A       ; IF ABS
002B'C2 00'34                   JNZ     NEXTR1
002E'C5                         PUSH    B       ; THEN GET SECOND BYTE
002F'CD 00'61                   CALL    GHEX
0032'D1                         POP     D       ; (RECOVER HI BYTE)
0033'43                         MOV     B,E
0034'                   ;
0034'69                 NEXTR1: MOV     L,C     ; HL:=LOAD ADDRESS
0035'60                         MOV     H,B
0036'                   ;
0036'                   ; IF LEN=0 THEN RETURN(HL=STARTING ADDRESS)
0036'                   ;
0036'3A 01'00                   LDA     RECLEN
0039'B7                         ORA     A
003A'C8                         RZ
003B'                   ;
003B'5F                         MOV     E,A     ; E:=REC LEN
003C'                   ;
003C'                   ; IF REC TYPE<>0 THEN IGNORE RECORD
003C'                   ;                ELSE PROCESS DATA RECORD
003C'                   ;
003C'CD 00'61                   CALL    GHEX
003F'79                         MOV     A,C
0040'B7                         ORA     A
0041'C2 00'14                   JNZ     NEXTRC  ; (IGNORE)
0044'                   ;
0044'                   ; RECTYPE=1 SO PROCESS DATA RECORD
0044'                   ;
0044'                   ; HL=LOAD ADR , E=COUNT OF BYTE LEFT
0044'                   ;
0044'CD 00'61           NEXTBY: CALL    GHEX    ; REPEAT GET DATA
0047'                   ;
0047'71                         MOV     M,C     ; STORE LOW BYTE
0048'23                         INX     H
0049'                   ;
0049'B7                         ORA     A       ; IF 2 BYTE (A>0)
004A'CA 00'50                   JZ      NEXTB1
004D'70                         MOV     M,B     ; THEN STORE HI BYTE
004E'23                         INX     H       ;    AND COUNT BYTE
004F'1D                         DCR     E
0050'                   ;(DOUBLE BYTE VALUE NEVER FOLDS OVER BETWEEN RECORDS)
0050'
0050'1D                 NEXTB1: DCR     E       ; COUNT LO BYTE
0051'C2 00'44                   JNZ     NEXTBY
0054'                   ;
0054'                   ; PROCESS CHECK SUM
0054'                   ;
0054'CD 00'61                   CALL    GHEX    ; GET CHECKSUM
0057'3A 01'01                   LDA     CKSUM
005A'B7                         ORA     A       ; IF TOTAL<>0
005B'C2 00'5E                   JNZ     NEXTB2  ; THEN ERROR
005E'                   ; USER MUST RE-DIRECT JUMP
005E'                   NEXTB2:
005E'C3 00'14                   JMP     NEXTRC
0061'                   ;
0061'                   ; GET A HEX VALUE
0061'                   ;       IF RELOC OR EXTERNAL ENCODED
0061'                   ;       THEN BC:=RESOLVED VALUE; A:=1;
0061'                   ;       ELSE (ABSOLUTE) C:=VALUE;A:=0;
0061'                   ;
0061'                   ; SAVES DE,HL
0061'                   ;
0061'D5                 GHEX:   PUSH    D
0062'E5                         PUSH    H       ; SAVE DEHL
0063'CD 00'00                   CALL    GCHR    ; GET 1ST CHAR
0066'CD 00'C4                   CALL    THEX    ; IF HEX (0..9,A..F)
0069'DA 00'73                   JC      GHEX1
006C'                   ;
006C'CD 00'DC                   CALL    DHEX    ; THEN C:=DECODE 2 HEX DIGITS
006F'AF                         XRA     A       ; TYPE:=0
0070'E1                         POP     H
0071'D1                         POP     D
0072'C9                         RET
0073'                   ;
0073'                   ; ELSE IF '@' (EXTERNAL)
0073'                   ;
0073'FE 40              GHEX1:  CPI     '@'
0075'C2 00'95                   JNZ     GHEX2
0078'                   ;
0078'CD 00'00                   CALL    GCHR    ; THEN BEGIN SKIP 2ND CHAR
007B'CD 00'D9                   CALL    DHEXF   ;           GET 2ND BYTE AS SERIAL
007E'3A 00'FF                   LDA     MAXEXT  ;         IF TOO BIG
0081'B9                         CMP     C
0082'D2 00'88                   JNC     GHEX1A
0085'01 FF FF                   LXI     B,-1    ;         THEN USE ILLEGAL EXT.
0088'                   ; USER COULD REPLACE WITH ERROR JUMP
0088'                   GHEX1A:
0088'2A 00'F9                   LHLD    EXTTAB
008B'09                         DAD     B
008C'09                         DAD     B       ; *2 BYTES
008D'4E                         MOV     C,M     ; INDEX EXT. TABLE
008E'23                         INX     H       ; WITH EXT SERIAL NUMBER
008F'46                         MOV     B,M
0090'                   ;
0090'3E 01                      MVI     A,1     ; TYPE:=1
0092'E1                         POP     H
0093'D1                         POP     D
0094'C9                         RET
0095'                   ;
0095'                   ; ELSE (RELOCATABLE)
0095'                   ;
0095'FE 47              GHEX2:  CPI     'G'     ; IF CHAR>='G'
0097'DA 00'9A                   JC      GHEX2E  ; DUMMY TOO SMAL ERR
009A'                   ;
009A'FE 56              GHEX2E: CPI     'V'     ; AND <='V'
009C'D2 00'A7                   JNC     GHEX3
009F'                   ;
009F'D6 47                      SUI     'G'     ; THEN 'ROG' RELOC.
00A1'2A 00'FB                   LHLD    RELO    ;      AND SET ORIGIN
00A4'C3 00'B1                   JMP     GHEX4
00A7'                   ; ELSE IF 'W'..SMALL 'F'
00A7'FE 67              GHEX3:  CPI     'F'+21H
00A9'DA 00'AC                   JC      GHEX3E
00AC'                   ;       DUMMY ERROR
00AC'D6 57              GHEX3E: SUI     'W'
00AE'2A 00'FD                   LHLD    REL1    ; THEN RAM SEGMENT
00B1'                   ;
00B1'                   ; COMPLETE RELOC. ADR SO BC:=VALUE; A:=1;
00B1'                   ;
00B1'E5                 GHEX4:  PUSH    H       ;SAVE SEGMENT ORIGIN
00B2'CD 00'DF                   CALL    DHEX1   ; COMPLETE HI BYTE
00B5'C5                         PUSH    B
00B6'CD 00'D9                   CALL    DHEXF   ; GET LO BYTE
00B9'D1                         POP     D
00BA'43                         MOV     B,E     ; BC:=RELATIVE ADR
00BB'E1                         POP     H
00BC'09                         DAD     B
00BD'44                         MOV     B,H
00BE'4D                         MOV     C,L     ; BC:=RESOLVED ADR
00BF'                   ;
00BF'3E 01                      MVI     A,1     ; TYPE:=1
00C1'E1                         POP     H
00C2'D1                         POP     D       ; RESTORE REGS
00C3'C9                         RET
00C4'                   ;
00C4'                   ;TEST IF HEX (CY=0) AND CONVERT TO BINARY
00C4'                   ;
00C4'FE 30              THEX:   CPI     '0'     ; IF <'0'
00C6'D8                         RC              ; THEN RETURN (NOT HEX)
00C7'                   ;
```

```
00C7'FE 3A              CPI     '9'+1       ; IF <='9'
00C9'D2 00'CF           JNC     THEX1
00CC'D6 30              SUI     '0'         ; THEN RETURN (HEX.0..9)
00CE'C9                 RET
00CF                    ;
00CF'FE 41      THEX1:  CPI     'A'         ; IF <'A'
00D1'D8                 RC
00D2'FE 47              CPI     'F'+1       ; OR >'F'
00D4'3F                 CMC
00D5'D8                 RC                  ; THEN RETURN (NOT HEX)
00D6'D6 37              SUI     'O'+7       ; ELSE RETURN (HEX.A..F)
00D8'C9                 RET
00D9                    ;
00D9                    ; DECODE HEX (RETURNS BC:=VALUE)
00D9                    ;  3 ENTRIES DEPENDING ON STATE OF 1ST DIGIT
00D9                    ;
00D9'CD 00'00   DHEXF:  CALL    GCHR        ; ENTRY- GET A FRESH 1ST CHAR
00DC                    ;
00DC'CD 00'C4   DHEX:   CALL    THEX        ; ENTRY- A=CHAR ALREADY
00DF                    ;
00DF'07         DHEX1:  RLC                 ; ENTRY- A=BINARY ALREADY
00E0'07                 RLC
00E1'07                 RLC
00E2'07                 RLC
00E3'E6 F0              ANI     0F0H        ; SHIFT INTO HI NYBBLE
00E5'F5                 PUSH    PSW
00E6                    ;
00E6'CD 00'00           CALL    GCHR
00E9'CD 00'C4           CALL    THEX
00EC'E6 0F              ANI     15          ; A:=LOW NYBBLE
00EE'C1                 POP     B           ; RECOVER HI NYBBLE(IN B)
00EF'80                 ADD     B
00F0'4F                 MOV     C,A         ; C:=BYTE (BINARY)
00F1'06 00              MVI     B,0         ; B:=CLEAR
00F3'                   ; ADD BYTE INTO CHECKSUM
00F3'21 01'01           LXI     H,CKSUM
00F6'86                 ADD     M
00F7'77                 MOV     M,A
00F8'C9                 RET
00F9'                   ;
00F9'                   ; DATA REGION
00F9'
00F9'00 00      EXTTAB: DW      0           ; EXTERNALS TABLE
00FB'00 00      RELO:   DW      0           ; ROG ORIGIN
00FD'00 00      REL1:   DW      0           ; RAM ORIGIN
00FF'00         MAXEXT: DB      0           ; MAX EXTERNALS
0100'00         RECLEN: DB      0           ; STORE RECORD LENGTH
0101'00         CKSUM:  DB      0           ; RUNNING CHECKSUM
0102'                   ;

GCHR**/0000 X   GCHR**/0000'    RLOAD**/0003'<  NEXTRC/0014'
NEXTR1/0034'    NEXTBY/0044'    NEXTB1/0050'    NEXTB2/005E'
GHEX**/0061'    GHEX1*/0073'    GHEX1A/0088'    GHEX2*/0095'
GHEX2E/009A'    GHEX3*/00A7'    GHEX3E/00AC'    GHEX4*/00B1'
THEX**/00C4'    THEX1*/00CF'    DHEXF*/00D9'    DHEX**/00DC'
DHEX1*/00DF'    EXTTAE/00F9'    RELO**/00FB'    REL1**/00FD'
MAXEXT/00FF'    RECLEN/0100'    CKSUM*/0101'

MEMORY DUMP OF MODULE
LOADED AT 200H

-D200,302
0200 C3 FF FF 23 23 22 F9 02 EB 22 FD 02 60 69 22 FB  ...##"..."  ei"
0210 02 32 FF 02 CD 00 02 FE 3D C2 14 02 AF 32 01 03  .2......=.....2..
0220 CD 61 02 79 32 00 03 CD 61 02 E7 C2 34 02 C5 CD  .A.Y2...A...4...
0230 61 02 D1 43 69 60 3A 00 03 E7 C8 5F CD 61 02 79  A..Cie`:...._.A.Y
0240 E7 C2 14 02 CD 61 02 71 23 E7 CA 50 02 70 23 1D  .....A.q#..P.p#.
0250 1D C2 44 02 CD 61 02 3A 01 03 B7 C2 5E 02 C3 14  ..D..A.:....^...
0260 02 D5 E5 CD 00 02 CD C4 02 DA 73 02 CD DC 02 AF  ..........s.....
0270 E1 D1 C9 FE 40 C2 95 02 CD 00 02 CD D9 02 3A FF  ....@.........:.
0280 02 B9 D2 88 02 01 FF FF 2A F9 02 09 09 4E 23 46  ........*....N#F
0290 3E 01 E1 D1 C9 FE 47 DA 9A 02 FE 56 D2 A7 02 D6  >.....G....V....
02A0 47 2A FB 02 C3 B1 02 FE E7 DA AC 02 D6 57 2A FD  G*.........W*.
02B0 02 E5 CD DF 02 C5 CD D9 02 D1 43 E1 09 44 4D 3E  ..........C..DM>
02C0 01 E1 D1 C9 FE 30 D8 FE 3A D2 CF 02 D6 30 C9 FE  .....0....0....
02D0 41 D8 FE 47 3F D8 D6 37 C9 CD 00 02 CD C4 02 07  A..G?..7........
02E0 07 07 E6 F0 F5 CD 00 02 CD C4 02 E6 0F C1 80     ...............
02F0 4F 06 00 21 01 03 86 77 C9 00 00 00 00 00 00 00  O..!...w........
0300 00 00 00                                         ...
-^C
B.
B.TYPE RLOAD, .HEX      INTEL FORMAT ABSOLUTE LOADABLE AT 200H
:1E0200003FFFF232322F902EB22FD02606922FB0232FF02CD0002FE3DC21402AF32D2
:1E021E000103CD61027932000 3CD6102B7C23402C5CD6102D14369603A0003B7C85F14
:1E023C00CD610279B7C21402CD61027123B7CA5002702231D1DC2402CD61023A010392
:1E025A00B7C25E02C3140 2D5E5CD0002CDC402DA7302CDDC02AFE1D1C9FE40C29502FD
:1E027800CD0002CDD9023AFF02B9D2880201FFFF2AF9020909 4E23463E01E1D1C9FEFC
:1E0296004 7DA9A02FE56D2A702D6472AFB02C3B102FE67DAAC02D6572AFD02E5CDDF30
:0102F4000247
:1E02B5005CDD902D143E109444D3E01E1D1C9FE30D8FE3AD2CF02D630C9FE41D8FEB0
:1E02D3004 73FD8D637C9CD0002CDC402070707 07E6F0F5CD0002CDC402E60FC1804FA9
:1102F1000600210103867 7C9000000000000000000000B
:00000001FF

B.
B.TYPE RLOAD.REL      RELOCATABLE OBJECT MODULE
-04010202RLOAD*000000 00F3
-02000016CHRX*000008RLOAD*030007EB
=1E00000C3@0002323222F9EB22FD6049 22F0FB32F0FFCDG000FE3DC2G014AF32DE
=1EG01E00G101CDG0617932G100CDG061E7C2G034C5CDG061D143 69603AG100B7C85F24
=1EG03C00CDG06179B7C2G014CDG0617123B7CAG050702231D1DC2G044CDG0613AG101A2
=1EG05A00B7C2G05EC3G014D5E5CDG000CDG0C4DAG073CDG0DCAFE1D1C9FE40C2G095DD
=1EG07800CDG000CDG0D923AG0FFE9D2G08E01FFFF2AG0F909094E23463E01E1D1C9FE08
=1FG096004 7DAG09AFE56D2G0A7D6472AG0FBC3G0B1FE67DAG0ACD6572AG0FDE5CDG0DF3D
=1EG0B500C5CDG0D9D143E109444D3E01E1D1C9FE30D8FE3AD2G0CFD630C9FE41D8FEB6
=1EG0D30047 3FD8D637C9CDG000CDG0C4070707 07E6F0F5CDG000CDG0C4E60FC1804FB3
-11G0F1000600210186 77C9000000000000000000000F
=00000001FF
```

AD INFINITUM IN THE GREAT SOFTWARE PIRACY DEBATE

Received: 77 Nov 25

Here's my contribution to the Great Software Piracy Debate. The main facts are clear: software is very expensive to write, moderately expensive to market, but reasonably cheap to pirate. Good software is therefore rare and piracy common. Can this be corrected? I think so, by making piracy—or, more precisely, copying programs from a friend—profitable to the author of the program. Here's how it might work... When you copy a program from a friend you send a small royalty ($2.00, say) to the program's author, who included his name and address in the program itself. This royalty buys three things: (1) the right to use the program, (2) a complete set of documentation by return mail and updates later, and (3) the right to claim certain rewards from the author.

The right to use a program would be valuable for a lot of us who want an alternative less expensive than the local computer store but less underhanded than piracy.

The documentation would provide an extra incentive. Remember that *printing* is still much cheaper when you're making a lot of copies. Of course, this would also be an incentive for good documentation, since the other kind wouldn't sell.

One reward which should be available is one for demonstrating *and correcting* a program bug (defined narrowly as a failure to meet the published specifications). Of course, only the first corrector (as determined by postmark) would win the reward. Other rewards might be for adapting the program to a new operating system or hardware configuration.

I'm tired of reading complaints about software pirates and have tried to find a practical cure. Does anyone have a better idea?

James R. Van Zandt

BITS ON BOOKS AND A NIP AT NIBL

Dr. Dobb's Journal of Toothsome Topicalities and Unscrewed Inscrutabilities

Dear Mr. Warren: *Received: 78 Jan 30*

About a year ago John Craig mentioned in *Kilobaud* that he had lost his favorite reference book, which turned out to be a manual for the CDP (professional Certificate in Data Processing). I found out from my favorite computer bookstore that there are *three* such books, and some students buy *all three* for fear of missing out on something; apparently they don't understand the difference between representative questions and actual questions. If you have someone up there who has done the CDP recently you should have him review them. There are manuals from Anaheim Publishing, Arco, and Petrocelli-Charter. (The Petrocelli-Charter 2nd edition comes in two volumes, one for Petrocelli and one for Charter.)

The bookstore, incidentally, is worth a mention. It is brand new (started last summer) and is an offshoot of a local computer store and electronics manufacturer, OpAmp Computers, in Hollywood. It's called OpAmp-Technical, 1033 N. Sycamore, Hollywood 90028. She has more computer books than Cal Tech, UCLA, USC, or Technical books in Westwood. (*She* being Mrs. Lyn Karch, the daughter of the computer store owner; it's her business, not his.) She also has books on engineering, math, general electronics, film making and such. 12 noon to 5:30, Mon. thru Sat.

One book you should check out personally is: *Microprocessor Programming for Computer Hobbyists*, by Neill Graham, Tab book no. 952, paperback, $8.95. I almost fell for it. It purports to be a book about a high-level language that can be used as a substitute for machine language. The chapter organization follows the outline of a typical college text on programming, such as Ward Maurer's or Donald Knuth's. But there is no bridge between the "high-level language" and the machine language; at least, I couldn't find one. And the "high-level language" is *PL-I*.

You owe your readers an explanation and apology, even at this late date, for some of the obscurities in the source listing of NIBL. Particulary if the version in the collected Vol 1 is unchanged from that in the Nov/Dec 1976 issue of *DDJ*. Neither Mark Alexander (corrections in April) nor Jef Raskin (review in May) picked them up. At addresses 015E, 0303, 04E2, 05D0, 05E7, 064D, 0772, and 0909 there are instructions with the mnemonic JS and the hex code C4. I have two SC/MP manuals (Technical Description, Sept. 1976; Microprocessor Applications, Oct. 1976). Neither one mentions a JS instruction. C4 is the hex code for Load Immediate. Further, there seems to be missing code, or kludges, or both, at each of the above addresses: four bytes at 015E, four bytes at 0303, five bytes at 04E2, five bytes at 05D0, four bytes at 05E7, five bytes at 064D, five bytes at 0772, and minus one byte at 0909. Most of these are jumps at the ends of subroutines. I strongly suspect that the program as published won't run.

I happen to be a Dijkstra freak. If NIBL is typical of commercial software, I have another excellent reason to remain one.

If Mark Alexander is still in Silicon Gulch, and if he let the printout get away without checking it, you should have him make amends. In *Principles of Compiler Design* by Alfred V. Aho and Jeffrey D. Ullmann, there is a subset of PASCAL defined in modified BNF.# It's Aho & Uhlmman's idea of a term project for a college junior majoring in CompSci. An 8080 Tiny PASCAL interpreter done on National Semiconductor's in-house hardware should take somewhat less time than a term project. #in LALR grammar form, to boot.

Has it occurred to you that "Dope Vector" would be an ideal title for that service to physicians?

And did The Dragon decide to become The Dragon Emeritus after Don Quixote Aho and Sancho Panza-Uhlmman did him in on the cover of *Principles of Compiler Design*? (No, because Addison-Wesley published it late last year; mine is apparently a review copy.)

Watch those bits,
Mike Fern

1046 S. Westlake, #1
Los Angeles, CA 90006

P.C.C.'s DAZZLER CONTEST WINNERS

Far back in the misty past (1976) People's Computer Company sponsored a software contest for a program resulting in a new and interesting display using the Cromemco TV Dazzler. Closing date for entries was September 30, 1976. For reasons no one here seems to remember, the contest was a long time a-judging. All seven (yes, seven) entries eventually ran and it was decided to award a prize to each entry. Cromemco agreed to issue three additional $100 gift certificates and People's Computer Company will be awarding two $50 certificates. We will publish some of the programs in Dr. Dobb's*–in this issue we are presenting the first-prize line drawing software by Gordon French and Greg Yob. –TRW*

Cromemco
2432 Charleston Road
Mountain View, CA 94043

Gentlemen: January 18, 1978

I am pleased to inform you of the results of the PCC/Cromemco Dazzler contest, at long last.

First prize of the $500 Cromemco certificate was won by Gordon French and Greg Yob (Box 354, Palo Alto, CA 94301) for their line drawing software.

Second prize of the $250 Cromemco certificate was won by Barry Seip (Electrical Engineering Department, Stanford University) for his color-dot-bar generator.

The three third prize winners of $100 Cromemco certificates were won by R. J. Gragg (250 Polaris Ave., Mountain View, CA 94043) for his memory tester; Dr. Kenneth B. Welles (Nela Park Bldg. 336, Cleveland, OH 44112) for his ASCII-3x5 & 5x7; and D. Kleinman and S. Grumette (921 North La Jolla Ave., Los Angeles, CA 90046) for their mandala program.

People's Computer Company will be giving fourth prizes to Robert Korody (3131 Homestead Rd. #18E, Santa Clara, CA 95050) and David Dameron (402 E. O'Keefe St., East Palo Alto, CA 94303) consisting of $50 in books and publications.

These prize winners will be listed in the next issue of our *Dr. Dobb's Journal*.

Thank you once again for your sponsorship of this contest. Best wishes for continued success at Cromemco.

Sincerely yours,
Robert B. Jacobsen
Executive Director
People's Computer Co.

ABSOLUTELY dazzling DAZZLEWARE

BY GORDON FRENCH AND GREGORY YOB
Box 354
Palo Alto, CA 94301
326-4039, 325-4209

A PHILOSOPHICAL RAMBLE

Our software is a tool for others to use. Rather than make a program capable of some special effects and little else, we provide two essential tools which make the TV Dazzler a genuine graphics display rather than a somewhat quirky toy.

Once having the tools, no dyed-in-the-wool hacker can resist using them. In that spirit we offer some demonstrations and hacks which evolved while we were making the programs. Bear in mind that our entry consists of the tools, and not the rather crude examples of how to use the tools.

WHAT IS IT?

The accompanying assembler source listing and assembled code in hex is our entry. It includes:

1) Two routines which plot points:
   DAZL for 128 X 128 BW mode
   CAZL for 64 X 64 color mode
2) A linedrawer which calls one of the above routines
3) Miscellaneous routines:
   CLEAR initialize Dazzler and clear display
   DIAG diagonal line diagnostic test
   DRAW initialize line drawer

Look at the code for the remarks which act as a guide to the operations performed. The dot drawers locate the quadrant, compute the displacement, locate the byte involved and mask to find the bits or nybble to plot. The linedrawer recursively divides the line in halves until dot-level, then plots the line as points.

THE GORY DETAILS

A) Prerequisites

The following material assumes that you know a few things:

1) The TV Dazzler documentation
2) 8080 Assembler
3) BASIC
4) How to load, map and access Processor Technology 5K BASIC (software #2) available from PTC for $15.00

B) Drawing dots

1) Use of DAZL for 128 X 128 BW mode
   DAZL takes as input the cells:
   XVAL = X– co ordinate
   YVAL = Y– co ordinate
   ZVAL = color (0 is black, 1 is white)

DAZL plots the dot of 'color' ZVAL at the location XVAL, YVAL on the display. 0, 0 is the lower left corner, and 127, 127 the upper right corner.

Warning: If XVAL or YVAL exceed 127, odd things can happen. This is usually wraparound, and occasionally things blow up!!! It is the driving program's responsibility to check the values.

2) Use of CAZL 64 X 64 BW or color mode
   This is very similar to DAZL:
   XVAL, YVAL = co-ordinates (0 to 63)
   ZVAL = color (low 4 bits 0 to 15)
   (grey or color depending on mode)

To the user DAZL and CAZL are very similar. Internally they differ because different Dazzler modes change the addressing involved. Both routines execute in less than 110 cycles which provides a maximum plotting rate of about 4000 dots per second. If you use BASIC, the rates will be very much slower due to the BASIC overhead. Typical of these programs is 3-5 points per second. If the code is entered at 'START', a diagonal line will be drawn. This is a nice test that all is OK with the load.

C) Drawing lines

The linedrawer starts at 'DINIT'. It draws a line starting at the coordinates in cells XZERO, YZERO to those of XONE, YONE. The linedrawer calls either CAZL or DAZL, so XVAL, etc. are also used.

At present, the linedrawer calls DAZL. To call CAZL instead, change line #1480 to:

Call CAZL

(This is at 'PLOT' +2)

These routines let you handle any dot or line graphics. The awkward co-ordinates of the raw Dazzler are tamed to a form you can use easily.

D) Point plotting with BASIC

Now the fun begins. The demo programs are set up for 128 X 128 BW mode. By appropriate address changes, the other modes may be used. Be sure to change the scaling factors. If you do not have PTC 5K BASIC, use the facilities of your BASIC to set the values of the cells and call the routines. As PTC BASIC is available for a small charge, get a copy if you prefer.

Program 1

This draws a spiral with a sinewave superposed on it. This can make 'doilies' and crochet-like patterns. The assignment statements of the form:

X = ARG (Y)

Pass the 16 bit values of Y to the H and L registers prior to an assembler call. The program must then save these values in the appropriate cells. This is a little more indirect than Poke, etc. in MITS BASIC.

C = CALL (Decimal address)

Transfers control to the specified address which must be represented in decimal.

Looking at Program 1, Line 20 clears H and L, and Line 25 calls the initialization routine which turns the Dazzler on and clears the screen. This is called 'to clear'. The space between 5K and about 16K is free for your use in BASIC. The loop in line 4000 moves the angle pointer 'T' from 0 to 100 radians. Subroutine 9000 returns a radius 'R' with which the polar co-ordinate R, T is located. Subroutine 8000 transforms the R, T values to the X, Y co-ordinates used by the

dot plotting routines. Lines 4030-4040 create the 16 bit value, P, which holds the co-ordinates in the form XZERO, YZERO. 4050-4060 pass the value and call DAZL.

To play with this program, see line 9030 for a nice set of values. N1 and N2 should be small, around .1 to .01. N3 should be more than 3. If N2 is zero, you get a simple spiral. If N1 is zero, a rosette-pattern is made.

Program 2

This makes circles on circles on circles. Lines 100, 110 turn on Dazzler and clear screen. Subroutine 7000 passes the co-ordinates and draws the dot. 2040-2070 convert from $-1 < X < 1$ to $0 < X < 127$ co-ordinates. You may make a circle with the inputs:

1, 0, 0

1, 0, 0 to the input requests.

Imagine a wheel. On its rim, attach another wheel. On that wheel's rim, attach yet another wheel. Now a bug lands on the third wheel. The path of that bug for a given combination of wheel sizes and spin rates is what this program draws. We call it the loop-the-loop program. This is fun to fiddle with. We suggest small numbers for starters. Note that this program will 'fix' the values so the picture does not run off screen and possibly crash.

See line 1130. Changing 'G' will vary the density of the plot. High densities are usually less attractive to look at. We hope these rather esoteric hacks will give you some graphical joys.

E) Drawing lines with BASIC

Program 3

You may either see a repeating spiral or draw your own lines with this demonstration. The linedrawer is under 200 cycles per dot, so the maximum rate comes to around 2000 dots/second. This works out to around 20 lines/second. BASIC will make this slower.

Subroutine 1000 starts things up.

Subroutine 5000 sets up the endpoints and calls the line-drawer.

Subroutine 6000 draws a frame. A simple modification changes these to make X and Y axes.

Subroutine 7000 draws a square spiral starting from the center.

Subroutine 9000 twiddles the endpoints.

100-200 will draw lines from your input co-ordinates.

To get here you must start the program from line 50, a modification will let you draw a line starting from the end of the preceding line:

185 X0 = X1; Y0 = Y1

190 GOTO 130

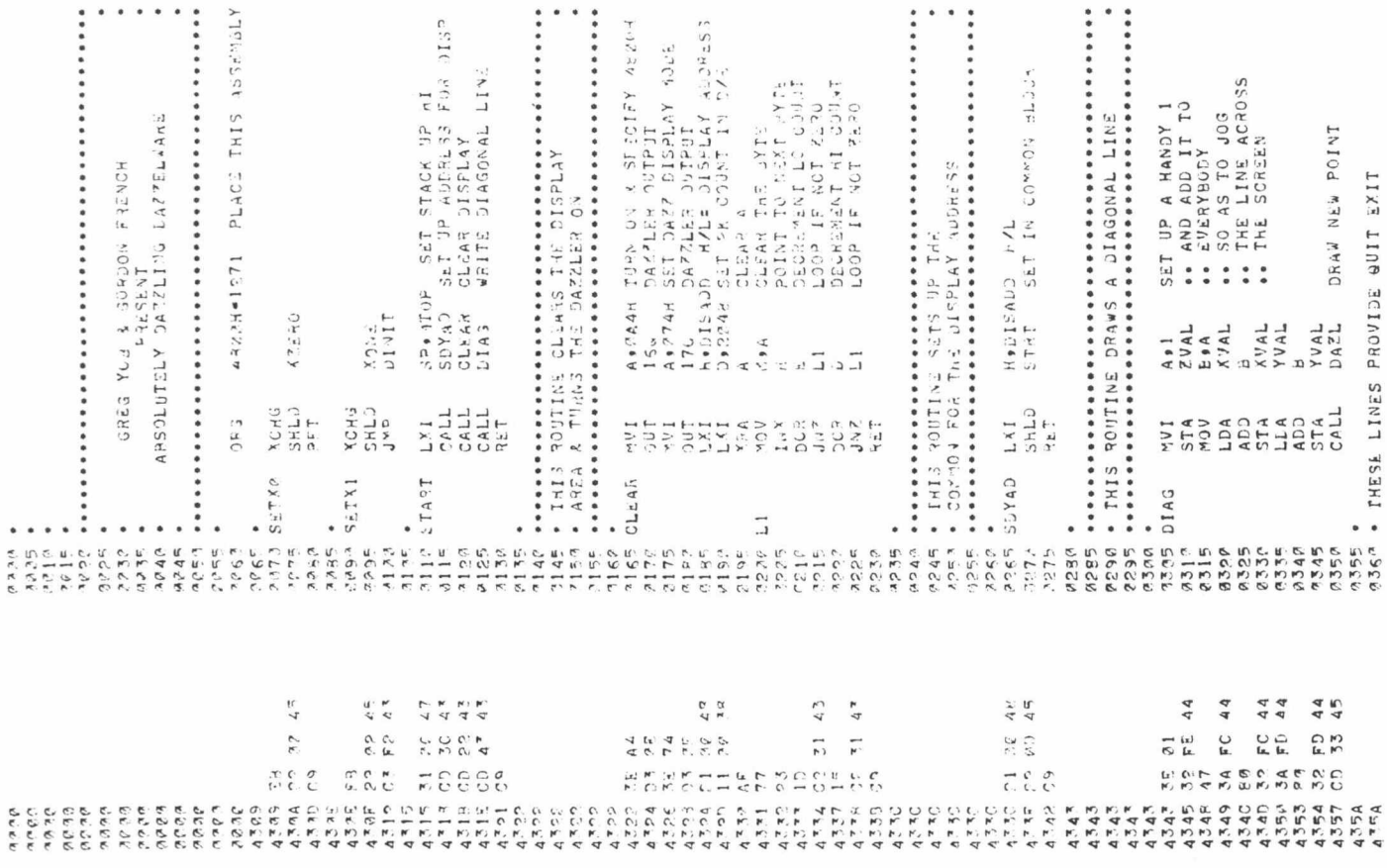
If you run from line 10, the screen is cleared, the frame drawn, and the spiral made. This is repeated. (And may be hypnotizing. . . . .)

CONCLUSION

Here are some tools. Use them freely. If you study the line-drawer, you will note the use of recursion and stacks. This could be done in other possibly better ways, but the idea was a fine bit of fun!! As for the dot plotters, they are nearly optimal and stand as a challenge to the dedicated super-hacker. We mortals will potter about in BASIC. . . .

P.S. 1) Do an inchworm or a turtle. Have the BASIC program get the commands for up, down, right, left, etc. . . .

2) We did not have a color monitor, so no color-oriented things here. Take a program like Mr. Yob's Mandala (see PCC's first games book) and have it output colors to the Dazzler instead of splats to the TTY.



```
435A
435A DB 00          0365 *
435C E6 40          0370        IN      00
435E CA 43 43       0375        ANI     64
4361 AF             0380        JZ      DIAG    CONTINUE
4362 D3 0E          0385        XRA     A       EXTINGUISH DIAPLAY
4364 C3 60 E0       0390        OUT     16Q
4367                0395        JMP     BASIC  QUIT
4367                0400 *
4367                0405 ***********************************************
4367                0410 *DAZC COLOR VERSION OF DRAW A DOT ON DAZZLER*
4367                0415 *        GIVEN XVAL, YVAL = POSITION OF DOT *
4367                0420 *                ZVAL = COLOR (0 - 15)      *
4367                0425 *             STRT   = 1ST DAZZLER BYTE     *
4367                0430 *      MAKE DESIGNATED DOT COLOR OF ZVAL     *
4367                0435 *      FOR 64 X 64 2K COLOR OR GREY DISPLAY  *
4367                0440 *                                            *
4367                0445 *        EXECUTION TIME 106 MACHINE CYCLES   *
4367                0450 *                                            *
4367                0455 *        FIRST, LOCATE THE QUADRANT          *
4367                0460 *        & COMPUTE THE OFFSET                *
4367                0465 *        CONSTANT.  SAVE IN H/L              *
4367                0470 *             NOTE: D/E=XVAN,YVAL            *
4367                0475 *                                            *
4367                0480 ***********************************************
4367                0485 *
4367 2A FC 44       0490 CAZL   LHLD    XVAL    CLEAR & SET UP REGS
436A 01 00 00       0495        LXI     B,0
436D 11 00 00       0500        LXI     D,0
4370 EB             0505        XCHG
4371 7B             0510        MOV     A,E     A=YVAL
4372 E6 20          0515        ANI     20H     TEST BIT 5
4374 C2 46 45       0520        JNZ     DAZ1
4377 21 00 04       0525        LXI     H,400H  H/L = 1024
437A 7A             0530 CAZ1   MOV     A,D     A = XVAL
437B E6 20          0535        ANI     20H     TEST BIT 5
437D CA 4F 45       0540        JZ      DAZ2
4380 01 00 02       0545        LXI     B,200H  B/C = 512
4383 09             0550 CAZ2   DAD     B       H/L = OFFSET
4384                0555 *
4384                0560 * SECOND, FIND BYTE LOCATION IN QUADRANT
4384                0565 * AND STORE IN B/C
4384                0570 *   NOTE: THE TRICKERY HERE IS INVOLVED!
4384                0575 *
4384 21 00 00       0580        LXI     B,0     CLEAR B/C
4387 7A             0585        MOV     A,D
4388 E6 1E          0590        ANI     01EH    MASK 0001 1110
438A 0F             0595        RRC
438B 4F             0600        MOV     C,A
438C 7B             0605        MOV     A,E
438D 2F             0610        CMA
438E E6 1F          0615        ANI     1FH     MASK 0001 1111
4390 17             0620        RAL
4391 17             0625        RAL
4392 17             0630        RAL
4393 17             0635        RAL     .       LEFT SHIFT 4X THRU CARRY
4394 D2 65 45       0640        JNC     DAZ3
4397 04             0645        INR     B
4398 81             0650 CAZ3   ORA     C
4399 4F             0655        MOV     C,A
439A 09             0660        DAD     B
439B                0665 *
439B                0670 * FIND WHICH NYBBLE IN BYTE
439B                0675 *
439B 7A             0680        MOV     A,D
439C E6 01          0685        ANI     1H      MASK 0000 0001
439E 11 00 4A       0690        LXI     D,DISADD  START OF BLOCK
43A1 19             0695        DAD     D
43A2 EB             0700        XCHG
43A3 C2 7C 45       0705        JNZ     DAZ4
43A6                0710 *
43A6                0715 * LOW NYBBLE PROCESS
43A6                0720 *
43A6 3A FE 44       0725        LDA     ZVAL
43A9 E6 0F          0730        ANI     0FH     MASK 0000 1111
43AB 47             0735        MOV     B,A
43AC 1A             0740        LDAX    D       FETCH FROM DISPLAY
43AD E6 F0          0745        ANI     0F0H    MASK PART OF BYTE TO SAV
                                                E
43AF B0             0750        ORA     B
43B0 12             0755        STAX    D
43B1 C9             0760        RET
43B2                0765 *
43B2                0770 * HIGH NYBBLE PROCESS
43B2                0775 *
43B2 3A FE 44       0780 CAZ4   LDA     ZVAL
43B5 E6 0F          0785        ANI     0FH
43B7 07             0790        RLC
43B8 07             0795        RLC
43B9 07             0800        RLC     .       LEFT SHIFT 4 TIMES
43BA 07             0805        RLC
43BB 47             0810        MOV     B,A
43BC 1A             0815        LDAX    D       FETCH BYTE FROM MEMORY
43BD E6 0F          0820        ANI     0FH     MASK PART TO SAVE
43BF B0             0825        ORA     B
43C0 12             0830        STAX    D
43C1 C9             0835        RET
43C2                0840 *
43C2                0845 ***********************************************
43C2                0850 *                                             *
43C2                0855 *  THE FOLLOWING TWO ROUTINES MAY BE RE*      *
43C2                0860 *  QUIRED BY CERTAIN BASIC PROCESSORS.        *
43C2                0865 *  THEY ARE SUPPLIED HERE AT NO CHARGE.       *
43C2                0870 *                                             *
43C2                0875 ***********************************************
43C2                0880 * UNPACK D/E & REVERSE
43C2                0885 *
43C2 7B             0890 UNPED  MOV     A,E
43C3 5A             0895        MOV     E,D
43C4 57             0900        MOV     D,A
43C5                0905 *
43C5                0910 * UNPACK D/E NO REVERSE
43C5                0915 *
43C5 EB             0920 UNPDE  XCHG
43C6 22 FC 44       0925        SHLD    XVAL
43C9 C3 33 45       0930        JMP     DAZL
43CC                0935 *
43CC                0940 * EXTINGUISH DAZZLER & RETURN
43CC                0945 *
43CC AF             0950 DAZOF  XRA     A
43CD D3 0E          0955        OUT     16Q
43CF C9             0960        RET
43D0                0965 *
43D0                0970 * FRONT END
43D0                0975 *
43D0 31 00 47       0980 DRAW   LXI     SP,STOP
43D3 2A EF D0       0985        LHLD    EBUF
43D6 22 00 45       0990        SHLD    XZERO
43D9 2A F1 D0       0995        LHLD    BBUF+2
43DC 22 22 45       1000        SHLD    XONE
43DF CD 3C 43       1005        CALL    SDYAD
43E2 CD 22 43       1010        CALL    CLEAR
43E5 3E 63          1015        MVI     A,063H  TURN ON & SPECIFY 434CH
43E7 D3 0F          1020        OUT     17Q     (BLACK AND WHITE)
43E9 CD 93 45       1025        CALL    PEND
43EC CD F2 43       1030        CALL    DINIT
43EF C9             1035        RET
43F0 00             1040        NOP     .       ADJUST (THESE WERE FOR J
                                                UR OWN
43F1 00             1045        NOP     .       ADJUST (CONVENIENCE = )
43F2                1050 *
43F2                1055 *
43F2                1060 ***********************************************
43F2                1065 *                                             *
43F2                1070 * LINE DRAWING ROUTINE                        *
43F2                1075 *                                             *
43F2                1080 ***********************************************
```

```
4352                    1085 *
4352                    1090 ********************************************
4352                    1095 *                                          *
4352                    1100 *  I N I T I A L I Z A T I O N             *
4352                    1105 *                                          *
4352                    1110 ********************************************
4352                    1115 *
4352 21 00 00           1120 DINIT  LXI    H,0     CLEAR H/L
4355 39                 1125        DAD    SP      GET STACK POINTER
4356 22 FA 44           1130        SHLD   STORE   ;;AND STORE IT AWAY
4359 31 00 45           1135        LXI    SP,DTOP  INITIALIZE STACK TOP
435C 3A 00 45           1140        LDA    XZERO   PICK UP X BASE ADDRESS
435F 57                 1145        MOV    D,A     COPY TO A
4360 3A 01 45           1150        LDA    YZERO   PICK UP Y BASE ADDRESS
4363 5F                 1155        MOV    E,A     COPY TO D/E PAIR
4364 D5                 1160        PUSH   D       PUT X0/Y0 ON STACK
4365 3A 09 45           1165        LDA    XONE    PICK UP X1 BASE ADDRESS
4368 47                 1170        MOV    B,A     COPY TO B/C PAIR
4369 3A 03 45           1175        LDA    YONE    PICK UP Y1 BASE ADDRESS
436C 4F                 1180        MOV    C,A     COPY TO B/C PAIR
436D C5                 1185        PUSH   B       PUT X1/Y1 ON STACK
436E                    1190 *
436E                    1195 * SEE IF STACK IS EMPTY
436E 21 00 00           1200 DEMTY  LXI    H,0     CLEAR H/L PAIR
4371 39                 1205        DAD    SP      H/L=STACK PTR VALUE
4372 01 00 00           1210        LXI    B,DTOP  B/C=TOP OF MEMORY
4375 79                 1215        MOV    A,C     GET LOW ORDER
4376 95                 1220        SUB    L       SUBTRACT STACK BASE
4377 CA 1F 44           1225        JMZ    DPOPIT  RETURN IF ZERO
437A 2A FA 44           1230        LHLD   STORE
437D F9                 1235        SPHL
437E C9                 1240        RET
437F                    1245 * NOTE: ONLY LOW ORDER BYTE COMPARED ABOVE
437F                    1250 * STACK IS SMALL (<256 BYTES)
437F                    1255 *
437F                    1260 ********************************************
437F                    1265 *                                          *
437F                    1270 * POP X0/Y0 INTO D/C REGISTER PAIR        *
437F                    1275 *                                          *
437F                    1280 *  CALCULATE X2=ABS(X1-X0)                 *
437F                    1285 *  CALCULATE Y2=ABS(Y1-Y0)                 *
437F                    1290 *                                          *
437F                    1295 ********************************************
437F                    1300 *
437F C1                 1305 DPOPIT POP    B       GET COORDINATES
4380 C1                 1310 DPOPI  POP    B       ; FROM STACK
4381 21 00 45           1315 DPOPA  LXI    H,XZERO  POINT TO X0/Y0 PAIR
4384 71                 1320        MOV    M,C     STORE B/C PAIR AS
4385 23                 1325        INX    H       ;; NEW X0 / Y0 PAIR
4386 70                 1330        MOV    M,B
4387 23                 1335        INX    H
4388 73                 1340        MOV    M,E     STORE D/E PAIR AS
4389 23                 1345        INX    H       ;; NEW X1 / Y1 PAIR
438A 72                 1350        MOV    M,D
438B 79                 1355        MOV    A,C
438C 93                 1360        SUB    E       FIND DIFFERENCE AND
438D D2 95 44           1365        JNC    DPOP2   ;; KEEP IT IN FLAG1
4390 2F                 1370        CMA    .       NORMALIZE DIFFERENCE
4391 3C                 1375        INR    A       ;; IF IT WAS NEGATIVE
4392 32 94 45           1380 DPOP2  STA    XTWO    KEEP DIFFERENCE AS XTWO
4395 78                 1385        MOV    A,B     NOW DO SAME THING WITH Y
                                                   ?
4396 91                 1390        SUB    C
4397 32 02 45           1395        STA    FLAG2
439A D2 A2 44           1400        JNC    DPOP3
439D 2F                 1405        CMA
439E 3C                 1410        INR    A
439F 32 02 45           1415 DPOP3  STA    L,YTWO-TABTOP
43A2 77                 1420        MOV    M,A     ;; AND KEEP Y2
43A3 2B                 1425        DCX    H       THIS ENDED UP BEING A
43A4 86                 1430        ADD    M       ;; HANDY WAY TO KEEP THE
43A5 F5                 1435        PUSH   PSW     ;; COMBINED VALUE
43A6                    1440
43A6                    1445 *
43A6                    1450 * FOXY CHECKS IF NULL-LENGTH SEGMENT
43A6                    1455 *
43A6                    1460 * PLOT DOT AT X0/Y0
43A6                    1465 *
43A6 2A 00 45           1470 PLOT   LHLD   XZERO   GET X0/Y0 PAIR
43A9 22 FC 44           1475        SHLD   XVAL    SET THEM UP AS A DATA
43AC CD 33 45           1480        CALL   DATA    ;; POINT AND PLOT THE POI
                                                   NT
43AF F1                 1485        POP    PSW
43B0 CA 0E 44           1490        JZ     DEMTY   YES, SEE IF STACK EMPTY
43B3                    1495 *
43B3                    1500 * COMPARE AND STORE X3=X0/2, Y3=Y2/2
43B3                    1505 *
43B3 21 04 45           1510        LXI    H,XTWO  POINT TO X3
43B6 16 7F              1515        MVI    D,7FH   MASK 0111 1111
43B8 7E                 1520        MOV    A,M     REG A IS X3
43B9 0F                 1525        RRC    .       MAKE X3 = X2/2
43BA A2                 1530        ANA    D       MASK X3
43BB 47                 1535        MOV    B,A     REG B NOW HAS X3
43BC 23                 1540        INX    H       POINT TO Y3
43BD 7E                 1545        MOV    A,M     MAKE Y3 = Y2/2
43BE 0F                 1550        RRC    .       DIVIDE
43BF A2                 1555        ANA    D       MASK IT
43C0 23                 1560        INX    H       POINT TO XTHRE
43C1 70                 1565        MOV    M,B     PUT 3 THERE
43C2 23                 1570        INX    H       POINT TO YTHRE
43C3 77                 1575        MOV    M,A     PUT A THERE
43C4                    1580 *
43C4                    1585 ********************************************
43C4                    1590 * CHOOSE WHICH KIND OF LINE IT IS:          *
43C4                    1595 *   CASE 1 X1>X0 & Y1>Y0                   *
43C4                    1600 *   CASE 2 X1<X0 & Y1<Y0                   *
43C4                    1605 *   CASE 3 X1>X0 & Y1<Y0                   *
43C4                    1610 *   CASE 4 X1<X0 & Y1>Y0                   *
43C4                    1615 *  AND REDUCE TO CASE 1 OR CASE 3          *
43C4                    1620 *  BY FLIPPING END POINTS                  *
43C4                    1625 ********************************************
43C4                    1630 *
43C4 2E 0A              1635        MVI    L,FLAG1-TABTOP POINT TO FLAG1
43C6 AF                 1640        XRA    A
43C7 B6                 1645        ORA    M       TEST SIGN OF FLAG1
43C8 FA 76 44           1650        JM     CHX
43CB 23                 1655        INX    H       POINT TO FLAG2
43CC AF                 1660        XRA    A
43CD B6                 1665        ORA    M       TEST SIGN OF FLAG2
43CE FA C3 44           1670        JM     CASE3
43D1 C3 96 44           1675        JMP    CASE1   NEITHER WAS SET
43D4 CD D4 44           1680 CHX    CALL   FLIP
43D7 3A 0B 45           1685        LDA    FLAG2
43DA F6 00              1690        ORI    0
43DC FA 96 44           1695        JM     CASE1   (CASE 2)
43DF C3 C8 44           1700        JMP    CASE3   (CASE 4)
43E2                    1705 *
43E2                    1710 * EXCHANGE ENDPOINTS ON LINE TO
43E2                    1715 * CONVERT CASE 2 TO CASE 1 OR CASE 4
43E2                    1720 * TO CASE 3
43E2                    1725 *
43E2 2E 02              1730 FLIP   MVI    L,XONE-TABTOP
43E4 46                 1735        MOV    B,M     GET X1 TO B
43E5 23                 1740        INX    H
43E6 4E                 1745        MOV    C,M     GET Y1 TO C
43E7 2A 00 45           1750        LHLD   XZERO   MAKE OLD X1 = X0
43EA 22 09 45           1755        SHLD   XONE
43ED 21 00 45           1760        LXI    H,XZERO  POINT TO X0
43F0 70                 1765        MOV    M,B     PUT IN X1
43F1 23                 1770        INX    H       POINT TO Y0
43F2 71                 1775        MOV    M,C     PUT IN Y1
43F3 C9                 1780        RET
43F4                    1785 *
43F4                    1790 * CASE 1 POSITIVE SLOPE LINE MOVING UP
43F4                    1795 * FIRST MIDPOINT PAIR X4,Y4
43F4                    1800 *
```

```
4496 2E 00              1805 CASE1  MVI     L,XZERO-TABTOP
4498 46                 1810        MOV     B,M     GET X0 TO B
4499 23                 1815        INX     H
449A 4E                 1820        MOV     C,M     GET Y0 TO C
449B 2E 06              1825        MVI     L,XTHRE-TABTOP
449D 56                 1830        MOV     D,M     GET X3 TO D
449E 23                 1835        INX     H
449F 5E                 1840        MOV     E,M     GET Y3 TO E
44A0 7A                 1845        MOV     A,D     MAKE X4 = X0 + X3
44A1 83                 1850        ADD     E
44A2 CA B0 44           1855        JZ      PAST1
44A5 78                 1860        MOV     A,B
44A6 82                 1865        ADD     D
44A7 57                 1870        MOV     D,A
44A8 79                 1875        MOV     A,C     MAKE Y4 = Y0 +Y3
44A9 83                 1880        ADD     E
44AA 5F                 1885        MOV     E,A
44AB 2E 00              1890        MVI     L,XZERO-TABTOP  STACK IT
44AD 46                 1895        MOV     B,M     GET X0 TO B
44AE 23                 1900        INX     H
44AF 4E                 1905        MOV     C,M     GET Y0 TO C
44B0 C5                 1910        PUSH    B       STACK IT
44B1 D5                 1915        PUSH    D
44B2                    1920 *
44B2                    1925 * SECOND MIDPOINT PAIR X4,Y4
44B2                    1930 *
44B2 2E 02              1935 PAST1  MVI     L,XONE-TABTOP
44B4 7E                 1940        MOV     A,M     GET X1 TO A
44B5 23                 1945        INX     H
44B6 46                 1950        MOV     B,M     GET Y1 TO B
44B7 2E 08              1955        MVI     L,XTHRE-TABTOP
44B9 96                 1960        SUB     M       MAKE X4 = X1 - X3
44BA 57                 1965        MOV     D,A     KEEP X4 IN D
44BB 23                 1970        INX     H
44BC 78                 1975        MOV     A,B     MAKE Y4 = Y1 - Y3
44BD 96                 1980        SUB     M
44BE 42                 1985        MOV     B,D     GET X4 TO B
44BF 4F                 1990        MOV     C,A     GET Y4 TO C
44C0 2E 02              1995        MVI     L,XONE-TABTOP
44C2 56                 2000        MOV     D,M     GET X1 TO D
44C3 23                 2005        INX     H
44C4 5E                 2010        MOV     E,M     GET Y1 TO E
44C5 C3 21 44           2015        JMP     DPOPX
44C8                    2020 *
44C8                    2025 * CASE 3  NEGATIVE SLOPE LINE MOVING DOWN
44C8                    2030 *  FIRST MIDPOINT PAIR X4,Y4
44C8                    2035 *
44C8 2E 00              2040 CASE3  MVI     L,XZERO-TABTOP
44CA 46                 2045        MOV     B,M     GET X0 TO B
44CB 23                 2050        INX     H
44CC 4E                 2055        MOV     C,M     GET Y0 TO C
44CD 2E 06              2060        MVI     L,XTHRE-TABTOP
44CF 56                 2065        MOV     D,M     GET X3 TO D
44D0 23                 2070        INX     H
44D1 5E                 2075        MOV     E,M     GET Y3 TO E
44D2 7A                 2080        MOV     A,D     MAKE X4 = X0 + X3
44D3 83                 2085        ADD     E
44D4 CA E4 44           2090        JZ      PAST2
44D7 78                 2095        MOV     A,B
44D8 82                 2100        ADD     D
44D9 57                 2105        MOV     D,A     KEEP X4 IN D
44DA 79                 2110        MOV     A,C     MAKE Y4 = Y0 + Y3
44DB 93                 2115        SUB     E
44DC 5F                 2120        MOV     E,A
44DD 2E 00              2125        MVI     L,XZERO-TABTOP
44DF 46                 2130        MOV     B,M     GET X0 TO B
44E0 23                 2135        INX     H
44E1 4E                 2140        MOV     C,M     GET Y0 TO C
44E2 C5                 2145        PUSH    B       STACK THEM
44E3 D5                 2150        PUSH    D
44E4                    2155 *
44E4                    2160 * SECOND MIDPOINT PAIR X4,Y4
44E4                    2165 *
```

```
44E4 2E 02              2170 PAST2  MVI     L,XONE-TABTOP
44E6 7E                 2175        MOV     A,M     GET X1 TO A
44E7 23                 2180        INX     H
44E8 46                 2185        MOV     B,M     GET Y1 TO B
44E9 2E 06              2190        MVI     L,XTHRE-TABTOP
44EB 96                 2195        SUB     M       MAKE X4 = X1 - X3
44EC 57                 2200        MOV     D,A     KEEP X4 IN D
44ED 23                 2205        INX     H
44EE 7B                 2210        MOV     A,B     MAKE Y4 = Y1 + Y3
44EF 86                 2215        ADD     M
44F0 42                 2220        MOV     B,D     GET X4 TO B
44F1 4F                 2225        MOV     C,A     GET Y4 TO C
44F2 2E 00              2230        MVI     L,XONE-TABTOP
44F4 56                 2235        MOV     D,M     GET X1 TO D
44F5 23                 2240        INX     H
44F6 5E                 2245        MOV     E,M     GET Y1 TO E
44F7 C3 21 44           2250        JMP     DPOPX
44FA                    2255 *
44FA                    2260 *
44FA                    2265 ********************************************
44FA                    2270 *                                          *
44FA                    2275 * LINE DRAWING STORAGE AREA               *
44FA                    2280 *                                          *
44FA                    2285 ********************************************
44FA                    2290 *
44FA 00 00              2295 STORE  DW      0000H   TEMP STACK STORAGE WORD
44FC 00                 2300 XVAL   DB      0       CURRENT X COORDINATE
44FD 00                 2305 YVAL   DB      0       "       Y "
44FE 01                 2310 ZVAL   DB      1       "       Z (COLOR) COORDI
                                                    NATE
44FF 01                 2315 ZLOC   DB      1       "       Z (BLK/W) COORDI
                                                    NATE
4500 00                 2320 XZERO  DB      0       X0
4501 00                 2325 YZERO  DB      0       Y0
4502 00                 2330 XONE   DB      0       X1
4503 00                 2335 YONE   DB      0       Y1
4504 00                 2340 XTWO   DB      0       X2
4505 00                 2345 YTWO   DB      0       Y2
4506 00                 2350 XTHRE  DB      0       X3
4507 00                 2355 YTHRE  DB      0       Y3
4508 00                 2360 XFOUR  DB      0       X4
4509 00                 2365 YFOUR  DB      0       Y4
450A 00                 2370 FLAG1  DB      0       INDICATOR #1
450B 00                 2375 FLAG2  DB      0       INDICATOR #2
450C 00                 2380 FOXY   DB      0       TRIX
450D                    2385 *
450D                    2390 ********************************************
450D                    2395 *DATA AREA                                  *
450D                    2400 ********************************************
450D                    2405 *
450D 00 00              2410 STRT   DW      0 FIRST ADDR OF DISPLAY PAGE
450F                    2415 TABTOP EQU     4500H
450F                    2420 ALSR   EQU     0E060H
450F                    2425 BASIC  EQU     ALSR
450F                    2430 DAV    EQU     64      DATA AVAIL FLAG BIT
450F                    2435 *
450F                    2440 * MAKE FOLLOWING LABELS "COMMON",
450F                    2445 * I.E. AVAILABLE TO EXTERNAL
450F                    2450 * PROGRAMS
450F                    2455 *
450F                    2460        COM     XVAL
450F                    2465        COM     YVAL
450F                    2470        COM     ZVAL
450F                    2475        COM     STRT
450F                    2480        COM     CALL
450F                    2485        COM     CLEAR
450F                    2490        COM     SETX0
450F                    2495        COM     SETX1
450F                    2500        COM     DRAW
450F                    2505 DISADD EQU     4800H   DAZZLER DISPLAY AREA
450F                    2510 *
450F                    2515 * THIS ROUTINE DRAWS A DIAGONAL LINE
450F                    2520 *
```

```
4525 3C 21          2525 LINE   MVI     A,1     SET UP A HANDY 1
4511 32 FF 44       2530        STA     ZLOC    PEN DOWN
4514 47             2535        MOV     B,A     ADD HANDY 1 TO
4515 3A FC 44       2540        LDA     XVAL    EVERYBODY
4518 80             2545        ADD     B       ,, EQUALLY
4519 32 FC 44       2550        STA     XVAL
451C 3A FD 44       2555        LDA     YVAL
451F 81             2560        ADD     C
4520 32 FD 44       2565        STA     YVAL
4523 CD 33 45       2570        CALL    DAZL
4526                2575 * THESE LINES PROVIDE QUIT EXIT
4526 DB 00          2580        IN      00      CHECK KEYBOARD
4528 E6 40          2585        ANI     DAV
452A CA 0F 45       2590        JZ      LINE    CONTINUE
452D AF             2595        XRA     A       EXTINGUISH DIAPLAY
452E D3 0E          2600        OUT     16D
4530 C3 60 E0       2605        JMP     BASIC   QUIT
4533                2610 *
4533                2615 ********************************************
4533                2620 * DAZL  DRAW A DOT ON THE DAZZLER          *
4533                2625 *       GIVEN XVAL, YVAL = POSITION OF DOT  *
4533                2630 *             ZVAL 1 = ON, (PEN DOWN)       *
4533                2635 *             ZVAL 0 = OFF,(PEN UP)         *
4533                2640 *             STRT   = 1ST DAZZLER BYTE     *
4533                2645 *       TURN DESIGNATED DOT "ON" OR "OFF"   *
4533                2650 *       FOR 128 X 128 B/W 2K DAZZLER DISPLAY*
4533                2655 *                                           *
4533                2660 *       EXECUTION TIME 106 MACHINE CYCLES   *
4533                2665 *                                           *
4533                2670 *       FIRST, LOCATE THE QUADRANT          *
4533                2675 *       & COMPUTE THE OFFSET                *
4533                2680 *       CONSTANT.  SAVE IN H/L              *
4533                2685 *             NOTE: D/E=XVAL,YVAL           *
4533                2690 *                                           *
4533                2695 ********************************************
4533 2A FC 44       2700 DAZL   LHLD    XVAL    CLEAR & SET UP REGS
4536 21 00 00       2725        LXI     H,0
4539 11 00 00       2710        LXI     D,0
453C EB             2715        XCHG
453D 79             2720        MOV     A,E     A=YVAL
453E E6 40          2725        ANI     40H     TEST BIT 6
4540 C2 46 45       2730        JNZ     DAZ1
4543 21 00 04       2735        LXI     H,400H  H/L = 1024
4546 7A             2740 DAZ1   MOV     A,D     A = XVAL
4547 E6 40          2745        ANI     40H     TEST BIT 6
4549 CA 4F 45       2750        JZ      DAZ2
454C 01 00 02       2755        LXI     B,200H  B/C = 512
454F 09             2760 DAZ2   DAD     B       H/L = OFFSET
4550                2765 * SECOND, FIND BYTE LOCATION IN QUADRANT
4550                2770 * AND STORE IN B/C
4550                2775 *   NOTE: THE TRICKERY HERE IS INVOLVED!
4550 01 00 00       2780        LXI     B,0     CLEAR B/C
4553 7A             2785        MOV     A,D
4554 E6 3C          2790        ANI     3CH     MASK IS 0011 1100
4556 0F             2795        RRC
4557 0F             2800        RRC
4558 4F             2805        MOV     C,A
4559 A0             2810        ANA     B
455A 78             2815        MOV     A,E
455B 2F             2820        CMA
455C E6 3F          2825        ANI     3FH     MASK IS 0011 1111
455E 17             2830        RAL
455F 17             2835        RAL
4560 17             2840        RAL     .       LEFT SHIFT 3X THR CARRY
4561 D2 65 45       2845        JNC     DAZ3
4564 04             2850        INR     B
4565 81             2855 DAZ3   ORA     C
4566 4F             2860        MOV     C,A
4567 09             2865        DAD     B
4568                2870 * PART 3 FIND DE LIDDLE BITTY BIT IN DE BYTE
4568 7A             2875        MOV     A,D
4569 E6 03          2880        ANI     03H MASK IS 0000 0011
456B 07             2885        RLC

4560 47             2890        MOV     B,A
456D 73             2895        MOV     A,E
456E E6 01          2900        ANI     01H
4570 B0             2905        ORA     B
4571                2910 * PART 4 SET UP LOTZA REGGIES
4571 37             2915        STC
4572 3F             2920        CMC
4573 01 9F 45       2925        LXI     B,TABL
4576 81             2930        ADD     C
4577 4F             2935        MOV     C,A
4578 D2 7C 45       2940        JNC     DAZ4
457B 04             2945        INR     B
457C EB             2950 DAZ4   XCHG
457D 2A 3D 45       2955        LHLD    STRT
4580 19             2960        DAD     D
4581 3A FF 44       2965        LDA     ZLOC
4584 0F             2970        RRC
4585 D2 8D 45       2975        JNC     DAZ5
4588                2980 * NOW WRITE THE DOT
4588 0A             2985        LDAX    B
4589 B6             2990        ORA     M
458A EB             2995        XCHG
458B 12             3000        STAX    D
458C C9             3005        RET
458D                3010 * OR ERASE THE DOT
458D 0A             3015 DAZ5   LDAX    B
458E 2F             3020        CMA
458F A6             3025        ANA     M
4590 EB             3030        XCHG
4591 12             3035        STAX    D
4592 C9             3040        RET
4593                3045 * SET PEN DOWN BIT
4593 3E 01          3050 PEND   MVI     A,1
4595 32 FF 44       3055        STA     ZLOC
4598 C9             3060        RET
4599                3065 * SET PEN UP
4599 AF             3070 PENUP  XRA     A
459A 32 FF 44       3075        STA     ZLOC
459D C9             3080        RET
459E 80             3085        DB      80H
459F                3090 TABL   EQU     $       MASK FOR BITS
459F 01             3095        DB      01H
45A0 01             3100        DB      01H
45A1 08             3105        DB      08H
45A2 02             3110        DB      02H
45A3 40             3115        DB      40H
45A4 10             3120        DB      10H
45A5 80             3125        DB      80H
45A6 20             3130        DB      20H
45A7                3135        ORG     4600H
4600                3140 SSTACK DS      256     SYSTEM STACK
4700                3145 MTOP   EQU     $
4700                3150 DSTACK DS      256     DRAWING SYSTEM STACK
4800                3155 DTOP   EQU     $
4800                3160 *

ALSB    E060
BASIC   E060    _0395 2605
CASE1   4496    _1675 1695
CASE3   44C8    _1670 1700
CAZ1    437A
CAZ2    4383
CAZ3    4398
CAZ4    4392
CAZL    4367
CHX     4476    _1650
CLEAR   4322    _0120 1010
DAV     0040    _2585
DAZ1    4546    _0520 2730
DAZ2    454F    _0540 2750
DAZ3    4565    _0640 2845
DAZ4    457C    _0705 2940
DAZ5    458D    _2975
```

```
DAZL    4533    0350 0930 1480 2570
DAZOF   43CC
DEMTY   440E    1490
DIAG    4343    0125 0380
DINIT   43F2    0102 1030
DISAD   4800    0185 0265 0690
DPOP1   4420
DPOP2   4435    1370
DPOP3   4442    1405
DPOPI   441F    1225
DPOPX   4421    2015 2250
DRAW    43D0
DSTAC   4700
DTOP    4800    1135 1210
FLAG1   450A    1365 1635
FLAG2   450B    1400 1685
FLIP    4434    1682
FOXY    450C
L1      4331    0215 0225
LINE    450F    2590
MTOP    4700    0110 0980
PAST1   44B2    1855
PAST2   44E4    2090
PEND    4593    1025
PENUP   4599
PLOT    444B
SDYAD   433C    0115 1005
SETX0   4309
SETX1   430E
SSTAC   4600
START   4315
STORE   44FA    1130 1230
STRT    450D    0270 2955
TABL    459F    2925
TABTO   4590    1420 1635 1730 1805 1825 1890 1935 1955 1995 2040 2060
                2125 2170 2190 2230
UNPDE   43C5
UNPED   43C2
XFOUR   4508
XONE    4502    0095 1030 1165 1730 1755 1935 1995 2170 2230
XTHRE   4506    1825 1955 2060 2190
XTWO    4504    1385 1510
XVAL    44FC    0320 0330 0490 0925 1475 2540 2550 2700
XZERO   4500    0075 0990 1140 1315 1470 1750 1760 1805 1890 2040 2125
YFOUR   4509
YONE    4503    1175
YTHRE   4507
YTWO    4505    1420
YVAL    44FD    0335 0345 0555 2565
YZERO   4501    1150
ZLOC    44FE    2530 2965 3055 3075
ZVAL    44FE    0310 0725 0780

4309: EB 22 00 45 C9 EB 22 02 45 C3 F2 43 31 00 47 CD
4319: 3C 43 CD 22 43 CD 43 43 C9 3E A4 D3 0E 3E 74 D3
4329: 0F 21 00 48 11 00 08 AF 77 23 1D C2 31 43 15 C2
4339: 31 43 C9 21 00 48 22 0D 45 C9 3E 01 32 FE 44 47
4349: 3A FC 44 80 32 FC 44 3A FD 44 80 32 FD 44 CD 33
4359: 45 DB 00 E6 40 CA 43 43 AF D3 0E C3 60 E3 2A FC
4369: 44 01 00 00 11 00 00 EB 7B E6 20 C2 46 45 21 00
4379: 04 7A E6 20 CA 4F 45 01 00 32 09 01 00 00 7A E6
4389: 1E 0F 4F 7B 2F E6 1F 17 17 17 17 D2 65 45 04 B1
4399: 4F 09 7A E6 01 11 00 48 19 EB C2 7C 45 3A FE 44
43A9: E6 0F 47 1A E6 F0 B0 12 C9 3A FE 44 E6 0F 07 07
43B9: 07 07 47 1A E6 0F B0 12 C9 7B 5A 57 EB 22 FC 44
43C9: C3 33 45 AF D3 0E C9 31 00 47 2A EF D0 22 00 45
43D9: 2A F1 D0 22 02 45 CD 3C 43 CD 22 43 3E 63 D3 0F
43E9: CD 93 45 CD F2 43 C9 00 00 21 00 00 39 22 FA 44
43F9: 31 00 48 3A 00 45 57 3A 01 45 5F D5 3A 02 45 47
4409: 3A 23 45 4F C5 21 00 00 39 31 00 48 79 95 C2 1F
4419: 44 2A FA 44 F9 C9 D1 C1 21 00 45 70 23 71 23 72
4429: 23 73 7A 90 32 0A 45 D2 35 44 2F 3C 32 04 45 78
4439: 91 32 0B 45 D2 42 44 2F 3C 2F 05 77 2B 86 F5 2A
4449: 00 45 22 FC 44 CD 33 45 F1 CA 2E 44 21 04 45 16

4459: 7F 7E 0F A2 47 23 7E 0F A2 23 70 23 77 2E 0A AF
4469: B6 FA 76 44 23 AF B6 FA C8 44 C3 96 44 CD 84 44
4479: 3A 0B 45 F6 00 FA 96 44 C3 C8 44 2E 02 46 23 4E
4489: 2A 00 45 22 02 45 21 00 45 70 23 71 C9 2E 00 46
4499: 23 4E 2E 06 56 23 5E 7A 83 CA A2 44 78 82 57 79
44A9: 83 5F 2E 00 46 23 4E C5 D5 2E 02 7E 23 46 2E 06
44B9: 96 57 23 7B 96 42 4F 2E 02 56 23 5E C3 21 44 2E
44C9: 00 46 23 4E 2E 06 56 23 5E 7A 93 CA E4 44 78 82
44D9: 57 79 93 5F 2E 00 46 23 4E C5 D5 2E 02 7E 23 46
44E9: 2E 06 96 57 23 7B 96 42 4F 2E 02 56 23 5E C3 21
44F9: 44 00 00 00 00 01 01 00 00 00 00 00 00 00 00 00
4509: 00 00 00 00 00 00 3E 01 32 FF 44 47 3A FC 44 80
4519: 32 FC 44 3A FD 44 90 32 FD 44 CD 33 45 D3 00 E6
4529: 40 CA 3F 45 AF D3 0E C3 53 E0 2A FC 44 01 00 00
4539: 11 00 00 EB 7B E6 40 C2 46 45 21 00 04 7A E6 40
4549: CA 4F 45 01 00 32 09 01 00 00 7A E6 3C 0F 0F 4F
4559: A0 78 2F E6 3E 17 17 17 D2 65 45 04 B1 4F 09 7A
4569: E6 03 07 47 7B E6 01 B0 37 3F 01 9F 45 81 4F D2
4579: 7C 45 04 EB 2A 0D 45 19 3A FF 44 0F DA 8D 45 0A
4589: B6 FB 12 C9 3A 2F A6 EB 12 C9 3E 01 32 FF 44 C9
4599: AF 32 FF 44 C9 E0 04 01 08 02 40 12 80 E0 2A
```

Program 1

```
MEM
FIRST ADDR 6484
LAST ADDR 8191
PROGRAM LOADED? Y

READY
LIST
15 Q=2*3.14159
20 X=ARG(0)
25 C=CALL(17186)
40 INPUT N1,N2,N3
50 GOSUB 4000
60 STOP
4000 FOR T = 0 TO 100 STEP .05
4010 GOSUB 9000
4020 GOSUB 8000
4030 D=D*256
4040 P=D+E
4050 X=ARG(P)
4060 C=CALL(17349)
4070 NEXT T
4080 STOP
8000 T1=T
8010 X0=R*COS(T1)+1
8020 Y0=R*SIN(T1)+1
8030 D=INT(64*X0+.5)
8040 E=INT(64*Y0+.5)
8050 RETURN
9000 R=N1*T +N2*T*SIN(N3*T)
9010 RETURN
9020 END
9030 REM FOR INPUT TRY -  .01,.013,10
```

Program 2

```
MEM
FIRST ADDR 6484
LAST ADDR 8191
PROGRAM LOADED? Y

READY
LIST
100 X=ARG(0)
110 C=CALL(17186)
1000 REM LOOP THE LOOP
1010 PRINT "3 MAGNITUDES "
1020 INPUT A1,A2,A3
1030 A4=(A1+A2+A3)*2.1
1040 A1=A1/A4
1050 A2=A2/A4
1060 A3=A3/A4
1070 P1=2*3.14159
1080 PRINT " 3 HARMONICS "
1090 INPUT P1,P2,P3
1100 B1=B1*P1
1110 B2=B2*P1
1120 B3=B3*P1
1130 G=4500
1140 G=P1/G
2000 REM OUTPUT LOOP
2010 FOR T = 0 TO 1 STEP G
2020 X=A1*COS(T*B1)+A2*COS(T*B2)+A3*COS(T*B3)
2030 Y=A1*SIN(T*B1)+A2*SIN(T*B2)+A3*SIN(T*B3)
2040 X=X+.5
2050 Y=Y+.5
2060 D=INT(X*64)
2070 E=INT(Y*64)
2080 GOSUB 7000
2090 NEXT T
2100 STOP
7000 D=D*256
7010 P=D+E
7020 X=ARG(P)
7030 C=CALL(17349)
7040 RETURN

RUN

3 MAGNITUDES

?1,0,0
 3 HARMONICS

?1,0,0
```

Program 3

```
.FIRST ADDR 6484
LAST ADDR 17160
PROGRAM LOADED? Y

READY
LIST
5 REM LINE DRAWING DEMO PROGRAM
10 GOSUB 1000
15 GOSUB 6000
20 GOTO 7000
49 REM ALTERNATE ENTRY POINT
50 GOSUB 1000
55 GOTO 100
100 REM INPUT LINE DESCRIPTION
110 PRINT "X0 -";:INPUT ,X0
120 PRINT "Y0 -";:INPUT ,Y0
130 PRINT "X1 -";:INPUT ,X1
140 PRINT "Y1 -";:INPUT ,Y1
150 GOSUB 5000
160 PRINT "NEXT STROKE OR END (0=END, 1=NEXT STROKE)"
170 INPUT A
180 IF A=0 THEN GOTO 200
190 GOTO 100
200 END
999 REM CLEAR SCREEN & INITIALIZE POINTERS SUBROUTINE
1000 Q=ARG(X0+Y0)
1010 Q=CALL(17363)
1020 RETURN
4999 REM DRAW LINE FROM X0/Y0 TO X1/Y1
5000 Q=ARG(X0*256+Y0)
5010 Q=CALL(17161)
5020 Q=ARG(X1*256+Y1)
5030 Q=CALL(17166)
5040 RETURN
5999 REM DRAW THE SURROUNDING FRAME
6000 X0=0:Y0=0:X1=0:Y1=127
6010 GOSUB 5000
6020 X0=127
6025 X1=127
6030 GOSUB 5000
6040 Y0=127
6045 X2=0
6050 GOSUB 5000
6060 X1=127
6070 Y1=0
6080 X0=0:Y0=0
6090 GOSUB 5000
6100 RETURN
6999 REM SET UP AND DRAW A SPIRAL OF STRAIGHT LINES
7000 X0=64:Y0=64:X1=64:Y1=64
7010 FOR Z = 0 TO 124 STEP 4
7020 X0=X0-Z-1
7030 GOSUB 5000
7040 GOSUB 9000
7050 Y0=Y0-Z-2
7060 GOSUB 5000
7070 GOSUB 9000
7080 X0=X0+Z+3
7090 GOSUB 5000
8000 GOSUB 9000
8010 Y0=Y0+Z+4
8020 GOSUB 5000
8030 GOSUB 9000
8040 NEXT Z
8050 FOR T = 0 TO 100
8060 NEXT T
8070 GOTO 10
8999 REM BOOTSTRAP THE BEGINNING CO-ORDINATES
9000 X1=X0:Y1=Y0
9010 RETURN
```

DAZZLER DRIVER Subroutines and a `Chase'

BY PHILLIP SIMPSON
1043 Shandwick Ct.
San Jose, CA 95136

Received: 78 Jan 6

Dear Jim and Tom:

After reading Joe Sanger's article Video Chase for the 8080/VDM *(*DDJ, *May 1977), I began to wonder why all the software for video display devices is being written for VDM's while I sit with my $200+ DAZZLER watching random color patterns. What happened to all those people who bought DAZZLER's? Why aren't they writing software for the rest of us?*

The solution was, of course, obvious . . . the reason I spent all that money on this funny looking box with the blinking lights was for the fun!? of programming. So I undertook to convert CHASE for the DAZZLER. This accomplished, I broke out the subroutines used to drive the DAZZLER, added a few more bells and whistles and created a DAZZLER DRIVER. What follows is a description of the driver and my adaptation of CHASE! I've also included some examples to show how it can be used for display of moving objects and alphanumerics. It can even be used by a BASIC program.

One other note for those people who, like me, have a computer without a front panel. If your DAZZLER mysteriously turns off from time to time, check to see if you have implemented the power on clear line. In my case, the line was floating and random noise would turn off my display about every ten seconds. This can be fixed by putting a pullup resistor on pin 13 if IC 58. A simple solution but it took me almost twenty hours of trouble shooting to figure it out!

Keep up the good work!

Phillip Simpson *1043 Shandwick Ct.*
San Jose, CA 95136

DAZZLER DRIVER

The DAZZLER DRIVER subroutines form the basis for a comprehensive display driver for the DAZZLER video boards. The DAZZLER itself has been described in other publications (*Byte*, June '76, *Popular Electronics*, Feb. '76) so I won't go into any detail here. Below is a description of each subroutine, not necessarily in order:

TRANS–Line 1200

This is the core of the driver package. It translates the X and Y coordinates into a memory address. It assumes the picture is in the 64 by 64 resolution mode. If the X and Y location is in the left half of the byte, the carry bit is set. This is a modification of a program that originally appeared in the December 1976 issue of *Byte*. In my version, X=0, Y=0 is in the lower left corner of the screen.

ERASE–Line 7620

This routine will erase 256 bytes times the number in the A register starting at the memory location in H and L.

CLEAR–Line 7700

This clears 2K of memory starting at the location contained at 'MEM' (line 0222). All routines make reference to 'MEM' for the starting location of the picture. To move the picture to a different area of memory, just change the value stored at 'MEM'. Remember, it's least significant byte first then most significant byte.

PLOTB–Line 8040

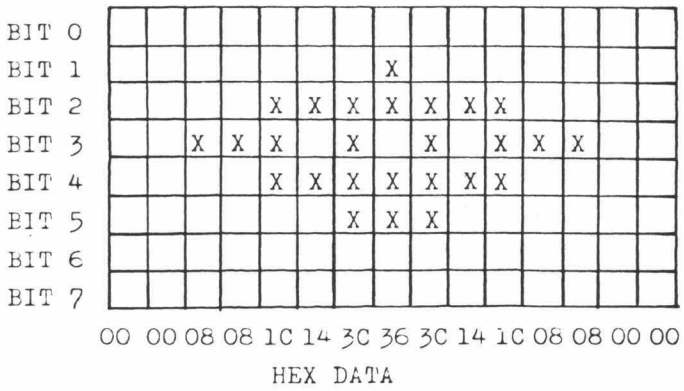
This routine will plot one element on a 128 by 128 grid. X and Y are in B and C respectively. If the carry bit is set, a white element is plotted. If the carry is reset, a black element is plotted. My original version of this used bit rotation to get to the right bit within the eight bit byte. However, after seeing Lichen Wang's solution to this problem (*DDJ*, Vol. 2, No. 8), I adopted his method.

PLOTG–Line 8300

This is used in the 128 by 128 mode. It will plot a figure of size D by E on the screen with the lower left corner at the X and Y location in B and C respectively. The location of the figure to be plotted must be in H and L. The following example plots a flying saucer on the screen, then moves it from left to right. When it disappears off the right side it reappears on the left. Note that the figure of the flying saucer has black elements on the left edge (The routine plots X, Y; X, Y+1; X, Y+2. . . . X, Y+n; X+1, Y; X+1, Y+1. . . . X+1, Y+n; etc.). This way it erases itself as it moves across the screen to the right.

```
7000            0010 * FLYING SAUCER
7000            0015 *
7000 CD1978     0100        CALL CLEAR
7003 3E7F       0105        MVI A,7FH
7005 D30F       0110        OUT 0FH
7007 016060     1000        LXI B,6060H
700A 11080E     1005        LXI D,0E08H
700D 211D7C     1010        LXI H,LOC
7C10 E5         1011 LOOP   PUSH H
7C11 D5         1012        PUSH D
7C12 C5         1013        PUSH B
7C13 CD1F78     1015        CALL PLOTG
7C16 C1         1016        POP B
7C17 D1         1017        POP D
7C18 E1         1018        POP H
7C19 04         1020        INR B
7C1A C3107C     1021        JMP LOOP
7C1D 00         1023 LOC    DB 00H
7C1E 0008       1024        DW 0800H
7C20 081C       1025        DW 1C08H
7C22 143C       1026        DW 3C14H
7C24 363C       1027        DW 3C36H
7C26 141C       1028        DW 1C14H
7C28 0808       1029        DW 0808H
7C2A 0000       1030        DW 0000H
7819            9000 CLEAR  EQU 7819H
781F            9010 PLOTG  EQU 781FH
```

Figure 1 shows how the individual bits are plotted.



Data is plotted most significant bit first. Grid can be any X and Y dimension but 8 bits high makes it easy.

FIGURE 1

PLTBY–Line 8835

This plots the nybble in register D (the four least significant bits) and the X and Y location in B and C. X and Y should be between 0 and 63. As with all of the routines, if the values exceed the limits the more significant bits are discarded. For example, 96 would be truncated to 32.

PLOTW–Line 8545

This is similar to PLOTG except that it plots one nybble at a time. Although it was designed to be used in the 64 by 64 mode, it can be used in the 128 by 128 resolution mode. X and Y must still be less than 64 but the display will be a series of dots depending upon whether or not a bit is set. The following example plots the letter 'A' on the screen using PLOTW and the 128 by 128 mode.

```
7000             0010 * PRINT AN "A"
7000             0015 *
7000 CD1978      0100        CALL CLEAR
7003 3E7F        0105        MVI A,7FH
7005 D30F        0110        OUT 0FH
7007 011D1D      1000        LXI B,1D1DH
700A 110403      1005        LXI D,0304H
700D 21167C      1010        LXI H,LOC
7010 CD2578      1015        CALL PLOTW
7013 C30000      1021        JMP 0
7016 1758        1024 LOC    DW 5817H
7018 0309        1025        DW 0903H
701A 1550        1026        DW 5015H
7819             9000 CLEAR  EQU 7819H
7825             9010 PLOTW  EQU 7825H
```

GETN–Line 6035

In many games, it is necessary to see what is in a location on the board to see what action to take next. GETN returns the value of the nybble at location X and Y specified by B and C. The value is returned in the four least significant bits of the A register.

PLOT–Line 7003

This is the first of three routines devised for the game of CHASE! Although the DAZZLER is in the 128 by 128 mode, the game 'board' is 32 by 32. Each square is made up of four nybbles. For example, to put an 'X' at location 5, 5 load 5 into both the B and C registers, 96 hex into D, 69 hex into E, then call PLOT.

GET–Line 7210

Get returns the value of the two bytes at the 'board' location in B and C. It is used by the COMP routine to see if the location contains a man, robot, or fence.

COMP–Line 7505

This tests for the presence of a figure at the X and Y 'board' location in B and C. The zero is set if the figure in D and E matches that at location B and C.

Access From A BASIC Program

In order to access the routines from a BASIC program, first POKE the values needed for the various registers in the locations provided (lines 0200 to 0215). Then CALL the appropriate routine.

BGETN–Line 2125

This will get the nybble addressed by the X and Y bytes POKED into location BC and put it in location ACM.

BPLTN–Line 2310

The nybbles stored in the least significant bits at DE are plotted at the X and Y location stored at BC.

BPLTB–Line 2310

The low order bit of the byte stored at ACM is plotted onto the screen at the X and Y location stored at BC. This is intended for use in the 128 by 128 mode.

Here is an example of a program that takes the X and Y values and the color (Z) and plots it using the DAZZLER.

```
LIST
10 REM  BASIC PROGRAM TO PLOT COLORS
20 REM  IN 64 BY 64 MODE. Z IS COLOR.
30 REM  Z IS IN THE RANGE 0 TO 15.
40 REM
100 CALL 30745
105 OUT 15,48
200 INPUT "X=";X
205 INPUT "Y=";Y
210 INPUT "Z=";Z
300 POKE 30721,X
305 POKE 30720,Y
310 POKE 30722,Z
315 CALL 30766
400 GOTO 200

READY
```

CHASE!

The rules are the same as in Sanger's version. However, scoring and comments are output to a teletype or TV Typewriter rather than displayed on the screen. Also, there is no compass rose displayed so you must remember the moves:

T Y U
G H J
B N M

where H means no move.

User supplied routines are:

MONITOR

Puts the address of your monitor at 7CCEH. This is where it

will jump when you press the escape key (7EH).

OUTA

Outputs the character in the A register. Used at 7B20H, 7E9AH, and 7EA5H.

CRLF

Outputs a carriage return and linefeed. Used at 7B0BH, 7B14H, 7B2EH, and at several places between 7EC8H and 7EF2H, also at 7E97H.

STACK

The location of your stack should be loaded at 7B08H and 7B2BH.

INP

The input routine at 7CBCH will have to be tailored to suit your system.

GOOD LUCK!

```
7000            0100 ********************
7000            0105 *                  *
7000            0110 * DAZZLER SUB-      *
7000            0115 *   ROUTINES       *
7000            0120 *                  *
7000            0125 ********************
7000            0130 *
7000            0135 * 12/21/77
7000            0140 *
7000            0145 *
7000            0180 * STORAGE AREA
7000            0181 * FOR ENTRY
7000            0182 * FROM BASIC.
7000            0183 *
7000 0000       0200 BC   DW 0
7002 0000       0205 DE   DW 0
7004 0000       0210 HL   DW 0
7006 0000       0215 ACM  DW 0
7008            0220 *
7008 0070       0222 MEM DW 7000H
700A            0225 *
700A C33478     0500       JMP TRANS
700D C3A178     0505       JMP PLOT
7010 C3C378     0510       JMP GET
7013 C3E178     0515       JMP COMP
7016 C3F078     0520       JMP ERASE
7019 C3FE78     0525       JMP CLEAR
701C C30E79     0530       JMP PLOTB
701F C35779     0535       JMP PLOTG
7022 C3027A     0540       JMP PLTBY
7025 C3A779     0545       JMP PLOTW
7028 C38F78     0550       JMP GETN
702B C36B78     0555       JMP BGETN
702E C37778     0560       JMP BPLTN
7031 C38378     0565       JMP BPLTB
7834            0995 *
7834            0996 *
7834            1000 * ROUTINE TO
7834            1005 * TRANSLATE B & C
7834            1010 * INTO AN ADDRESS
7834            1015 * IN H & L.
7834            1020 * E MUST CONTAIN
7834            1025 * THE MOST SIG-
7834            1030 * NIFICANT HALF OF
7834            1035 * PICTURE ADDRESS.
7834            1040 *
7834 79         1200 TRANS MOV A,C
7835 2F         1205   CMA
7836 E63F       1207   ANI 3FH
7838 4F         1210   MOV C,A
7839 78         1235   MOV A,B
783A E620       1240   ANI 20H
783C CA4578     1245   JZ QUADL
783F 1C         1250   INR E
7840 1C         1255   INR E
7841 78         1260   MOV A,B
7842 D620       1265   SBI 20H
7844 47         1270   MOV B,A
7845 79         1275 QUADL MOV A,C
7846 E620       1280   ANI 20H
7848 CA5378     1285   JZ QUADU
784B 7B         1290   MOV A,E
784C C604       1295   ADI 4H
784E 5F         1300   MOV E,A
784F 79         1305   MOV A,C
7850 D620       1310   SBI 20H
7852 4F         1315   MOV C,A
7853 78         1320 QUADU MOV A,B
7854 0F         1330   RRC
7855 47         1335   MOV B,A
7856 F5         1340   PUSH P
7857 78         1355   MOV A,B
7858 E60F       1360   ANI 0FH
785A 47         1365   MOV B,A
785B 79         1370   MOV A,C
785C 07         1375   RLC
785D 07         1376   RLC
785E 07         1380   RLC
785F 17         1385   RAL
7860 D26478     1390   JNC LOWER
7863 1C         1395   INR E
7864 E6F0       1400 LOWER ANI 0F0H
7866 80         1405   ADD B
7867 6F         1410   MOV L,A
7868 63         1415   MOV H,E
7869 F1         1417   POP P
786A C9         1420   RET
786B            2000 *
786B            2005 *
786B            2010 * THE NEXT THREE
786B            2015 * ROUTINES ARE
786B            2020 * ACCESSED FROM A
786B            2025 * BASIC PROGRAM.
786B            2030 * THE REGISTERS
786B            2035 * ARE FIRST POKED
786B            2040 * INTO THE STORAGE
786B            2045 * PROVIDED.
786B            2050 * THE DESIRED
786B            2055 * ROUTINE IS THEN
786B            2060 * ENTERED THROUGH
786B            2065 * THE JMP CODE
786B            2070 * AT THE BEGINING
786B            2075 * OF THIS LISTING.
786B            2080 *
786B            2100 * BGETN GETS THE
786B            2105 * NYBBLE ADDRESED
786B            2110 * BY B&C AND PUTS
786B            2115 * IT IN ACM.
786B            2120 *
786B 2A0078     2125 BGETN LHLD BC
786E 44         2130       MOV B,H
786F 4D         2135       MOV C,L
7870 CD8F78     2140       CALL GETN
7873 320678     2145       STA ACM
7876 C9         2150       RET
7877            2155 *
7877            2160 *
7877            2200 * BPLTN PLOTS THE
7877            2205 * NYBBLE IN DE AT
7877            2210 * THE LOCATION IN
7877            2215 * BC
```

```
7877            2220 *
7877 2A0078     2225 BPLTN LHLD BC
787A 44         2230       MOV B,H
787B 4D         2235       MOV C,L
787C 3A0278     2240       LDA DE
787F 57         2245       MOV D,A
7880 C3027A     2250       JMP PLTBY
7883            2300 *
7883            2305 *
7883            2310 * BPLTB PLOTS THE
7883            2315 * LOW ORDER BIT
7883            2320 * IN ACM INTO THE
7883            2325 * X&Y LOCATION IN
7883            2330 * BC.  128*128 MODE
7883            2335 *
7883 2A0078     2340 BPLTB LHLD BC
7886 44         2345       MOV B,H
7887 4D         2350       MOV C,L
7888 3A0678     2355       LDA ACM
788B 0F         2360       RRC
788C C30E79     2365       JMP PLOTB
788F            6000 *
788F            6005 *
788F            6010 * GETN RETURNS THE
788F            6015 * NYBBLE IN LOCA-
788F            6020 * TION B&C IN THE
788F            6025 * ACCUMULATOR.
788F            6030 *
788F 2A0878     6035 GETN  LHLD MEM
7892 5C         6037       MOV E,H
7893 CD3478     6040       CALL TRANS
7896 7E         6045       MOV A,M
7897 D29E78     6050       JNC RIGHT
789A 07         6055       RLC
789B 07         6060       RLC
789C 07         6065       RLC
789D 07         6070       RLC
789E E60F       6075 RIGHT ANI 0FH
78A0 C9         6080       RET
78A1            6085 *
78A1            6800 *
78A1            6801 *
78A1            6803 * PLOT PUTS THE
78A1            6805 * BYTE IN REG D
78A1            6810 * INTO THE X & Y
78A1            6815 * LOCATION IN
78A1            6820 * REG B & C AND
78A1            6825 * THE BYTE IN
78A1            6830 * REG E INTO
78A1            6835 * X,Y+1.  X & Y
78A1            6840 * MUST BE BETWEEN
78A1            6845 * 0 AND 31.  THE
78A1            6850 * LOWER LEFT
78A1            6855 * CORNER IS 0,0.
78A1            6857 *
78A1            6860 * "MEM" IS THE
78A1            6865 * STARTING AD-
78A1            6870 * DRESS OF THE
78A1            6875 * PICTURE.
78A1            6880 *
78A1 E5         7003 PLOT PUSH H
78A2 2A0878     7004   LHLD MEM
78A5 C5         7005   PUSH B
78A6 E5         7008   PUSH H
78A7 D5         7015   PUSH D
78A8 AF         7020   XRA A
78A9 78         7025   MOV A,B
78AA 17         7030   RAL
78AB 47         7035   MOV B,A
78AC 79         7040   MOV A,C
78AD 17         7045   RAL
78AE 4F         7050   MOV C,A
78AF 5C         7055   MOV E,H
78B0 C5         7060   PUSH B
78B1 CD3478     7065   CALL TRANS
78B4 C1         7070   POP B
78B5 D1         7075   POP D
78B6 72         7080   MOV M,D
78B7 0C         7085   INR C
78B8 E1         7090   POP H
78B9 D5         7095   PUSH D
78BA 5C         7096   MOV E,H
78BB CD3478     7100   CALL TRANS
78BE D1         7105   POP D
78BF 73         7110   MOV M,E
78C0 C1         7115   POP B
78C1 E1         7117   POP H
78C2 C9         7120   RET
78C3            7198 *
78C3            7199 *
78C3            7200 * GET RETURNS
78C3            7201 * THE BYTES IN
78C3            7202 * LOCATIONS X,Y
78C3            7203 * AND X,Y+1 IN
78C3            7204 * REG D & E.
78C3            7205 * ENTER WITH
78C3            7206 * X & Y IN REG
78C3            7207 * B & C
78C3            7208 *
78C3 2A0878     7210 GET LHLD MEM
78C6 C5         7215   PUSH B
78C7 E5         7220   PUSH H
78C8 AF         7225   XRA A
78C9 78         7230   MOV A,B
78CA 17         7235   RAL
78CB 47         7240   MOV B,A
78CC 79         7245   MOV A,C
78CD 17         7250   RAL
78CE 4F         7255   MOV C,A
78CF 5C         7260   MOV E,H
78D0 C5         7265   PUSH B
78D1 CD3478     7270   CALL TRANS
78D4 C1         7275   POP B
78D5 56         7280   MOV D,M
78D6 0C         7285   INR C
78D7 E1         7290   POP H
78D8 5C         7293   MOV E,H
78D9 D5         7295   PUSH D
78DA CD3478     7300   CALL TRANS
78DD D1         7305   POP D
78DE 5E         7310   MOV E,M
78DF C1         7315   POP B
78E0 C9         7320   RET
78E1            7400 *
78E1            7401 *
78E1            7404 * COMP COMPARES
78E1            7405 * THE BYTES IN
78E1            7410 * REG D & E WITH
78E1            7415 * THE BYTES IN
78E1            7420 * LOC X,Y & X,Y+1.
78E1            7425 * IT RETURNS WITH
78E1            7430 * THE ZERO BIT
78E1            7435 * SET IF THEY
78E1            7440 * ARE EQUAL.
78E1            7445 *
78E1 C5         7505 COMP PUSH B
78E2 D5         7510   PUSH D
78E3 CDC378     7515   CALL GET
78E6 C1         7520   POP B
78E7 78         7525   MOV A,B
78E8 BA         7530   CMP D
78E9 C2EE78     7535   JNZ COMP1
78EC 79         7540   MOV A,C
78ED BB         7545   CMP E
78EE C1         7547 COMP1 POP B
78EF C9         7550   RET
78F0            7600 *
78F0            7601 *
78F0            7605 * ROUTINE TO
78F0            7606 * ERASE 256*A
78F0            7607 * BYTES STARTING
78F0            7608 * AT H & L.
78F0            7609 *
```

```
78F0 1600      7620 ERASE MVI D,0
78F2 5A        7625   MOV E,D
78F3 3600      7630   MVI M,0
78F5 23        7635 NEXT INX H
78F6 13        7645   INX D
78F7 BA        7650   CMP D
78F8 C8        7655   RZ
78F9 3600      7660   MVI M,0
78FB C3F578    7665   JMP NEXT
78FE           7690 *
78FE           7691 *
78FE           7692 * CLEAR 2K
78FE           7693 * MEMORY START-
78FE           7694 * ING AT "MEM"
78FE           7695 *
78FE 2A0878    7700 CLEAR LHLD MEM
7901 E5        7705   PUSH H
7902 3E08      7710   MVI A,8
7904 CDF078    7715   CALL ERASE
7907 E1        7720   POP H
7908 7C        7725   MOV A,H
7909 37        7730   STC
790A 1F        7735   RAR
790B D30E      7740   OUT DAZ1
790D C9        7745   RET
790E           7900 *
790E           7905 *
790E           7910 * PLOT A BIT ON A
790E           7915 * 128*128 GRID.
790E           8000 * ENTER WITH X&Y
790E           8005 * IN B&C.  ON/OFF
790E           8010 * BIT IN CARRY.
790E           8035 *
790E 2A0878    8040 PLOTB LHLD MEM
7911 D5        8045       PUSH D
7912 F5        8047       PUSH P
7913 C5        8050       PUSH B
7914           8052 * DIV X BY 2
7914 78        8055       MOV A,B
7915 1F        8060       RAR
7916 E63F      8065       ANI 3FH
7918 47        8070       MOV B,A
7919 79        8075       MOV A,C
791A 1F        8080       RAR
791B E63F      8085       ANI 3FH
791D           8087 * DIV Y BY 2
791D 4F        8090       MOV C,A
791E 5C        8095       MOV E,H
791F           8097 * GET ADDRESS
791F CD3478    8100       CALL TRANS
7922           8102 * RETRIEVE X&Y
7922 C1        8105       POP B
7923 79        8106       MOV A,C
7924 2F        8110       CMA
7925 E67F      8112       ANI 7FH
7927 07        8113       RLC
7928 07        8114       RLC
7929 57        8115       MOV D,A
792A 58        8117       MOV E,B
792B 014F79    8118       LXI B,TAB
792E EB        8120       XCHG
792F 7D        8125       MOV A,L
7930 E603      8130       ANI 03H
7932 6F        8135       MOV L,A
7933 7C        8140       MOV A,H
7934 E604      8145       ANI 04H
7936 B5        8150       ORA L
7937 81        8155       ADD C
7938 6F        8160       MOV L,A
7939 60        8165       MOV H,B
793A F1        8170       POP P
793B DA4379    8175       JC WHITE
793E 3E00      8180       MVI A,0
7940 C34579    8185       JMP CONT
7943 3EFF      8190 WHITE MVI A,0FFH
7945 A6        8195 CONT  ANA M
7946 4F        8200       MOV C,A
7947 7E        8205       MOV A,M
7948 2F        8210       CMA
7949 EB        8215       XCHG
794A A6        8220       ANA M
794B B1        8225       ORA C
794C 77        8230       MOV M,A
794D D1        8235       POP D
794E C9        8240       RET
794F 01        8245 TAB   DB 01H
7950 02        8246       DB 02H
7951 10        8247       DB 10H
7952 20        8248       DB 20H
7953 04        8249       DB 04H
7954 08        8250       DB 08H
7955 40        8251       DB 40H
7956 80        8252       DB 80H
7957           8269 *
7957           8270 * B&C HAS X&Y LOC
7957           8271 * OF LOW LEFT COR-
7957           8272 * NER OF FIGURE.
7957           8273 * D&E HAS X&Y SIZE
7957           8274 * OF GRID.  H&L HAS
7957           8275 * LOC OF DATA TO
7957           8276 * BE MAPPED.
7957           8277 * 128*128 MODE
7957           8280 *
7957 22A479    8300 PLOTG SHLD LGRID
795A EB        8305       XCHG
795B 22A279    8310       SHLD GRID
795E E5        8315       PUSH H
795F CD9479    8320       CALL LDGR
7962 C5        8325       PUSH B
7963 C5        8330 LOOP1 PUSH B
7964 CD8879    8335       CALL ROTA
7967 D5        8340       PUSH D
7968 CD0E79    8345       CALL PLOTB
796B D1        8350       POP D
796C C1        8355       POP B
796D 2AA279    8360       LHLD GRID
7970 2D        8365       DCR L
7971 CA7B79    8370       JZ ADJST
7974 22A279    8375       SHLD GRID
7977 0C        8380       INR C
7978 C36379    8385       JMP LOOP1
797B C1        8390 ADJST POP B
797C E1        8395       POP H
797D 25        8400       DCR H
797E C8        8405       RZ
797F E5        8410       PUSH H
7980 22A279    8415       SHLD GRID
7983 04        8417       INR B
7984 C5        8420       PUSH B
7985 C36379    8425       JMP LOOP1
7988 3AA679    8430 ROTA  LDA FIG
798B 07        8435       RLC
798C 32A679    8440       STA FIG
798F 1D        8445       DCR E
7990 CC9479    8450       CZ LDGR
7993 C9        8455       RET
7994 2AA479    8460 LDGR LHLD LGRID
7997 7E        8465       MOV A,M
7998 32A679    8470       STA FIG
799B 23        8475       INX H
799C 22A479    8480       SHLD LGRID
799F 1E08      8485       MVI E,8
79A1 C9        8490       RET
79A2           8495 GRID  DS 2
79A4           8496 LGRID DS 2
79A6           8497 FIG   DS 1
79A7           8500 *
79A7           8505 * PLOTW PLOTS THE
79A7           8510 * FIGURE BEGINING
79A7           8515 * AT THE LOCATION
79A7           8520 * IN H&L IN A GRID
79A7           8525 * OF SIZE D&E.  THE
79A7           8530 * LOW LEFT CORNER
```

```
79A7            8535 * BEGINS AT B&C
79A7            8536 * 64*64 MODE
79A7            8540 *
79A7 22FF79     8545 PLOTW SHLD LGRBY
79AA EB         8550       XCHG
79AB 22FD79     8555       SHLD GRBY
79AE 1E00       8560       MVI E,0
79B0 E5         8562       PUSH H
79B1 C5         8565       PUSH B
79B2 C5         8570 LOOP2 PUSH B
79B3 CDCA79     8575       CALL GFIG
79B6 D5         8580       PUSH D
79B7 CD027A     8585       CALL PLTBY
79BA D1         8590       POP D
79BB C1         8595       POP B
79BC 2AFD79     8600       LHLD GRBY
79BF 2D         8605       DCR L
79C0 CAF079     8610       JZ ADJBY
79C3 22FD79     8615       SHLD GRBY
79C6 0C         8620       INR C
79C7 C3B279     8625       JMP LOOP2
79CA 7B         8630 GFIG  MOV A,E
79CB FE00       8632       CPI 0
79CD CAD779     8635       JZ LDBY
79D0 3A017A     8640       LDA BYFIG
79D3 57         8645       MOV D,A
79D4 1E00       8650       MVI E,0
79D6 C9         8655       RET
79D7 2AFF79     8660 LDBY  LHLD LGRBY
79DA 56         8665       MOV D,M
79DB 7A         8670       MOV A,D
79DC E60F       8675       ANI 0FH
79DE 32017A     8680       STA BYFIG
79E1 7A         8685       MOV A,D
79E2 0F         8690       RRC
79E3 0F         8695       RRC
79E4 0F         8700       RRC
79E5 0F         8705       RRC
79E6 E60F       8710       ANI 0FH
79E8 57         8715       MOV D,A
79E9 23         8720       INX H
79EA 22FF79     8725       SHLD LGRBY
79ED 1EFF       8730       MVI E,0FFH
79EF C9         8732       RET
79F0 C1         8735 ADJBY POP B
79F1 E1         8740       POP H
79F2 25         8745       DCR H
79F3 C8         8750       RZ
79F4 E5         8755       PUSH H
79F5 22FD79     8760       SHLD GRBY
79F8 04         8765       INR B
79F9 C5         8770       PUSH B
79FA C3B279     8775       JMP LOOP2
79FD            8780 GRBY  DS 2
79FF            8785 LGRBY DS 2
7A01            8790 BYFIG DS 1
7A02            8800 *
7A02            8805 * THIS ROUTINE
7A02            8810 * PLOTS THE MYBBLE
7A02            8815 * IN REG D AT THE
7A02            8820 * X&Y LOCATION
7A02            8825 * IN B&C
7A02            8830 *
7A02 E5         8835 PLTBY PUSH H
7A03 2A0678     8840       LHLD MEM
7A06 5C         8845       MOV E,H
7A07 7A         8846       MOV A,D
7A08 E60F       8847       ANI 0FH
7A0A 57         8848       MOV D,A
7A0B C5         8850       PUSH B
7A0C D5         8852       PUSH D
7A0D CD3478     8855       CALL TRANS
7A10 D1         8857       POP D
7A11 DA1A7A     8860       JC LEFT
7A14 7E         8865       MOV A,M
7A15 E6F0       8870       ANI 0F0H
7A17 C3237A     8875       JMP THRU
7A1A 7A         8900 LEFT  MOV A,D
7A1B 07         8905       RLC
7A1C 07         8910       RLC
7A1D 07         8915       RLC
7A1E 07         8920       RLC
7A1F 57         8925       MOV D,A
7A20 7E         8930       MOV A,M
7A21 E60F       8935       ANI 0FH
7A23 B2         8940 THRU  ORA D
7A24 77         8942       MOV M,A
7A25 C1         8943       POP B
7A26 E1         8945       POP H
7A27 C9         8950       RET
7A28            8998 *
7A28            8999 *
000E            9000 DAZ1 EQU 0EH
000F            9005 DAZ2 EQU 0FH
```

```
>D 7800 7A27

7800  00 00 00 00 00 00 00 00
7808  00 70 C3 34 78 C3 A1 78
7810  C3 C3 78 C3 E1 78 C3 F0
7818  78 C3 FE 78 C3 0E 79 C3
7820  57 79 C3 02 7A C3 A7 79
7828  C3 8F 78 C3 6B 78 C3 77
7830  78 C3 83 78 79 2F E6 3F
7838  4F 78 E6 20 CA 45 78 1C
7840  1C 78 DE 20 47 79 E6 20
7848  CA 53 78 78 C6 04 5F 79
7850  DE 20 4F 78 0F 47 F5 78
7858  E6 0F 47 79 07 07 07 17
7860  D2 64 78 1C E6 F0 80 6F
7868  63 F1 C9 2A 00 78 44 4D
7870  CD 8F 78 32 06 78 C9 2A
7878  00 78 44 4D 3A 02 78 57
7880  C3 02 7A 2A 00 78 44 4D
7888  3A 06 78 0F C3 0E 79 2A
7890  08 78 5C CD 34 78 7E D2
7898  9E 78 07 07 07 07 E6 0F
78A0  C9 E5 2A 08 78 C5 E5 D5
78A8  AF 78 17 47 79 17 4F 5C
78B0  C5 CD 34 78 C1 D1 72 0C
78B8  E1 D5 5C CD 34 78 D1 73
78C0  C1 E1 C9 2A 08 78 C5 E5
78C8  AF 78 17 47 79 17 4F 5C
78D0  C5 CD 34 78 C1 56 0C E1
78D8  5C D5 CD 34 78 D1 5E C1
78E0  C9 C5 D5 CD C3 78 C1 78
78E8  BA C2 EE 78 79 BB C1 C9
78F0  16 00 5A 36 00 23 13 BA
78F8  C8 36 00 C3 F5 78 2A 08
7900  78 E5 3E 08 CD F0 78 E1
7908  7C 37 1F D3 0E C9 2A 08
7910  78 D5 F5 C5 78 1F E6 3F
7918  47 79 1F E6 3F 4F 5C CD
7920  34 78 C1 79 2F E6 7F 07
7928  07 57 58 01 4F 79 EB 7D
7930  E6 03 6F 7C E6 04 B5 81
7938  6F 60 F1 DA 43 79 3E 00
7940  C3 45 79 3E FF A6 4F 7E
7948  2F EB A6 B1 77 D1 C9 01
7950  02 10 20 04 08 40 80 22
7958  A4 79 EB 22 A2 79 E5 CD
7960  94 79 C5 C5 CD 88 79 D5
7968  CD 0E 79 D1 C1 2A A2 79
7970  2D CA 7B 79 22 A2 79 0C
7978  C3 63 79 C1 E1 25 C8 E5
7980  22 A2 79 04 C5 C3 63 79
```

```
7988  3A A6 79 07 32 A6 79 1D
7990  CC 94 79 C9 2A A4 79 7E
7998  32 A6 79 23 22 A4 79 1E
79A0  08 C9 00 00 00 00 22 22
79A8  FF 79 EB 22 FD 79 1E 00
79B0  E5 C5 C5 CD CA 79 D5 CD
79B8  02 7A D1 C1 2A FD 79 2D
79C0  CA F0 79 22 FD 79 0C C3
79C8  B2 79 7B FE 00 CA D7 79
79D0  3A 01 7A 57 1E 00 C9 2A
79D8  FF 79 56 7A E6 0F 32 01
79E0  7A 7A 0F 0F 0F 0F E6 0F
79E8  57 23 22 FF 79 1E FF C9
79F0  C1 E1 25 C8 E5 22 FD 79
79F8  04 C5 C3 B2 79 00 00 00
7A00  00 E5 E5 2A 08 78 5C 7A
7A08  E6 0F 57 C5 D5 CD 34 78
7A10  D1 DA 1A 7A 7E E6 F0 C3
7A18  23 7A 7A 07 07 07 07 57
7A20  7E E6 0F B2 77 C1 E1 C9
```

```
7B00          0010 ****************
7B00          0015 *              *
7B00          0020 *    CHASE     *
7B00          0025 *              *
7B00          0027 *  P.H. SIMPSON *
7B00          0028 *              *
7B00          0030 ****************
7B00          0035 *
7B00          0040 * 12/20/77
7B00          0045 *
7B00          0095 *STORE 0-0 SCORE
7B00 3E30     0100 BEGIN MVI A,30H
7B02 32B47F   0105   STA YSCORE
7B05 32B37F   0110   STA RSCORE
7B08 31FF87   0115 ENTER LXI SP,STACK
7B0B CD0380   0125   CALL CRLF
7B0E          0127 * WRITE HEADING
7B0E 01FC7E   0130   LXI B,MNAME
7B11 CD947E   0135   CALL MSGO
7B14 CD0380   0137   CALL CRLF
7B17          0140 * INPUT DIFFICULTY
7B17          0141 * FACTOR
7B17 011B7F   0145 DIFF LXI B,MDIFF
7B1A CDA17E   0150   CALL MSGO1
7B1D CD8C7C   0155   CALL INP
7B20 CD2A89   0156   CALL OUTA
7B23 E60F     0160   ANI 0FH
7B25 32B27F   0165   STA DFST
7B28          1995 * CLEAR SCREEN
7B28 CD1978   2000 START CALL CLEAR
7B2B 31FF87   2005   LXI SP,STACK
7B2E CD0380   2010   CALL CRLF
7B31          2011 * SET DAZZLER
7B31          2012 * CONFIGURATION
7B31          2013 * TO 128*128
7B31 3E7F     2015   MVI A,CONF
7B33 D30F     2020   OUT DAZ2
7B35 010000   2025   LXI B,0
7B38          2030 * DRAW BOTTOM
7B38 3E0F     2035 BOTM MVI A,WIDTH
7B3A F5       2040 T1 PUSH PSW
7B3B 116996   2042   LXI D,X
7B3E CD0D78   2045   CALL PLOT
7B41 114812   2050   LXI D,SPACE
7B44 04       2055   INR B
7B45 CD0D78   2060   CALL PLOT
7B48 F1       2065   POP PSW
7B49 04       2070   INR B
7B4A 3D       2075   DCR A
7B4B C23A7B   2080   JNZ T1
7B4E          2085 * DRAW SIDE
```

```
7B4E 3E0F     2090 RSIDE MVI A,HEIGHT
7B50 F5       2095 R1 PUSH PSW
7B51 116996   2097   LXI D,X
7B54 CD0D78   2100   CALL PLOT
7B57 114812   2105   LXI D,SPACE
7B5A 0C       2110   INR C
7B5B CD0D78   2115   CALL PLOT
7B5E F1       2120   POP PSW
7B5F 0C       2125   INR C
7B60 3D       2130   DCR A
7B61 C2507B   2135   JNZ R1
7B64          2140 * DRAW TOP
7B64 3E0F     2145 TOP MVI A,WIDTH
7B66 F5       2150 B1 PUSH PSW
7B67 116996   2152   LXI D,X
7B6A CD0D78   2155   CALL PLOT
7B6D 114812   2160   LXI D,SPACE
7B70 05       2165   DCR B
7B71 CD0D78   2170   CALL PLOT
7B74 F1       2175   POP PSW
7B75 05       2180   DCR B
7B76 3D       2185   DCR A
7B77 C2667B   2190   JNZ B1
7B7A          2195 * DRAW SIDE
7B7A 3E0F     2200 LSIDE MVI A,HEIGHT
7B7C F5       2205 L1 PUSH PSW
7B7D 116996   2207   LXI D,X
7B80 CD0D78   2210   CALL PLOT
7B83 114812   2215   LXI D,SPACE
7B86 0D       2220   DCR C
7B87 CD0D78   2225   CALL PLOT
7B8A F1       2230   POP PSW
7B8B 0D       2235   DCR C
7B8C 3D       2240   DCR A
7B8D C27C7B   2245   JNZ L1
7B90          2295 * LAY POSTS
7B90 3AB27F   2300 SEED LDA DFST
7B93 07       2305   RLC
7B94 47       2310   MOV B,A
7B95 3E1E     2315   MVI A,1EH
7B97 90       2320   SUB B
7B98 57       2325   MOV D,A
7B99 CD3C7C   2330 LOOP1 CALL RND
7B9C E61F     2335   ANI 1FH
7B9E 47       2340   MOV B,A
7B9F CD3C7C   2345   CALL RND
7BA2 E61F     2350   ANI 1FH
7BA4 4F       2355   MOV C,A
7BA5 CD267C   2360   CALL LIMIT
7BA8 DA997B   2365   JC LOOP1
7BAB D5       2367   PUSH D
```

```
7BAC 116996   2370   LXI D,X
7BAF CD0D78   2375   CALL PLOT
7BB2 D1       2377   POP D
7BB3 15       2380   DCR D
7BB4 C2997B   2385   JNZ LOOP1
7BB7          2395 * PLACE ROBOTS
7BB7 CD0B7B   2400 ROBOT CALL SEL
7BBA 22B77F   2425   SHLD ROB1
7BBD CD0B7B   2430   CALL SEL
7BC0 22B97F   2445   SHLD ROB2
7BC3 CD0B7B   2450   CALL SEL
7BC6 22BB7F   2465   SHLD ROB3
7BC9 CD0B7B   2470   CALL SEL
7BCC 22BD7F   2485   SHLD ROB4
7BCF CD0B7B   2490   CALL SEL
7BD2 22BF7F   2505   SHLD ROB5
7BD5 CDFF7B   2510   CALL YOU
7BD8 C3657C   2515   JMP YMOVE
7BDB          2595 * SELECT SPOT FOR
7BDB          2596 * ROBOTS
7BDB CD3C7C   2600 SEL CALL RND
7BDE E61F     2605   ANI 1FH
7BE0 47       2610   MOV B,A
7BE1 CD3C7C   2615   CALL RND
7BE4 E61F     2620   ANI 1FH
7BE6 4F       2625   MOV C,A
7BE7 CD267C   2630   CALL LIMIT
7BEA DA0B7B   2635   JC SEL
7BED 110000   2640   LXI D,0
7BF0          2642 * IS SPOT VACANT
7BF0 CD1378   2645   CALL COMP
7BF3 C20B7B   2650   JNZ SEL
7BF6          2653 * PUT ROBOT ON
7BF6          2654 * SPOT
7BF6 111B57   2655 LROB LXI D,ROB
7BF9 CD0D78   2660   CALL PLOT
7BFC 60       2662   MOV H,B
7BFD 69       2663   MOV L,C
7BFE C9       2665   RET
7BFF          2695 * SELECT SPOT FOR
7BFF          2696 * YOU
7BFF CD3C7C   2700 YOU CALL RND
7C02 E61F     2705   ANI 1FH
7C04 47       2710   MOV B,A
7C05 CD3C7C   2715   CALL RND
7C08 E61F     2720   ANI 1FH
7C0A 4F       2725   MOV C,A
7C0B CD267C   2730   CALL LIMIT
7C0E DAFF7B   2735   JC YOU
7C11          2737 * IS SPOT VACANT
7C11 110000   2740   LXI D,0
7C14 CD1378   2745   CALL COMP
7C17 C2FF7B   2750   JNZ YOU
7C1A          2751 * PUT YOU ON SPOT
7C1A 114E46   2752 LYOU LXI D,YOUS
7C1D CD0D78   2753   CALL PLOT
7C20 60       2755   MOV H,B
7C21 69       2760   MOV L,C
7C22 22B57F   2765   SHLD YOUP
7C25 C9       2770   RET
7C26          2795 * MAKE SURE X&Y
7C26          2796 * LOCATION IS ON
7C26          2797 * THE SCREEN
7C26 78       2800 LIMIT MOV A,B
7C27 DE04     2805   SBI LOWD
7C29 37       2810   STC
7C2A F8       2815   RM
7C2B 78       2820   MOV A,B
7C2C DE1C     2825   SBI HID
7C2E 37       2830   STC
7C2F F0       2835   RP
7C30 79       2840   MOV A,C
7C31 DE04     2845   SBI  LOWE
7C33 37       2850   STC
7C34 F8       2855   RM
7C35 79       2860   MOV A,C
7C36 DE1C     2865   SBI HIE
7C38 37       2870   STC
7C39 F0       2875   RP
7C3A 3F       2880   CMC
7C3B C9       2885   RET
7C3C          2895 * RANDOM NUMBER
7C3C          2896 * GENERATOR
7C3C C5       2900 RND PUSH B
7C3D D5       2905   PUSH D
7C3E E5       2910   PUSH H
7C3F 21B17F   2915   LXI H,SH+3
7C42 0608     2920   MVI B,8
7C44 7E       2925   MOV A,M
7C45 07       2930 RTOP RLC
7C46 07       2935   RLC
7C47 07       2940   RLC
7C48 AE       2945   XRA M
7C49 17       2950   RAL
7C4A 17       2955   RAL
7C4B 2D       2960   DCR L
7C4C 2D       2965   DCR L
7C4D 2D       2970   DCR L
7C4E 7E       2975   MOV A,M
7C4F 17       2980   RAL
7C50 77       2985   MOV M,A
7C51 2C       2990   INR L
7C52 7E       2995   MOV A,M
7C53 17       3000   RAL
7C54 77       3005   MOV M,A
7C55 2C       3010   INR L
7C56 7E       3015   MOV A,M
7C57 17       3020   RAL
7C58 77       3025   MOV M,A
7C59 2C       3030   INR L
7C5A 7E       3035   MOV A,M
7C5B 17       3040   RAL
7C5C 77       3045   MOV M,A
7C5D 05       3050   DCR B
7C5E C2457C   3055   JNZ RTOP
7C61 E1       3060   POP H
7C62 D1       3065   POP D
7C63 C1       3070   POP B
7C64 C9       3075   RET
7C65          3100 * ROUTINE TO MOVE
7C65          3101 * YOU
7C65 CD8C7C   3105 YMOVE CALL INP
7C68 F5       3110   PUSH PSW
7C69 2AB57F   3115   LHLD YOUP
7C6C 44       3120   MOV B,H
7C6D 4D       3125   MOV C,L
7C6E 110000   3130   LXI D,0
7C71 CD0D78   3135   CALL PLOT
7C74 F1       3140   POP PSW
7C75 FE4A     3145   CPI 'J'
7C77 CCA07D   3150   CZ RT
7C7A FE55     3155   CPI 'U'
7C7C CCA27D   3160   CZ UR
7C7F FE59     3165   CPI 'Y'
7C81 CC9A7D   3170   CZ UP
7C84 FE54     3175   CPI 'T'
7C86 CCA57D   3180   CZ UL
7C89 FE47     3185   CPI 'G'
7C8B CC9E7D   3190   CZ LF
7C8E FE42     3195   CPI 'B'
7C90 CCA07D   3200   CZ LL
7C93 FE4E     3205   CPI 'N'
7C95 CC9C7D   3210   CZ DN
7C98 FE4D     3215   CPI 'M'
7C9A CCAB7D   3220   CZ LR
7C9D FE50     3225   CPI 'P'
7C9F CCD07D   3230   CZ PANIC
7CA2 FE48     3235   CPI 'H'
7CA4 CAE07C   3240   JZ Y1
7CA7          3242 * IS SPOT VACANT
7CA7 110000   3245   LXI D,0
7CAA CD1378   3250   CALL COMP
7CAD C2AE7D   3255   JNZ DEATH
```

```
7CB0 CD1A7C   3260 Y1 CALL LYOU
7CB3          3280 * MOVE ROBOTS
7CB3 CD027C   3285   CALL RMOVE
7CB6          3286 * ARE ALL ROBOTS
7CB6          3287 * DEAD
7CB6 CD6E7E   3290   CALL END
7CB9 C3657C   3295   JMP YMOVE
7CBC          3297 * INPUT ROUTINE
7CBC DB00     3300 INP IN STAT
7CBE E680     3305   ANI RDA
7CC0 C2BC7C   3310   JNZ INP
7CC3 DB01     3315   IN DATA
7CC5 E67F     3320   ANI 7FH
7CC7 FE0D     3325   CPI 0DH
7CC9 CA087B   3330   JZ ENTER
7CCC FE7E     3335   CPI ESC
7CCE CA0080   3340   JZ MONITOR
7CD1 C9       3345   RET
7CD2          3347 * MOVE ROBOTS
7CD2 2AB77F   3350 RMOVE LHLD ROB1
7CD5 CD1C7D   3355   CALL RDEAD
7CD8 DAE17C   3360   JC RMOV1
7CDB CD287D   3365   CALL SEEK
7CDE 22B77F   3370   SHLD ROB1
7CE1 2AB97F   3375 RMOV1 LHLD ROB2
7CE4 CD1C7D   3380   CALL RDEAD
7CE7 DAF07C   3385   JC RMOV2
7CEA CD287D   3390   CALL SEEK
7CED 22B97F   3395   SHLD ROB2
7CF0 2ABB7F   3400 RMOV2 LHLD ROB3
7CF3 CD1C7D   3405   CALL RDEAD
7CF6 DAFF7C   3410   JC RMOV3
7CF9 CD287D   3415   CALL SEEK
7CFC 22BB7F   3420   SHLD ROB3
7CFF 2ABD7F   3425 RMOV3 LHLD ROB4
7D02 CD1C7D   3430   CALL RDEAD
7D05 DA0E7D   3435   JC RMOV4
7D08 CD287D   3440   CALL SEEK
7D0B 22BD7F   3445   SHLD ROB4
7D0E 2ABF7F   3450 RMOV4 LHLD ROB5
7D11 CD1C7D   3455   CALL RDEAD
7D14 D8       3460   RC
7D15 CD287D   3465   CALL SEEK
7D18 22BF7F   3470   SHLD ROB5
7D1B C9       3475   RET
7D1C          3495 * IS ROBOT DEAD
7D1C 44       3500 RDEAD MOV B,H
7D1D 4D       3505   MOV C,L
7D1E 111B57   3510   LXI D,ROB
7D21 CD1378   3515   CALL COMP
7D24 37       3520   STC
7D25 C0       3525   RNZ
7D26 3F       3530   CMC
7D27 C9       3535   RET
7D28          3540 * COMPUTE PATH
7D28          3541 * TOWARDS YOU
7D28 110000   3550 SEEK LXI D,0
7D2B CD0078   3551   CALL PLOT
7D2E CD877D   3552   CALL SUBH
7D31 CD927D   3553   CALL SUBV
7D34 114E46   3554 CHEK LXI D,YOUS
7D37 CD1378   3555   CALL COMP
7D3A          3557 * IF ROBOT GETS
7D3A          3558 * YOU, YOU'RE DEAD
7D3A CAAE7D   3560   JZ DEATH
7D3D          3562 * HAS ROBOT BUMPED
7D3D          3563 * FENCE
7D3D 116996   3565   LXI D,X
7D40 CD1378   3570   CALL COMP
7D43 CA5B7D   3575   JZ FLASH
7D46 111B57   3576   LXI D,ROB
7D49 CD1378   3577   CALL COMP
7D4C CC5B7D   3578   CZ FLASH
7D4F 114812   3580   LXI D,SPACE
7D52 CD1378   3582   CALL COMP
7D55 CA5B7D   3585   JZ FLASH
7D58 C3F67B   3590   JMP LROB
7D5B          3595 * DEATH DANCE
7D5B E5       3600 FLASH PUSH H
7D5C 26A0     3602   MVI H,0A0H
7D5E 111B57   3605 FLAS1 LXI D,ROB
7D61 E5       3610   PUSH H
7D62 CD0078   3615   CALL PLOT
7D65 E1       3620   POP H
7D66 2EFF     3625   MVI L,0FFH
7D68 2D       3630 FLAS2 DCR L
7D69 C2687D   3635   JNZ FLAS2
7D6C 2EFF     3640   MVI L,0FFH
7D6E 119669   3645   LXI D,RID
7D71 E5       3650   PUSH H
7D72 CD0078   3655   CALL PLOT
7D75 E1       3660   POP H
7D76 2D       3665 FLAS3 DCR L
7D77 C2767D   3670   JNZ FLAS3
7D7A 25       3675   DCR H
7D7B C25E7D   3680   JNZ FLAS1
7D7E 116996   3685   LXI D,X
7D81 CD0078   3690   CALL PLOT
7D84 AF       3693   XRA A
7D85 E1       3694   POP H
7D86 C9       3695   RET
7D87          3697 * COMPUTE HORIZ
7D87          3698 * DIRECTION
7D87 2AE57F   3700 SUBH LHLD YOUP
7D8A 7C       3705   MOV A,H
7D8B 90       3710   SUB B
7D8C C8       3715   RZ
7D8D F2907D   3720   JP RT
7D90 05       3725   DCR B
7D91 C9       3730   RET
7D92          3740 * COMPUTE VERTICAL
7D92          3741 * DIRECTION
7D92 7D       3745 SUBV MOV A,L
7D93 91       3750   SUB C
7D94 C8       3755   RZ
7D95 F29A7D   3760   JP UP
7D98 0D       3765   DCR C
7D99 C9       3770   RET
7D9A          3795 * MOVE TABLE
7D9A 0C       3800 UP INR C
7D9B C9       3805   RET
7D9C 0D       3810 DN DCR C
7D9D C9       3815   RET
7D9E 05       3820 LF DCR B
7D9F C9       3825   RET
7DA0 04       3830 RT INR B
7DA1 C9       3835   RET
7DA2 0C       3840 UR INR C
7DA3 04       3845   INR B
7DA4 C9       3850   RET
7DA5 0C       3855 UL INR C
7DA6 05       3860   DCR B
7DA7 C9       3865   RET
7DA8 0D       3870 LL DCR C
7DA9 05       3875   DCR B
7DAA C9       3880   RET
7DAB 0D       3885 LR DCR C
7DAC 04       3890   INR B
7DAD C9       3895   RET
7DAE          3897 * YOUR DEATH
7DAE          3898 * DANCE
7DAE CD5B7D   3900 DEATH CALL FLASH
7DB1 11460A   3902   LXI D,FLOWER
7DB4 CD0078   3905   CALL PLOT
7DB7 01387F   3910   LXI B,M2
7DBA CD947E   3915   CALL MSGO
7DBD          3917 * OUTPUT SCORE
7DBD 3AB37F   3920   LDA RSCORE
7DC0 3C       3925   INR A
7DC1 32B37F   3930   STA RSCORE
7DC4 CD367E   3935   CALL OVER
7DC7 CDC57E   3937   CALL SCORE
7DCA CD8C7C   3941   CALL INP
```

```
7DCD            3942 * START OVER
7DCD C3287B     3945   JMP START
7DD0            3946 * RANDOM JUMP
7DD0 F5         3950 PANIC PUSH PSW
7DD1 3AB27F     3955   LDA DFST
7DD4 FE09       3960   CPI 9
7DD6 C2FE7D     3965   JNZ PAN3
7DD9 1E07       3970   MVI E,7
7DDB CD3C7C     3975   CALL RND
7DDE A3         3980   ANA E
7DDF FE01       3985   CPI 1
7DE1 CAE77D     3990   JZ PAN5
7DE4 C3237E     3995   JMP PAN4
7DE7 C5         4000 PAN5 PUSH B
7DE8 2AB77F     4005   LHLD ROB1
7DEB CD2D7E     4010   CALL PUTRB
7DEE 2AB97F     4015   LHLD ROB2
7DF1 CD2D7E     4020   CALL PUTRB
7DF4 2ABB7F     4025   LHLD ROB3
7DF7 CD2D7E     4030   CALL PUTRB
7DFA C1         4035   POP B
7DFB C3257E     4040   JMP PANN
7DFE 3AB27F     4045 PAN3 LDA DFST
7E01 DE03       4050   SBI 3
7E03 F2087E     4055   JP PAN1
7E06 1E01       4060   MVI E,1
7E08 C31A7E     4065   JMP PANOK
7E0B 3AB27F     4070 PAN1 LDA DFST
7E0E DE06       4075   SBI 6
7E10 1E03       4080   MVI E,3
7E12 F2187E     4085   JP PAN2
7E15 C31A7E     4090   JMP PANOK
7E18 1E07       4095 PAN2 MVI E,7
7E1A CD3C7C     4100 PANOK CALL RND
7E1D A3         4105   ANA E
7E1E FE01       4110   CPI 1
7E20 CA257E     4115   JZ PANN
7E23 F1         4120 PAN4 POP PSW
7E24 C9         4125   RET
7E25 CDFF7B     4130 PANN CALL YOU
7E28 F1         4135   POP PSW
7E29 F1         4140   POP PSW
7E2A C3657C     4145   JMP YMOVE
7E2D 111B57     4175 PUTRB LXI D,ROB
7E30 44         4180   MOV B,H
7E31 4D         4185   MOV C,L
7E32 CD0D78     4190   CALL PLOT
7E35 C9         4195   RET
7E36            4197 * HAS SCORE
7E36            4198 * REACHED 9
7E36 3AB37F     4200 OVER LDA RSCORE
7E39 FE39       4205   CPI 39H
7E3B CA537E     4210   JZ END1
7E3E 3AB47F     4215 OVER1 LDA YSCORE
7E41 FE39       4220   CPI 39H
7E43 C0         4225   RNZ
7E44            4227 * END OF GAME
7E44 CD8C7C     4230 ENDGAME CALL INP
7E47 016A7F     4240   LXI B,ENDB2
7E4A CD947E     4245   CALL MSGO
7E4D CDC57E     4250   CALL SCORE
7E50 C35F7E     4280   JMP END3
7E53 CD8C7C     4285 END1 CALL INP
7E56 01547F     4295   LXI B,ENDB1
7E59 CD947E     4300   CALL MSGO
7E5C CDC57E     4305   CALL SCORE
7E5F CD8C7C     4350 END3 CALL INP
7E62 C3007B     4355   JMP BEGIN
7E65            4370 * CHECK FOR ROBOT
7E65            4371 * IN SQUARE
7E65 44         4375 GETROB MOV B,H
7E66 4D         4380   MOV C,L
7E67 111B57     4385   LXI D,ROB
7E6A CD1378     4390   CALL COMP
7E6D C9         4395   RET
7E6E            4396 * ARE ALL ROBOTS
7E6E            4397 * DEAD
7E6E 2AB77F     4400 END LHLD ROB1
7E71 CD657E     4405   CALL GETROB
7E74 C8         4410   RZ
7E75 2AB97F     4415   LHLD ROB2
7E78 CD657E     4420   CALL GETROB
7E7B C8         4425   RZ
7E7C 2ABB7F     4430   LHLD ROB3
7E7F CD657E     4435   CALL GETROB
7E82 C8         4440   RZ
7E83 2ABD7F     4445   LHLD ROB4
7E86 CD657E     4450   CALL GETROB
7E89 C8         4451   RZ
7E8A 2ABF7F     4452   LHLD ROB5
7E8D CD657E     4455   CALL GETROB
7E90 C8         4460   RZ
7E91 C3AC7E     4465   JMP WIN
7E94            4495 * MESSAGE HANDLERS
7E94 0A         4500 MSGO LDAX B
7E95 FEFF       4525   CPI 0FFH
7E97 CA0380     4530   JZ CRLF
7E9A CD2A89     4535   CALL OUTA
7E9D 03         4540   INX B
7E9E C3947E     4545   JMP MSGO
7EA1 0A         4550 MSGO1 LDAX B
7EA2 FEFF       4555   CPI 0FFH
7EA4 C8         4560   RZ
7EA5 CD2A89     4565   CALL OUTA
7EA8 03         4570   INX B
7EA9 C3A17E     4575   JMP MSGO1
7EAC 016A7F     4700 WIN LXI B,ENDB2
7EAF CD947E     4705   CALL MSGO
7EB2 3AB47F     4710   LDA YSCORE
7EB5 3C         4715   INR A
7EB6 32B47F     4720   STA YSCORE
7EB9 CDC57E     4725   CALL SCORE
7EBC CD867E     4730   CALL OVER
7EBF CD8C7C     4740   CALL INP
7EC2 C3287B     4745   JMP START
7EC5            4747 * OUTPUT SCORE
7EC5 CD0380     4750 SCORE CALL CRLF
7EC8 CD0380     4765   CALL CRLF
7ECB CD0380     4770   CALL CRLF
7ECE 01917F     4780   LXI B,MSR
7ED1 CDA17E     4785   CALL MSGO1
7ED4 CD0380     4787   CALL CRLF
7ED7 01907F     4790   LXI B,MSR1
7EDA CDA17E     4795   CALL MSGO1
7EDD 3AB47F     4800   LDA YSCORE
7EE0 CD2A89     4810   CALL OUTA
7EE3 01A37F     4815   LXI B,MSR2
7EE6 CDA17E     4820   CALL MSGO1
7EE9 3AB37F     4825   LDA RSCORE
7EEC CD2A89     4835   CALL OUTA
7EEF CD0380     4837   CALL CRLF
7EF2 CD0380     4838   CALL CRLF
7EF5 01797F     4840   LXI B,ENDB3
7EF8 CD947E     4845   CALL MSGO
7EFB C9         4850   RET
7EFC 202A       8000 MNAME DW '* '
7EFE 202A       8005   DW '* '
7F00 202A       8010   DW '* '
7F02 2054       8015   DW 'T '
7F04 4845       8020   DW 'EH'
7F06 2047       8025   DW 'G '
7F08 4140       8030   DW 'MA'
7F0A 4520       8035   DW ' E'
7F0C 3046       8040   DW 'F0'
7F0E 2043       8045   DW 'C '
7F10 4841       8050   DW 'AH'
7F12 5345       8055   DW 'ES'
7F14 202A       8060   DW '* '
7F16 202A       8065   DW '* '
7F18 202A       8070   DW '* '
7F1A FF         8075   DB 0FFH
7F1B 204C       8100 MDIFF DW 'L '
7F1D 4556       8105   DW 'VE'
```

```
7F1F 454C      8110   DW 'LE'
7F21 204F      8115   DW 'O '
7F23 4620      8120   DW ' F'
7F25 4449      8125   DW 'ID'
7F27 4646      8130   DW 'FF'
7F29 4943      8135   DW 'CI'
7F2B 554C      8140   DW 'LU'
7F2D 5459      8145   DW 'YT'
7F2F 2028      8150   DW '(-'
7F31 302D      8155   DW '-0'
7F33 3929      8160   DW ')9'
7F35 3F2D      8170   DW '-?'
7F37 FF        8175   DB 0FFH
7F38 2059      8200 M2 DW 'Y '
7F3A 4F55      8205   DW 'UO'
7F3C 2048      8210   DW 'H '
7F3E 4156      8215   DW 'VA'
7F40 4520      8220   DW ' E'
7F42 4245      8225   DW 'EB'
7F44 454E      8230   DW 'NE'
7F46 2044      8235   DW 'D '
7F48 4553      8240   DW 'SE'
7F4A 5452      8245   DW 'RT'
7F4C 4F59      8250   DW 'YO'
7F4E 4514      8255   DW 'DE'
7F50 2120      8260   DW ' !'
7F52 00FF      8275   DW 0FF00H
7F54 5448      8300 ENDB1 DW 'HT'
7F56 4520      8305   DW ' E'
7F58 524F      8310   DW 'OR'
7F5A 4230      8315   DW '0B'
7F5C 5453      8320   DW 'ST'
7F5E 2048      8325   DW 'H '
7F60 4156      8330   DW 'VA'
7F62 4520      8335   DW ' E'
7F64 574F      8340   DW 'OW'
7F66 4E21      8345   DW '!N'
7F68 00FF      8360   DW 0FF00H
7F6A 5930      8400 ENDB2 DW '0Y'
7F6C 5520      8405   DW ' U'
7F6E 4841      8410   DW 'AH'
7F70 5645      8415   DW 'EV'
7F72 2057      8420   DW 'W '
7F74 4F4E      8425   DW 'NO'
7F76 2120      8430   DW ' !'
7F78 FF        8450   DB 0FFH
7F79 5052      8500 ENDB3 DW 'RP'
7F7B 4553      8505   DW 'SE'
7F7D 5320      8510   DW ' S'
7F7F 414E      8515   DW 'NA'
7F81 5920      8520   DW ' Y'
7F83 4B45      8525   DW 'EK'
7F85 5920      8530   DW ' Y'
7F87 544F      8535   DW 'OT'
7F89 2050      8540   DW 'P '
7F8B 4C41      8545   DW 'AL'
7F8D 5920      8550   DW ' Y'
7F8F 00FF      8565   DW 0FF00H
7F91 2D2D      8600 MSR DW '--'
7F93 5343      8605   DW 'CS'
7F95 4F52      8610   DW 'RO'
7F97 452D      8615   DW '-E'
7F99 2D20      8620   DW ' -'
7F9B 00FF      8635   DW 0FF00H
7F9D 5930      8650 MSR1 DW '0Y'
7F9F 553A      8655   DW ':U'
7FA1 20FF      8665   DW 0FF20H
7FA3 2020      8700 MSR2 DW '  '
7FA5 524F      8702   DW 'OR'
7FA7 4230      8705   DW '0B'
7FA9 5453      8710   DW 'ST'
7FAB 3A20      8715   DW ' :'
7FAD FF        8725   DB 0FFH
0006           9000 SP EQU 6
0006           9001 PSW EQU 6
8000           9010 MONITO EQU 8000H
000F           9015 HEIGHT EQU 0FH
000F           9020 WIDTH EQU 0FH
1248           9025 SPACE EQU 1248H
9669           9030 X EQU 9669H
6996           9032 RID EQU 6996H
571B           9033 ROB EQU 571BH
FFFF           9034 RIB EQU 0FFFFH
464E           9035 YOUS EQU 464EH
000E           9040 DAZ1 EQU 0EH
000F           9045 DAZ2 EQU 0FH
007F           9050 CONF EQU 7FH
0004           9055 LOWD EQU 4
001C           9060 HID EQU 28
0004           9065 LOWE EQU 4
001C           9070 HIE EQU 28
87FF           9075 STACK EQU 87FFH
892A           9080 OUTA EQU 892AH
000A           9090 LINFD EQU 0AH
0000           9095 STAT EQU 0
0080           9100 RDA EQU 80H
0001           9105 DATA EQU 1
007E           9110 ESC EQU 7EH
0A46           9125 FLOWER EQU 0A46H
780D           9140 PLOT EQU 780DH
7810           9145 GET EQU 7810H
7813           9150 COMP EQU 7813H
0003           9155 CRLF EQU 0003H
7819           9160 CLEAR EQU 7819H
7FAE 3412      9500 SH DW 1234H
7FB0 7856      9505    DW 5678H
7FB2           9510 DFST DS 1
7FB3           9515 RSCORE DS 1
7FB4           9517 YSCORE DS 1
7FB5           9520 YOUP DS 2
7FB7           9525 ROB1 DS 2
7FB9           9530 ROB2 DS 2
7FBB           9535 ROB3 DS 2
7FBD           9540 ROB4 DS 2
7FBF           9545 ROB5 DS 2
```

```
DD 7B00 7FB1

7B00  3E 30 32 B4 7F 32 B3 7F
7B08  31 FF 87 CD 03 80 01 FC
7B10  7E CD 94 7E CD 03 80 01
7B18  1B 7F CD A1 7E CD BC 7C
7B20  CD 2A 89 E6 0F 32 B2 7F
7B28  CD 19 78 31 FF 87 CD 03
7B30  80 3E 7F D3 0F 01 00 00
7B38  3E 0F F5 11 69 96 CD 0D
7B40  78 11 48 12 04 CD 0D 78
7B48  F1 04 3D C2 3A 7B 3E 0F
7B50  F5 11 69 96 CD 0D 78 11
7B58  48 12 0C CD 0D 78 F1 0C
7B60  3D C2 50 7B 3E 0F F5 11
7B68  69 96 CD 0D 78 11 48 12
7B70  05 CD 0D 78 F1 05 3D C2
7B78  66 7B 3E 0F F5 11 69 96
7B80  CD 0D 78 11 48 12 0D CD
7B88  0D 78 F1 0D 3D C2 7C 7B
7B90  3A B2 7F 07 47 3E 1E 90
7B98  57 CD 3C 7C E6 1F 47 CD
7BA0  3C 7C E6 1F 4F CD 26 7C
7BA8  DA 99 7B D5 11 69 96 CD
7BB0  0D 78 D1 15 C2 99 7B CD
7BB8  DB 7B 22 B7 7F CD DB 7B
7BC0  22 B9 7F CD DB 7B 22 BB
7BC8  7F CD DB 7B 22 BD 7F CD
7BD0  DB 7B 22 BF 7F CD FF 7B
7BD8  C3 65 7C CD 3C 7C E6 1F
7BE0  47 CD 3C 7C E6 1F 4F CD
7BE8  26 7C DA DB 7B 11 00 00
7BF0  CD 13 78 C2 DB 7B 11 1B
7BF8  57 CD 0D 78 60 69 C9 CD
7C00  3C 7C E6 1F 47 CD 3C 7C
7C08  E6 1F 4F CD 26 7C DA FF
7C10  7B 11 00 00 CD 13 78 C2
7C18  FF 7B 11 4E 46 CD 0D 78
7C20  60 69 22 B5 7F C9 78 DE
7C28  04 37 F8 78 DE 1C 37 F0
7C30  79 DE 04 37 F8 79 DE 1C
7C38  37 F0 3F C9 C5 D5 E5 21
7C40  B1 7F 06 08 7E 07 07 07
7C48  AE 17 17 2D 2D 2D 7E 17
7C50  77 2C 7E 17 77 2C 7E 17
7C58  77 2C 7E 17 77 05 C2 45
7C60  7C E1 D1 C1 C9 CD BC 7C
7C68  F5 2A B5 7F 44 4D 11 00
7C70  00 CD 0D 78 F1 FE 4A CC
7C78  A0 7D FE 55 CC A2 7D FE
7C80  59 CC 9A 7D FE 54 CC A5

7C88  7D FE 47 CC 9E 7D FE 42
7C90  CC A8 7D FE 4E CC 9C 7D
7C98  FE 4D CC AB 7D FE 50 CC
7CA0  D0 7D FE 48 CA B0 7C 11
7CA8  00 00 CD 13 78 C2 AE 7D
7CB0  CD 1A 7C CD D2 7C CD 6E
7CB8  7E C3 65 7C DB 00 E6 80
7CC0  C2 BC 7C D8 01 E6 7F FE
7CC8  0D CA 06 78 FE 7E CA 00
7CD0  80 C9 2A B7 7F CD 1C 7D
7CD8  DA E1 7C CD 28 7D 22 B7
7CE0  7F 2A B9 7F CD 1C 7D DA
7CE8  F0 7C CD 28 7D 22 B9 7F
7CF0  2A BB 7F CD 1C 7D DA FF
7CF8  7C CD 28 7D 22 BB 7F 2A
7D00  BD 7F CD 1C 7D DA 0E 7D
7D08  CD 28 7D 22 BD 7F 2A BF
7D10  7F CD 1C 7D D8 CD 28 7D
7D18  22 BF 7F C9 44 4D 11 1B
7D20  57 CD 13 78 37 C8 3F C9
7D28  11 03 00 CD 0D 78 CD 87
7D30  7D CD 92 7D 11 4E 46 CD
7D38  13 78 CA AE 7D 11 69 96
7D40  CD 13 78 CA 5B 7D 11 1B
7D48  57 CD 13 78 CC 5B 7D 11
7D50  48 12 CD 13 78 CA 5B 7D
7D58  C3 F6 7B E5 26 A0 11 1B
7D60  57 E5 CD 0D 78 E1 2E FF
7D68  2D C2 68 7D 2E FF 11 96
7D70  69 E5 CD 0D 78 E1 20 C2
7D78  76 7D 25 C2 5E 7D 11 69
7D80  96 CD 0D 78 AF E1 C9 2A
7D88  B5 7F 7C 90 C8 F2 A0 7D
7D90  05 C9 7D 91 C8 F2 9A 7D
7D98  0D C9 0C C9 0D C9 05 C9
7DA0  04 C9 0C 04 C9 0C 05 C9
7DA8  0D 05 C9 0D 04 C9 CD 5B
7DB0  7D 11 46 0A CD 0D 78 01
7DB8  38 7F CD 94 7E 3A B3 7F
7DC0  3C 32 B3 7F CD 36 7E CD
7DC8  C5 7E CD BC 7C C3 28 7B
7DD0  F5 3A B2 7F FE 09 C2 FE
7DD8  7D 1E 07 CD 3C 7C A3 FE
7DE0  01 CA E7 7D C3 23 7E C5
7DE8  2A B7 7F CD 2D 7E 2A B9
7DF0  7F CD 2D 7E 2A BB 7F CD
7DF8  2D 7E C1 C3 25 7E 3A B2
7E00  7F DE 03 F2 0B 7E 1E 01
7E08  C3 1A 7E 3A B2 7F DE 06
7E10  1E 03 F2 18 7E C3 1A 7E
7E18  1E 07 CD 3C 7C A3 FE 01

7E20  CA 25 7E F1 C9 CD FF 7B
7E28  F1 F1 C3 65 7C 11 1B 57
7E30  44 4D CD 00 78 C9 3A B3
7E38  7F FE 39 CA 53 7E 3A B4
7E40  7F FE 39 D6 CD BC 7C 01
7E48  6A 7F CD 94 7E CD C5 7E
7E50  C3 5F 7E CD BC 7C 01 54
7E58  7F CD 94 7E CD C5 7E CD
7E60  BC 7C C3 00 7B 44 4D 11
7E68  1B 57 CD 13 78 C9 2A B7
7E70  7F CD 65 7E C8 2A B9 7F
7E78  CD 65 7E C8 2A BB 7F CD
7E80  65 7E C8 2A BD 7F CD 65
7E88  7E C8 2A BF 7F CD 65 7E
7E90  C8 C3 AC 7E 0A FE FF CA
7E98  03 80 CD 2A 89 03 C3 94
7EA0  7E 0A FE FF C8 CD 2A 89
7EA8  03 C3 A1 7E 01 6A 7F CD
7EB0  94 7E 3A B4 7F 3C 32 B4
7EB8  7F CD C5 7E CD 36 7E CD
7EC0  BC 7C C3 28 7B CD 03 80
7EC8  CD 03 80 CD 03 80 01 91
7ED0  7F CD A1 7E CD 03 80 01
7ED8  90 7F CD A1 7E 3A B4 7F
7EE0  CD 2A 89 01 A3 7F CD A1
7EE8  7E 3A B3 7F CD 2A 89 CD
7EF0  03 80 CD 03 80 01 79 7F
7EF8  CD 94 7E C9 20 2A 20 2A
7F00  20 2A 20 54 48 45 20 47
7F08  41 4D 45 20 30 46 20 43
7F10  48 41 53 45 20 2A 20 2A
7F18  20 2A FF 20 4C 45 56 45
7F20  4C 20 4F 46 20 44 49 46
7F28  46 49 43 55 4C 54 59 2D
7F30  28 30 2D 39 29 3F 2D FF
7F38  20 59 4F 55 20 48 41 56
7F40  45 20 42 45 45 4E 20 44
7F48  45 53 54 52 4F 59 45 44
7F50  21 20 00 FF 54 48 45 20
7F58  52 4F 42 30 54 53 20 48
7F60  41 56 45 20 57 4F 4E 21
7F68  00 FF 59 30 55 20 48 41
7F70  56 45 20 57 4F 4E 21 20
7F78  FF 50 52 45 53 53 20 41
7F80  4E 59 20 48 45 59 20 54
7F88  4F 20 50 4C 41 59 20 00
7F90  FF 2D 20 53 43 4F 52 45
7F98  2D 2D 20 00 FF 59 30 55
7FA0  3A 20 FF 20 20 52 4F 42
7FA8  30 54 53 3A 20 FF 34 12
7FB0  78 56
```

Z-80 RAM TESTER

Dear Dr. Dobb's:

In the August issue of *DDJ* you carried an article by Jim Butterfield which listed a RAM test program for the 6502. This article in turn referenced an algorithm described by Knaizuk and Hartmann in the *IEEE Transactions on Computers* (April 1977). Enclosed for your perusal is an assembly listing for the Z-80 version of this program. Some frills have been added for those lucky people who have the TDL "ZAPPLE" Monitor. These allow the user to specify the start and stop addresses and print out the location of the error byte.

I have found this test program to be exceptionally good as well as fast. It even found a pattern sensitive 2102 that I had saved for the occasion from a previous hair tearing session. Keep up the good work!

John MacDougall

110 North Gordon Way
Los Altos CA 94022

Received: 77 Dec 7

```
                .TITLE 'MEMORY TEST PROGRAM FOR Z-80'
                .RADIX 16
                ;This is a program written in TDL assembler
                ;which tests memory by writing one's in every
                ;location and then writing 00's in every third
                ;location and checking the result.  The test
                ;is repeated with the 00's pos'n shifted one
                ;to the left until all locations have been
                ;covered.  At this time  the roles of one's
                ;and zero's are reversed and the test repeated
                ;again.  The test is fairly thorough but
                ;takes only about 20 seconds for 20K bytes.
                ;Ref: DDJ Aug 1977-- John MacDougall Nov 19 1977
0000' 0E02              MVI     C,02    ;SET FOR TWO PARAMS
0002' CD F4FF           CALL    0F4FF   ;MON PROG TO GET TWO PARAMS
0005' 22 002E'          SHLD    START   ;STORE START
0008' ED53 0030'        SDED    END     ;STORE END
000C' 3E02              MVI     A,02    ;TWO FF'S BEFORE 00
000E' 32 002D'  NEXT:   STA     PODD    ;STORE POS'N OF ODD BYTE
0011' 3EFF              MVI     A,0FF   ;MAJOR BYTE
0013' 4F        GO:     MOV     C,A     ;SET WORKING REGISTER
0014' CD 0032'          CALL    STUFF   ;STUFF MEM BLOCK WITH BYTES
0017' CD 003B'          CALL    STUFF3  ;STUFF WITH MINOR BYTES
001A' CD 004F'          CALL    CHECK   ;CHECK IT AND PRINT ERRORS
001D' 79                MOV     A,C     ;WHERE ARE WE?
001E' 2F                CMA             ;COMPL A
001F' B7                ORA     A       ;ZERO?
0020' 28F1              JRZ     GO      ;YES, NOW DO WITH MAJOR 0'S
0022' 3A 002D'          LDA     PODD    ;GET POS'N BYTE
0025' FE00              CPI     00      ;ARE WE FINISHED?
0027' CA F000           JZ      0F000   ;YES, BACK TO MONITOR
002A' 3D                DCR     A       ;MOVE ONE DOWN
002B' 18E1              JMPR    NEXT    ;TO NEXT PASS WIT NEW POS'N
                ;Rather than use a fixed location
                ;for the start and stop addresses the following
                ;locations in the program are reserved.  The
                ;program is therefore not ROMable.
002D'           PODD:                   ;RESERVED FOR POS'N BYTE
002D'           .BLKB 1
002E'           START:                  ;RESERVED FOR START ADDR
002E'           .BLKB 2
0030'           END:                    ;RESERVED FOR STOP ADDR
0030'           .BLKB 2
                ;This subroutine stuffs the majority byte
0032' CD 006C'  STUFF:  CALL    STASTO  ;GET START/STOP ADDR +POSN
0035' 71        DOIT:   MOV     M,C     ;STUFF MAJOR BYTE ALL OVER
0036' CD F561           CALL    0F561   ;CALL MONITOR ENDCHECK
0039' 18FA              JMPR    DOIT    ;KEEP ON STUFFING
                ;This subroutine stuffs the minority byte
003B' CD 006C'  STUFF3: CALL    STASTO  ;GET START/STOP ADDR + POSN
003E' 47                MOV     B,A     ;IS MINOR BYTE HOME?
003F' FE00              CPI     00      ;WELL?
0041' 2005              JRNZ    HIL     ;CONTINUE
0043' 79        MINOR:  MOV     A,C     ;PREPARE COMPLEMENT
0044' 2F                CMA             ;DO IT
0045' 77                MOV     M,A     ;STUFF IT
0046' 0603              MVI     B,03    ;SET COUNT TO DO AGAIN
0048' CD F561   HIL:    CALL    0F561   ;MONITOR ENDCHECK
004B' 10FB              DJNZ    HIL     ;KEEP  MAJOR BYTE
004D' 18F4              JMPR    MINOR   ;DO THE MINOR BYTE
                ;This routine checks all the locations at once
004F' CD 006C'  CHECK:  CALL    STASTO  ;GET START/STOP ADDR+POSN
0052' 47                MOV     B,A     ;READY FOR TESTING
0053' FE00              CPI     00      ;=0?
0055' 2007              JRNZ    MINOD   ;TO CHECK MAJOR BYTE
0057' 79        MINOC:  MOV     A,C     ;TO CHECK COMPLEMENT
0058' 2F                CMA             ;COMPLA
0059' BE                CMP     M       ;COMPARE WITH MEMORY BYTE
005A' 0603              MVI     B,03    ;SET COUNT FOR  NEXT  MINOR
005C' 1802              JMPR    ER      ;TO PRINT ERR IF NECESSARY
005E' 79        MINOD:  MOV     A,C     ;FOR MAJOR BYTE
005F' BE                CMP     M       ;COMPARE WITH MEMORY
0060' C5        ER:     PUSH    B       ;SAVE THE BC REGISTER
0061' C4 F470           CNZ     0F470   ;MONITOR TO PRINT ADDRESS
0064' C1                POP     B       ;RECOVER B
0065' CD F561           CALL    0F561   ;CALL ENDCHECK IN MONITOR
0068' 10F4              DJNZ    MINOD   ;KEEP GOING ON MAJOR BYTE
006A' 18EB              JMPR    MINOC   ;NOW TO MINOR BYTE
                ;Subroutine for reloading STOP/START & Posn info
006C' 3A 002D'  STASTO: LDA     PODD    ;LOAD POSN OF MINOR BYTE
006F' 2A 002E'          LHLD    START   ;GET START ADDR
0072' ED5B 0030'        LDED    END     ;GET END ADDR
0076' C9                RET             ;RETURN
                .END
```

```
>R
Typical printout when testing a pattern sensitive 2102!
>GO
4000 4FFF

4EF0
4EF0
4EF0

>R
Complete hex listing--
>DO 76

0000  0E 02 CD FF F4 22 2E 00 ED 53 30 00 3E 02 32 2D
0010  00 3E FF 4F CD 32 00 CD 3B 00 CD 4F 00 79 2F B7
0020  28 F1 3A 2D 00 FE 00 CA 00 F0 3D 18 E1 FF FF FF
0030  FF FF CD 6C 00 71 CD 61 F5 18 FA CD 6C 00 47 FE
0040  00 20 05 79 2F 77 06 03 CD 61 F5 10 FB 18 F4 CD
0050  6C 00 47 FE 00 20 07 79 2F BE 06 03 16 02 79 BE
0060  C5 C4 70 F4 C1 CD 61 F5 10 F4 18 EB 3A 2D 00 2A
0070  2E 00 ED 5B 30 00 C9
>
```

```
+++++ SYMBOL TABLE +++++

CHECK  004F'    DOIT   0035'    END    0030'    ER     0060'
O      0013'    HIL    0048'    MINOC  0057'    MINOD  005E'
MINOR  0043'    NEXT   000E'    PODD   002D'    START  002E'
STASTO 006C'    STUFF  0032'    STUFF3 003B'
```

MINI-EDITOR FOR 8080 PILOT

Tim Scully is a friend and business associate who is an inmate at a Federal penitentiary. Despite that, he's busier and more productive than most prople I know. Tim has been designing biofeedback equipment and doing biofeedback research for many years. He is now working on a doctorate in psychology; his dissertation project involves researching and developing biofeedback systems and techniques for use in drug rehabilitation. He is also publishing articles on biofeedback and microcomputers in People's Computers *and, as you see here, writing articles in other areas as well.*

With permission from prison officials, Tim has begun developing a microcomputer-based system which will allow a cerebral palsy patient to communicate. A knee switch on the wheel-chair system will enable the patient to construct messages which will be displayed on a CRT. He plans to make his work available to the public.

In addition, Tim is teaching a computer class to fellow inmates, using the computer system belonging to the cerebral palsy patient. Tim is hopeful of eventually obtaining a computer system for the prison, despite the fact that prison resources are scarce and he cannot solicit donations. But while donations can't be solicited *they can be* accepted *so long as no strings are attached. Contact me for more information if you have equipment to donate to Tim.*

Phyllis Cole
Editor, People's Computers
Box E
Menlo Park, CA 94025

Tim Scully *Received: 77 Oct 23*

I have a Poly System 4 with 8K of RAM (from 2000H–3FFFH). A few weeks ago I decided to get PILOT running on this system, and translated Dr. John A. Starkweather's 8080 PILOT 1.1 (from May 1977 *Dr. Dobb's*) so that it would live comfortably in Poly's memory, starting at 2000H. After writing a few simple PILOT programs, the need for an editor was clear. Being short on memory, I didn't have room for an elaborate editor program, I wanted to squeeze a mini-editor into the space between a jump to the start of PILOT at 2000H and the actual start of PILOT at 2100H.

The program listed below was hand assembled because Poly has neither an assembler nor a printer. The program uses several subroutines from the Polymorphics 4.0 monitor ROM, and these are the only labeled calls in the program listing. If you have another hardware configuration, you will want to replace these calls with some of your own. Here's a brief description of what each one does, in case you don't have the Poly monitor.

WHO – fetches a character from the keyboard and returns it in A.

WH1 – prints the character in A on the TV screen, recognizes a carriage return as carriage return/line feed, recognizes CTL Z as a jump to monitor command (so I modified PILOT to use CTL B where Dr. Starkweather used CTL Z), and recognize CTL L as a command to erase a line (so I modified PILOT to use CTL L where Dr. Starkweather used CTL U).

DEOUT – prints the contents of DE as a four digit hex number.

MOVE – moves – BC bytes from an area starting at (HL) to the area starting at (DE) . . . only useful for moving from higher addresses to lower addresses.

The mini-editor displays one line of your PILOT program at a time, beginning with the first line. At the beginning of each line it displays the memory address (in hex) of the first character in the line (you can easily delete this feature if you don't need it). The mini-editor is designed to drive a video display,

3-WAY BASIC MARRIAGE

Received: 77 Nov 18

I have been most pleasantly surprised by *DDJCC&O* after many rip offs and disappointments (i.e. Logic Press). I would gladly pay twice your subscription rates. You are truly doing an outstanding job of providing a needed service for the private computer user.

I have adapted the LLL BASIC which was printed in *DDJCC&O* for my home made 8080 system and of course the first thing off I want more goodies. It seems to me that the Extended Accuracy Arithmetic package by Dinely in Vol. 2 No. 3 of *DDJCC&O* would be a useful addition to the BASIC interpreter. If there is any one else of like mind, I'd appreciate the chance to correspond with them. Another major feature which I am now adapting for my system is Dr. Starkweather's 8080 PILOT. I plan to combine the LLL BASIC with PILOT using the source line editor in LLL BASIC to edit PILOT statements. Dr. Starkweather's 8080 code for PILOT is beautifullly organized and commented and his "Guide to Pilot 8080" is also an exemplary job.

If you are interested, I can send a souce listing of my results of the marriage of those 3 contributions from *DDJCC&O.*

Yours Truly,

B. Reeves

2-18 Patpong Road
Bangkok 5
Thailand

Many thanks for the compliments! And yes indeed to that offer. We'd be most pleased... sounds like a "marriage made in Heaven"–TRW

but could easily work with a printer if you don't mind using up some paper.

Because of the small memory space available in my system for this editor, it does not use a cursor to indicate where insertions or deletions are to be made. Instead, if insertion is requested, a character will be inserted just before the first character presently being displayed. If deletion (rubout) is requested, the first character on the display will be deleted. The spacing feature of the program allows part of a line to be displayed so that insertions and deletions can be made in the middle of a line. When first called, the editor will display the first line of your program. The editor commands are listed below.

Carriage return – displays the next line

Line feed – backs up the display the previous line if you are looking at a complete line. If you are looking at part of a line, it backs you up to the beginning and shows you all of it (useful for seeing the results of insertions/deletions).

Space – skips the first character of the previously displayed line and displays the remainder of the line, useful for editing in the middle of a line.

CTL B – returns to PILOT if you called the mini-editor from PILOT.

CTL D – inserts a space just in front of the first character on the display.

CTL E – inserts a carriage return just in front of the first character presently on display.

Any other control code – will be ignored (except CTL Z if you are using the Poly monitor ROM).

Any other non-control character – will be inserted just in front of the first character presently on display.

RUBOUT – deletes the first character on display.

```
203E  23        INX H       ;move ahead one character
203F  C31820    JMP 2018H   ;display the remaining line
2042  FE7F      CPI 7FH     ;check for 'rubout'
2044  C26220    JNZ 2062H   ;if not rubout, keep looking
2047  2A0E20    LHLD 200EH  ;fetch starting address
204A  E5        PUSH H      ;copy HL
204B  D1        POP D       ;into DE
204C  010000    LXI B,0     ;initialize BC for use as counter
204F  3E01      MVI A,1     ;set up to check for end-of-program flag
2051  2B        DCX H
2052  23        INX H
2053  0B        DCX B       ;count one character
2054  BE        CMP M       ;check for end flag
2055  C25220    JNZ 2052H   ;if not end, keep counting
2058  2A0E20    LHLD 200EH  ;fetch starting address
205B  23        INX H       ;skip the character to be rubbed out
205C  CD0901    CALL MOVE   ;move the rest of the program down one byte
205F  C31B20    JMP 201BH   ;display remainder of line (after rubout)
2062  FE0D      CPI 0DH     ;check for carriage return
2064  CA1820    JZ 2018H    ;carriage return displays next line
2067  FE0A      CPI 0AH     ;check for line feed
2069  C27B20    JNZ 207BH   ;if not, keep looking
206C  2A0E20    LHLD 200EH  ;fetch start of line
206F  2B        DCX H       ;line feed backs up a line
2070  2B        DCX H       ;back up
2071  7E        MOV A,M     ;fetch character
2072  FE0D      CPI 0DH     ;look for carriage return
2074  C27020    JNZ 2070H   ;and keep backing up until you find one
2077  23        INX H       ;move ahead to start of line
2078  C31820    JMP 2018H   ;and display it
207B  FE02      CPI 2       ;look for CTL B
207D  C8        RZ          ;CTL B returns us to PILOT
207E  FE04      CPI 4       ;check for CTL D
2080  CAB320    JZ 20B3H    ;CTL D inserts a space
2083  FE05      CPI 5       ;check for CTL E
2085  CAB820    JZ 20B8H    ;CTL E inserts a carriage return
2088  FE1F      CPI 1FH     ;skip any other control codes
208A  DA3320    JC 2033H    ;by fetching another keypress
208D  2A0E20    LHLD 200EH  ;fetch starting address
2090  F5        PUSH PSW    ;otherwise, save character so we can insert it
2091  3E01      MVI A,1     ;set up to look for end-of-program flag
2093  010000    LXI B,0     ;initialize BC for use as counter
2096  2B        DCX H
2097  23        INX H
2098  0B        DCX B       ;count one character
2099  BE        CMP M       ;check for end flag
209A  C29720    JNZ 2097H   ;if not end, keep counting
209D  54        MOV D,H
209E  5D        MOV E,L     ;copy HL into DE
209F  13        INX D       ; move program
20A0  7E        MOV A,M     ; up one character
20A1  12        STAX D      ; to make room for
20A2  1B        DCX D       ; insertion
20A3  2B        DCX H       ;
20A4  0C        INR C       ;count one character
20A5  C29D20    JNZ 209DH   ;keep on if not last character
20A8  04        INR B
20A9  C29D20    JNZ 209DH
20AC  F1        POP PSW     ;fetch back the character
20AD  12        STAX D      ;and insert it
20AE  23        INX H
20AF  23        INX H       ;space ahead
20B0  C31820    JMP 2018H   ;and display the remaining line
20B3  3E20      MVI A,' '   ;insert space
20B5  C38D20    JMP 208DH
20B8  3E0D      MVI A,0DH   ;insert carriage return
20BA  C38D20    JMP 208DH
```

listing for mini-editor

```
2010  210030    LXI H,3000H ;start of program storage
2013  3E0C      MVI A,0CH
2015  CD240C    CALL WH1    ;clear TV screen
2018  220E20    SHLD 200EH  ;store starting address
201B  2A0E20    LHLD 200EH  ;fetch starting address
201E  EB        XCHG        ;move address to DE
201F  CDD103    CALL DEOUT  ;print DE as 4 hex digits
2022  EB        XCHG        ;restore address to HL
2023  3E20      MVI A,' '
2025  CD240C    CALL WH1    ;print a space
2028  7E        MOV A,M     ;fetch character from memory
2029  CD240C    CALL WH1    ;print it
202C  7E        MOV A,M     ;fetch it again
202D  23        INX H       ;set up for next character
202E  FE0D      CPI 0DH     ;check for carriage return = end of line
2030  C22820    JNZ 2028H   ;if not end of line, print more
2033  CD2C0C    CALL WHO    ;fetch keyboard input
2036  FE20      CPI ' '     ;check for space
2038  C24220    JNZ 2042H   ;if not, keep looking
203B  2A0E20    LHLD 200EH  ;fetch starting address
```

THE LAST REHASH OF BEAU HEX?

Some letters on the subject

Dear Dr. Dobb: *Received: 78 Jan 24*

I have been noticing for some time, a gradual trend toward pseudo-religious fanaticism in many phases of computer related activity. We are condemned to everlasting failure if we don't organize our programs like bands of angels. Some other fate awaits if we don't throw away our flowchart templates and take up weiner diagrams, whatever they are.

Those of us who know from long experience, that older and other methods work well when employed by competent people, have tended to dismiss these cults as the products of people who sought to cure their basic inability to organize their work logically as they went along, and then found that they could make (lots of) money by selling their ideas to the ignorant. We recognized them as rather childish simplifications of the situation, shrugged our shoulders and went on about our business.

The letter from Mark J. Nitzberg, in issue 2/9, reveals that this may have been a mistake. We have a young person of unknown experience level, who raves over the virtues of hexidecimal notation, and condemns octal, both with religious fervor, and with arguments which have nothing to do with the problem at hand.

I think his letter has gotten me in a violent enough mood to respond in kind, and I have prepared a counter argument in the form of an open letter to Mark:

James C. Matthews

Dear Mark,

I read your letter endorsing the use of hex. It seems that you are quite comfortable with this notation. Probably you can even do hex arithmetic in your head. However, there are two errors in your assumption that we must all abandon octal or suffer some everlasting (but undefined) torment: 1. The fact that you find hex easy to work with does not mean that all of us are equally able to manipulate large-base numbers, and; 2. You don't understand the purpose behind the introduction of either the octal or hex formats. I shall try to explain these points, along with some other areas about which you seem a little fuzzy.

First off, storage space, and the waste or conservation thereof, has nothing to do with either form of notation. Very little, if any, data is ever stored in either. The beast can count neither to 8 nor to 16, but only to 2. Everything is stored and acted upon in binary. If we really wanted to save paper and keystrokes, the thing to do is develop a printable set of 256 distinct characters. Then a simple 8-key pad would suffice for data entry. (Don't laugh—many users of ARPANET's DNLS have become surprisingly proficient at entering ASCII data with the mouse buttons and fingerboard. They use the normal keyboard for initial entry, but only for very long changes while editing.) Of course, we would need to memorize the entire set of 256 characters, and do the binary-to-octabinary (seems like a good name for our new notation, we can say OB for short) in our heads, but then that's a small price to pay for all the paper we would save.

Octal appeared very early on the computer scene. The human brain seems to have something against long strings of repetitive characters. Binary, having only 2 character variations, reaches the rejection point very quickly. The limit seems to be about 6 bits, because most people who work with Baudot teletype soon learn to read the 5-unit code from punched tape, but very few ever learn more than a few characters of 7-unit ASCII. (I do know this may seem to go against what I said about the DNLS users, but in that case it is broken into a 4-unit string for the left hand and a 3-unit for the right.) In any event, when memories were slow, and word lengths were growing by leaps and bounds to minimize the number of required accesses, people soon found that they could not deal effectively in the language the Machine was compelled to speak. People were used to dealing in decimal representation of numbers, and octal was the nearest subset of decimal that used an even binary set. Octal, therefore is purely a matter of convenience for the *human* in dealing with the blinkey beast.

The development of hex is a compromise. Since most applications of computers require some nibbling at, if not actual crunching of decimal numbers, most machines become proficient, via hardware instructions, at either XS3 or BCD arithmetic. (You mention the 8080's DAA as though it had something to do with hex. It doesn't, it's for BCD.) The great sages in their ivory towers, who spew forth monitors and operating systems, discovered that ASCII is effectively a superset of BCD, and they could save a few bytes of code in their creations by dragging the humans across the boundary of decimal to something easier to do on latter-day machines. And what it was, was hexidecimal. Hex was rather well accepted for several reasons: a. It came with the machine. If you wanted to talk to it, you learned hex. b. Doing arithmetic with alpha characters mixed freely with numbers enhances the mystique of the profession. While most octal users, in my experience, do simple arithmetic in their heads, most hex users seem to do their's on paper. This is almost certain to awe the onlookers.

You are totally wrong about any advantage of hex in memorizing the 8080 instruction set. In hex, there is no visible relationship of families of instructions, and you must memorize the entire set of over 200 instructions separately. In octal, the relationships are easily seen, and there are only about 20 basic instructions and variant rules to memorize. This is a clear 10-to-1 advantage for octal in that most precious of all storage units, your one and only brain.

You also seem to fail to understand the logic behind the "amazingly unorthodox" split octal representation used when 8-bit machines have 16-bit addressing capability. This again is simply a concession to the human, allowing him to see the contents of each of the registers holding the 16-bit number, separately, as they really exist internally, and as they must be acted upon, in most cases.

And finally, some philosophy. You did not *have* to compile that letter. Most people who become addicted to bit-flipping will be exposed to all manner of number representations, and hopefully, will be able to choose one that is right for them. I don't have to write this, but I am a technical atheist, and take pleasure in espousing things looked down upon by those who allow some religious fervor to override their logic and condemn things without good reason. I don't believe that any Supreme Being gives a hang how we structure our programs, draw our flowcharts, or represent our numbers. God, and the religious orders which help to channel some of our energies in search of truth and goodness, have far greater concerns in getting us to live with one and another, and with and upon this poor old Earth. You are wasting the energy and enthusiasm of your youth in pursuit of some rather nebulous dragons. Please try to find a channel that will have more lasting and meaningful results than trying to teach people to count on all 16 fingers when they have only ten.

Very sincerely,
James C. Matthews

2028 Merrily Dr.
Montgomery, AL 36111

Dear Sir, *Received: 78 Jan 16*

I am new to computing, so was unaware of the octal/hex controversy. Needless to say, I prefer octal, since there is less to learn. What hex proponent can state that hex never led him into error?

(more)

SWTPC CT-1024 AUTOMATIC SCROLL

Dear Mr. Warren, *Received 77 Dec 12*

Having implemented the modifications suggested by Mr. R. Lynn Smith in the February 77 issue (pp. 25-27), I found it desirable to keep full control over the cursor (using the SWTPC Cursor Control board and Screen Read board) rather than have a cursor stuck at the last line of the screen.

This can be achieved in a very simple fashion, as described next. The purpose is keeping all standard behaviour and all options normally functioning, *except* when a line feed is issued with the cursor in the last line. In this case the text must be scrolled up, and the new line "cleared". The solution given can be used independently of whether or not the 64 char/line option of Mr. Smith is implemented. It requires only 2 more IC's: a 7420 quad NAND and a 74193 counter.

The present situation (with or without 64 char/line) is shown in Fig. 1a, and the required modification in Fig. 1b. When the cursor line counter (IC34) decimal value is 0-14, the behaviour is as if nothing had changed. When count 15 is reached, further "up" counting is inhibited. Instead, the next "count up" pulses increment the 74193 counter whose contents are added to the address portion A5-A8 (addressing the memory), thus causing scroll-up. This counter/adder configuration (74193 and 74283) can be built on a 1" by 2" PC card attached to the memory board, thus requiring only one "external" connection.

Software aspects. In order to maintain modularity and flexibility, a difference is made between "line feed" and "cursor down" when the cursor is in the last line. The first causes scroll-up, the second causes cursor wrap-around to the first line. Thus, programs that consider the CT-1024 as teletype replacement always use "line feed". Programs considering the complete screen field as basic output record format use all normal cursor positioning commands (up, down, left, right) etc.. Note also that IC34 has still its preset capabilities available for those wishing to use direct cursor positioning.

I hope that these suggestions will be helpful to some readers.

Sincerely yours,

R.T. Boute
Bell Telephone μClub

Francis Wellesplein 1
B-2000 Anterwerpen (Belgium)

4 MHZ Z-80 A BOARD BY NORTH STAR

News Release *Received: 77 Dec 20*

North Star Computers, Inc, announces a 4MHZ Z-80A processor board (ZPB) for use in S-100 bus computers. The Zilog Z-80 A executes compatible software more than twice as fast as the 8080. The North Star ZPB will operate in systems with or without front panels. It includes space for 1K of 2708 EPROM.

Other features include auto-jump startup and vectored interrupts. Available as a kit for just $199 or $259 fully assembled. EPROM option costs are $49 for kit and $69 assembled. For additional information write: North Star Computers, Inc, 2547 Ninth St, Berkeley, CA 94710, (415) 548-0858.

Machines know nothing of the controversy; they work in binary. Octal and hex are merely convenient ways of restating binary for the sake of humans. A manufacturer wishing to please everyone, could issue two sets of manuals, one set in octal, and the other in hex. (Or could they?)

The new fact seems important to me, that innovative manufacturers aiming at mass markets ("home computing") have stayed with octal. I mean Heath and Radio Shack.

Yours &c;. &c.,
Joe Schram

P.O. Box 1818
Washington, DC 20013

COMMUNICATIONS APPLICATION FOR THE HANDICAPPED

Tim Scully, who has been designing biofeedback equipment and doing biofeedback research for years, is currently an inmate at a Federal penitentiary where he is working on his doctorate in psychology; his dissertation project involves researching and developing biofeedback systems and techniques for use in drug rehabilitation.

With permission from prison officials, Tim has developed a Poly-88 based system which allows a cerebral palsy patient to communicate. A knee switch on the wheelchair enables the patient to construct messages which are displayed on a CRT. A complete description of the system including the assembly code listing has been published in Volme 6, No. 5 of *People's Computers* (March/April issue).

Tim is hopeful of eventually obtaining a computer system, despite the fact that prison resources are scarce and he cannot solicit donations. But while donations can't be *solicited*, they can be *accepted* so long as no strings are attached.

For further information contact:

Phyllis Cole
Editor, *People's Computers*
Box E
Menlo Park, CA 94025

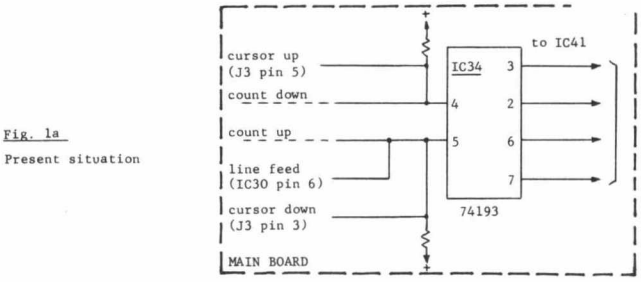


Fig. 1a
Present situation

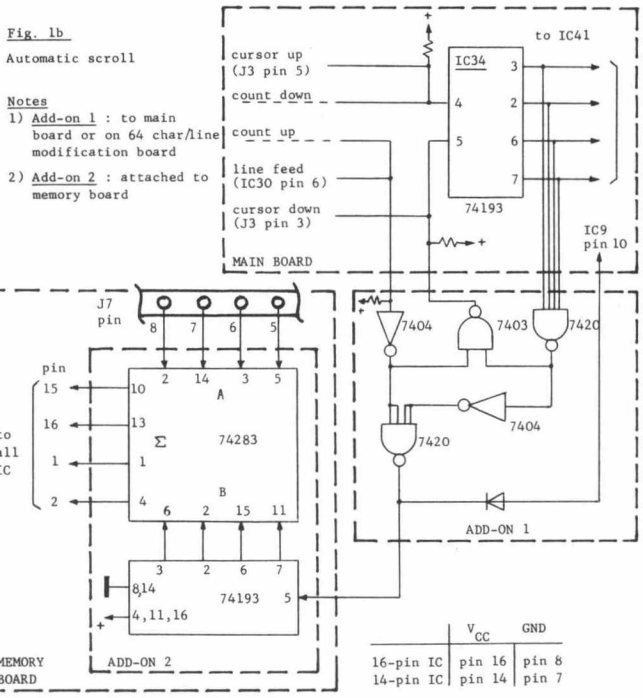


Fig. 1b
Automatic scroll

Notes
1) Add-on 1 : to main board or on 64 char/line modification board
2) Add-on 2 : attached to memory board

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 23 | March 1978 | Volume 3, Issue 3

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

Status of the USCD PASCAL Project – *Kenneth Bowles* 16
Report on efforts to implement PASCAL on various processors

SAM 76 – Additional Comments – *Ancelme Roichel* 46
Addenda to the documentation published by Dr. Dobb's

SOFTWARE

For 6502's

Renumbering and Appending BASIC Programs on the Apple-II Computer – *Steve Wozniak* 4
Apple's R & D Head tells how

A KIM Binary Calculator – *John Eaton* 13
How to solve those problems you never had until you had a computer

For 6800's

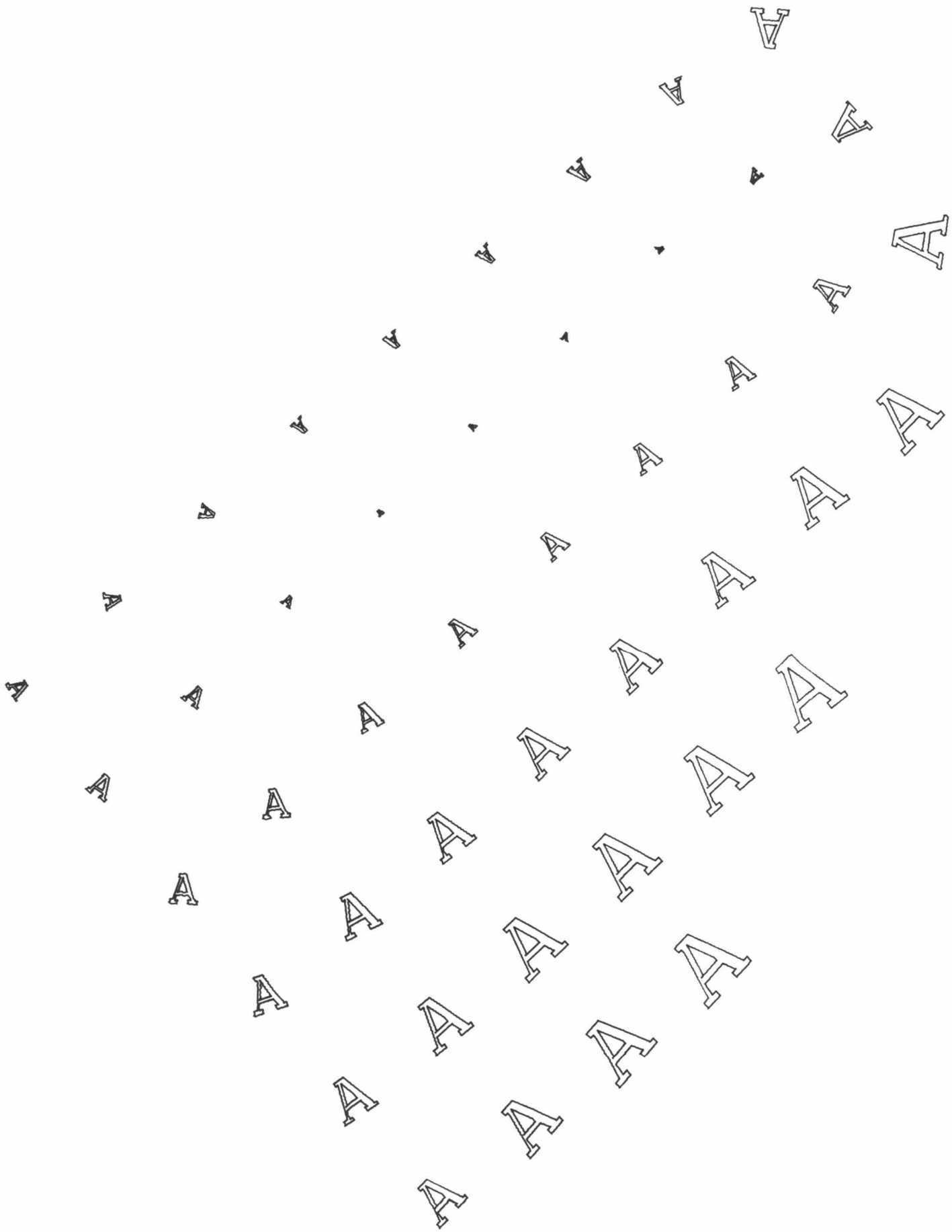
STRUBAL: A Structured BASIC Compiler – *Robert Grappel* 10

For Z-80's

A Z-80 Tracer – *Arthur W. Cline* 11

For 8080's

ACT – An 8080 Macroprocessor – *Alex Cecil* 20
PL/M Listing with documentation



© David Dameron 1979

LEST WE FORGET. . .

Technology often has a way of absorbing us in its intricacies to the extent that we can lose sight of the implications of what we are doing. It is for this reason that we have decided to interrupt our "bit fiddling" for a brief moment and refer to the ad which appears on the inside back cover of this issue.

It is certainly not the policy of Dr. Dobb's *to preach on controversial issues. But we who are interested in reason and "artificial intelligence"–still an extension of human intelligence–should stop to consider that there exists on this planet another race of intelligent beings which is being systematically hunted to extinction.*

There are, of course, innumerable other causes we could choose to comment on, but the slaughter of whales has symbolic value. It asks a question of technological society that strikes at the relationship of technology and industry to intelligent life. We at People's Computer Company wish to express the hope that the machines we play with be used to aid and increase living intellect and not to destroy it.

Tom Williams

KAPIAR ERRATA

Dear Tom, *Received: 78 Jan 23*

Here is an errata sheet for some of the goofs in my documentation. (Sorry 'bout that . . .)

p. 31, column 1, line 8 from bottom: insert 'macro' before the right paren;

p. 32, column 2, line 9: 'a@b@DEBLANK(b)' should have been 'a@DEBLANK(b)';

p. 32, column 2, line 12: 'in' should be 'is';

p. 32, column 2, middle: 'b@@a' should be 'b@ @a';

p. 33, column 2, line 26: '<parm 1>, . . .' should have been '<parm 1>˜, . . .';

p. 34, column 1, line 22 from bottom: 'pairs of a' should have been 'pairs or a';

Finally, all the examples of '&' as a continuation character were too long to fit the *DDJ* 2 column/page format, so the lines (as printed) are broken *without* continuation character, and continuation character '&' is *not* followed by a new line as intended in the example.

Cordially,
Steve Newberry

24225 Summerhill Ave.
Los Altos, CA 94022

OTTAWA MICROCOMPUTING

Dear Dr. Dobbs, *Received: 78 Feb 6*

There is a club called "The Ottawa Computing Group" which is very active locally. Although electronic components are more expensive here (import duty, etc.), I would estimate there to be well in excess of 250 microcomputers operational. Interest is very high. Being the personal owner of an RCA COSMAC, I am disappointed that the use of this processor is so low. There has been very little mention of this processor in *Dr. Dobb's* which I hope will be rectified in the future. Perhaps an article by yourselves discussing a few of the nice features of the CDO1802 over other available processors might help stimulate more interest. In my estimation, the 1802 is a superior MPU but few people realize that fact.

Keep up the good work and keep those issues coming.

Yours truly,
D. C. Underwood

Thanks for the encouragement and yes, we'd love to publish items of interest on the CDP1802 if folks out there would care to send us something. – *TRW*

6502 DISASSEMBLER FIX

Dear Dr. Dobb, *Received: 78 Feb 22*

Since my Basic language disassembler for 6502 appeared in your Volume 3, Issue 1 as "Challenging Challenger ROMs", I have discovered an omission. I somehow left out the ROR and ROL instructions. Apparently they don't occur in the ROM areas I was examining at the time. The ROL mnemonic was in my data list although the decoding scheme I used would flag it as an illegal opcode. ROR is missing entirely. A proper fix would insert ROR after ROL in line 60 and completely rearrange the table in lines 210 to 520. This simpler fix also works. Just change the following lines.

```
70 DATA SEI,STA,STX,STY,TAX,TAY,TSX,TXA,TXS,TYA,ROR
230 DATA10200,35222,56100,56100,56100,35203,3203,56100
250 DATA 29300,2221,56100,56100,7200,2200,40200,56100
260 DATA 39100,2210,40130,56100,7300,2300,40300,56100
270 DATA 8200,2222,56100,56100,56100,2203,40203,56100
280 DATA 44100,2304,56100,56100,56100,2303,40303,56100
340 DATA 38100,1210,56130,56100,28325,1300,56300,56100
350 DATA 13200,1222,56100,56100,56100,1203,56203,56100
360 DATA 46100,1304,56100,56100,56100,1303,56303,56100
```

Add

```
611 IF C=56100 OR C 1100 THEN 1000
```

Delete 630 and 631

I'm enjoying the controversy you have generated over the Heathkit systems. Although I don't use their computers, I did order some cassettes from the, believing that I could be certain of getting quality suitable for saving programs. The result has been disappointing. Although my Challenger uses the Kansas City format for recording (like Heath), all the Heath tapes consistently aquire a few errors in recording, and one is totally unusable. I don't have these problems with other cassettes, and my recorder is the identical GE model sold by Heath. Buyer beware.

Sincerely yours,
Stephen P. Smith

P.O. Box 831
Parksley, VA 23421

BEYOND THE 8080

Received: 78 Feb 10

Certainly. Am glad to tell you why I do not wish to resubscribe to your Journal. After two years of subscription, I find that the same people who started the Journal and writing for it still are doing so, still talking about 8080 materials, little being written or published about 6502 or others. To check this imbalance, just take any five issues, as example, inventory those issues for content – by microprocessor type – and see for yourself. In other words, it appears to me that the "world" has passed you by. You are "out of step". True, you have a market, but you are missing the far bigger and more diversified market that is there and needing your kind of help.

If you change, I change. Until then, good luck. You *had* a a good idea!

C. E. Brown
Rt. 1 Box 159 B1
White Stone, VA 22578

Fact is, we probably are behind the times out here in 8080 Land. But we'd be more *than happy to publish stuff on 6502's and others if folks would write it up and send it to us.* Dr. Dobb's *exists to help pass software around to the eager hands of users.* – *TRW*

RENUMBERING & APPENDING BASIC PROGRAMS ON THE APPLE-II COMPUTER

BY STEVE WOZNIAK
Apple Computer
20863 Stevens Creek Blvd, B32
Cupertino, CA 95014

The laws of entropy insure that the line numbers of a debugged and operational BASIC program give the appearance of having been selected by a KENO machine. In fact, while several texts detail how the boundary conditions of a KENO game lead to predictable outcomes, finished programs seldom exhibit this property. Many a time I have spent an extra hour retyping a finished program while spacing the line numbers evenly just to make it "look good".

Another difficulty which I have experienced is joining two BASIC programs into a single, larger one. This 'append' operation is easier to accomplish by hand than renumbering. The sophistocated user can examine the BASIC memory map and perform some manual manipulations to join the programs providing that the line numbers do not overlap. Still, the manual append operation is highly prone to error.

The APPLE-II BASIC user now has a solution to these needs in the form of a hand- or tape-loadable program RENUM/APPEND, described herein. The CALL command is used to activate one of three machine level programs. The renumber operation (RENUM) requires user specification of the origiinal line number range over which renumbering is to occur, the new initial line number to be applied to the range, and the new line number increment to use. The example below specifies that lines 200 to 340 be renumbered starting with 100 and spaced by 10's.

RANGE BEGIN	200
RANGE END	340
NEW BEGIN	100
NEW INCREMENT	10

A second RENUM entry renumbers the entire program, relieving the user of the need to specify the range begin and end parameters. The append operation (APPEND) reads the second user (BASIC) program off tape with the first in memory.

Renumber and append error conditions (memory full and line number overlap) are detected just as in BASIC. In case of error the user is notified and no program alteration occurs.

Using RENUM/APPEND

1. Load RENUM/APPEND (* 300.3D4 R)
   Note that the high-order bytes of page 3 are not loaded, preventing inadvertant alteration of the interrupt and user function (Y^C) vectors. The '*' is generated by the MONITOR, not the user.
2. Load a BASIC program.
3. To renumber entire program:

POKE 2, START L POKE 3, START H	User must supply low and high bytes of new STARTing line number.
POKE 4, INCR L POKE 5, INCR H	User must supply low and high bytes of new line number INCRement.
CALL 768	(does not alter locations 2-5)

Note: START L is equivalent to START MOD 256
START H is equivalent to START / 256

4. To renumber a range of the program

POKE 2, START L POKE 3, START H	
POKE 4, INCR L POKE 5, INCR H	
POKE 6, RANGE START L POKE 7, RANGE START H	User must supply low and high bytes of renumber range starting line number.
POKE 8, RANGE END L POKE 9, RANGE END H	User must supply low and high bytes of renumber range ending line number.
CALL 776	(does not alter locations 2-9)

5. To append program #2 (larger line numbers) to program #1 (smaller line numbers):
   (a) Load program #2
   (b) CALL 956
   Be sure you are running the tape of program #1 as this command will load it.
   (c) If you get a memory full error then use the command CALL 973 to recover the original program.

Errors

1. If not enough free memory exists to contain the line number table during pass 1 or RENUM then the message '(beep) *** MEM FULL ERR' is displayed and no renumbering occurs. The same message is displayed if not enough free memory exists to hold the product of an APPEND. In the case of APPEND, the user will have to type the BASIC command CALL 973 to recover his original program. The user can free additional memory by eliminating all active BASIC variables with the CLR command.
2. If renumbering results in a line number overlap (detected during pass 1 of RENUM) then the message '(beep) *** RANGE ERR' is displayed and no renumbering occurs. This error may mean that one or more parameters were not specified or were incorrectly specified.

Cautions

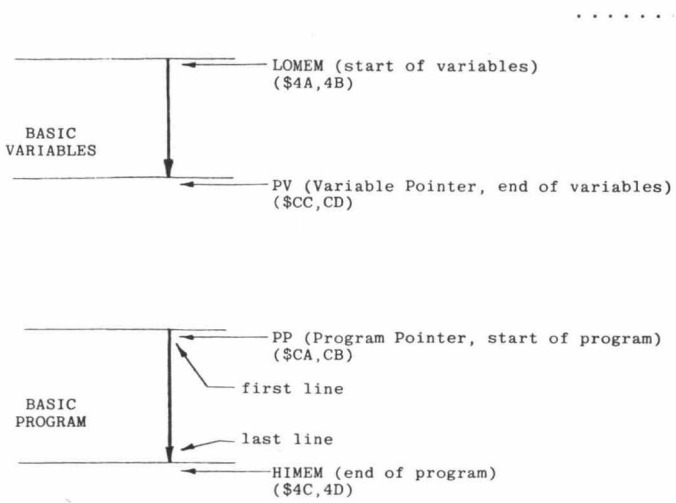
1. When appending a program, always load the one with *greater* line numbers first.
2. The user must be aware that branch target expressions may not be renumbered. For example, the statement GOTO ALPHA will not be modified by RENUM. The statement GOTO 100 + ALPHA will be modified only to reflect the new line number assigned to the old line 100.

APPLE-II BASIC STRUCTURE

An understanding of the internal representation of a BASIC program is necessary in order to develop RENUMBER and APPEND algorithms. Figure 1 illustrates the significant pointers for a program in memory. Variable and symbol table assignment begins at the location whose address is contained in

the pointer LOMEM ($4A and $4B where '$' stands for hex). This is $800 (2048) on the APPLE-II unless changed by the user with the LOMEM: command. A second pointer, PV (Variable Pointer, at $CC and $CD) contains the address of the location immediately following the last location allocated to variables. PV is equal to LOMEM if no variables are actively assigned as is the case after a NEW, CLR, or LOMEM: command. As variables are assigned, PV increases.

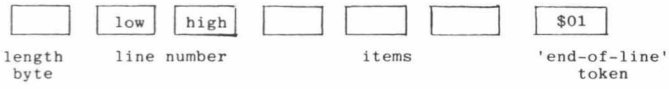
Figure 1 - MEMORY MAP



The BASIC program is stored beginning with the lowest numbered line at the location whose address is contained in the pointer PP (Program Pointer, at $CA and $CB). The pointer HIMEM ($4C and $4D) contains the address of the location immediately following the last byte of the last line of the program. This is normally the top of memory unless changed by the user with the HIMEM: command. As the program grows, PP decreases. PP is equal to HIMEM if there is no program in memory. Adequate checks in the BASIC insure that PV never exceeds PP. This is essence says that variables and program are not permitted to overlap.

Lines of a BASIC program are not stored as they were originally entered (in ASCII) on the APPLE-II due to a pre-translation stage. Internally each line begins with a length byte which may serve as a link to the next line. The length byte is immediately followed by a two-byte line number stored in binary, low-order byte first. Line numbers range from 0 to 32767. The line number is followed by 'items' of various types, the final of which is an 'end-of-line' token ($01). Refer to Figure 2.

Figure 2 - LINE REPRESENTATION



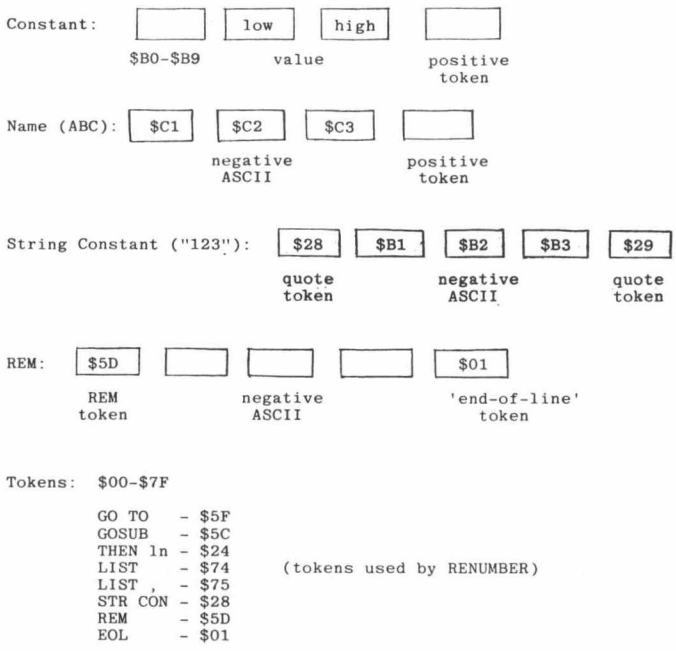
Single bytes of value less than $80 (128) are 'tokens' generated by the translator. Each token stands for a fixed unit of text as required by the syntax of the language BASIC. Some stand for keywords such as PRINT or THEN while others stand for punctuation or operators such as ',' or '+'.

Integer constants are stored as three consecutive bytes. The first contains $B0-$B9 (ASCII '0'-'9') signifying that the next two contain a binary constant stored low-order byte first. The line number itself is not preceeded by $B0-$B9. All constants are in this form including line number references such as 500 in the statement GOTO 500. Constants are always followed by a token. Although one or both bytes of a constant may be positive (less than $80) they are not tokens.

Variable names are stored as consecutive ASCII characters with the high order bit set. The first character is between $C1 and $DA (ASCII 'A' – 'Z'), distinguishing names from constants. All names are terminated by a token which is recognizable by a clear high-order bit. The '$' in string names such as A$ is treated as a token.

String constants are stored as a token of value $28 followed by ASCII text (with high-order bits set) followed by a token of value $29. REM statements begin with the REM token ($5D) followed by ASCII text (with high-order bits set) followed by the 'end-of-line' token.

Figure 3 - ITEMS



Renumber – Theory of Operation

Because of the rigid internal representation of APPLE-II BASIC programs (insured by the translator syntax check), writing a renumber program was a somewhat easier task than it would have been on many smaller BASIC's. Fortunately all constants in APPLE-II BASIC (including line number references) are preconverted to binary.

The normal renumber subroutine entry point is RENUM ($308). The RENX entry ($300) conveniently sets the renumber range for the user such that the entire program will be renumbered. RENUM extensively uses SWEET16, the code-saving 16-bit interpretive machine built into the APPLE-II (*Byte Magazine*, Nov. 1977). Occasional 6502 code is interspersed throughout RENUM for even greater code efficiency.

RENUM scans the entire program from beginning to end twice. During pass 1 a line number table is built containing all line numbers of the program found to be within the renumber range. This table begins at the address specified

by the BASIC variable pointer, PV, and is limited in length by the program pointer, PP. Each entry is two bytes long. A memory full error occurs if not enough free memory is available for the table.

As line numbers are entered in the table corresponding new line numbers are generated and both new and old are displayed. Should the new line numbers result in an 'out of ascending sequence' condition, then a range error occurs and renumbering is terminated. It is assumed that the line numbers of the original program are in ascending sequence.

The purpose of pass 2 is to scan the entire BASIC program while updating all references of line numbers found in the table to new assignments. Aside from the line numbers themselves, the line number references sought are identified as constants *immediately* preceded by one of the following tokens:

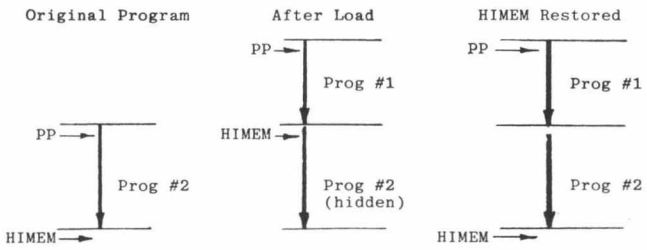
GOTO
GOSUB
THEN 1no
LIST
LIST ,

No other statement normally permitted within an APPLE-II BASIC program may contain a line number reference. No errors will occur during pass 2. Exceptions such as empty line number table and null program are properly considered by both passes of RENUM.

Append – Theory of Operation

When APPEND is called, the user program with larger line numbers will be in memory and the one with smaller line numbers will be read off tape. The current program resides between two pointers, PP and HIMEM. HIMEM is preserved and set to the value contained in PP. This 'hides' the original program and prepares to load a new one immediately above it in memory.

The BASIC load subroutine is called and a normal memory full error condition will result if not enough free memory is available to contain both programs. If this error occurs then the original program will still be hidden. Fortunately, it can be recovered by calling the tail end of APPEND at $3CD which simply restores HIMEM. If the load is successful then HIMEM is restored to its original value and both programs will be joined. No line number overlap check is performed.




```
RENUMBER EXAMPLE

Original

>LIST
  1 GOTO 100
  2 GOSUB 103
  3 IF TRUE THEN 107
  4 LIST 109,110
100 REM
103 REM
107 REM
109 REM
110 REM
200 FOR I=1 TO 10
210 PRINT I
220 NEXT I
230 GOTO 1

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

>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
100->150
103->160
107->170
109->180
110->190
>LIST
  1 GOTO 150
  2 GOSUB 160
  3 IF TRUE THEN 170
  4 LIST 180,190
150 REM
160 REM
170 REM
180 REM
190 REM
200 FOR I=1 TO 10
210 PRINT I
220 NEXT I
230 GOTO 1

     Renumber lines 100-110 to start at
          10 spaced by 5

>POKE 2, 10 MOD 256
>POKE 3, 10 / 256
>POKE 4, 5 MOD 256
>POKE 5, 5 / 256
```

```
>CALL 768
1->10
2->15
3->20
4->25
150->30
160->35
170->40
180->45
190->50
200->55
210->60
220->65
230->70

>LIST
   10 GOTO 30
   15 GOSUB 35
   20 IF TRUE THEN 40
   25 LIST 45,50
   30 REM
   35 REM
   40 REM
   45 REM
   50 REM
   55 FOR I=1 TO 10
   60 PRINT I
   65 NEXT I
   70 GOTO 10
```

APPEND EXAMPLE

```
>LIST
  100 REM
  200 REM THE ORIGINAL PROGRAM
  300 REM

>CALL 956

>LIST
   10 REM
   20 REM THIS PROGRAM CAME FROM TAPE
   30 REM
  100 REM
  200 REM THE ORIGINAL PROGRAM
  300 REM
```

APPLE-II BASIC RENUMBER/APPEND SUBROUTINES

```
 1      TITLE 'APPLE-II BASIC RENUMBER/APPEND SUBROUTINES'
 2     ************************
 3     *                      *
 4     *   APPLE-II BASIC     *
 5     * RENUMBER AND APPEND  *
 6     *     SUBROUTINES      *
 7     *                      *
 8     *       RENUMBER       *
 9     *  NEW INITIAL (2,3)   *
10     *  NEW INCR (4,5)      *
11     *  RANGE BEG (6,7)     *
12     *  RANGE END (8,9)     *
13     *                      *
14     *   USE RENX ENTRY     *
15     *  FOR RENUMBER ALL    *
16     *                      *
17     *   WOZ     11/16/77   *
18     * APPLE COMPUTER INC.  *
19     *                      *
20     ************************
21
22      SUBTTL 6502 EQUATES
23     ROL         EPZ  $0          LOW-ORDER SW16 RO BYTE
24     ROH         EPZ  $1            HI-ORDER.
25     R11L        EPZ  $16         LOW-ORDER SW16 R11 BYTE
26     R11H        EPZ  $17           HI-ORDER.
27     HIMEM       EPZ  $4C         BASIC HIMEM POINTER.
28     PPL         EPZ  $CA         BASIC PROG POINTER.
29     PVL         EPZ  $CC         BASIC VAR POINTER.
30     MEMFULL     EQU  $E36B       BASIC MEM FULL ERROR.
31     PRDEC       EQU  $E51B       BASIC DECIMAL PRINT SUBR.
32     RANGERR     EQU  $EE68       BASIC RANGE ERROR.
33     LOAD        EQU  $F0DF       BASIC LOAD SUBR.
34     SW16        EQU  $F689       SWEET16 ENTRY.
35     CROUT       EQU  $FD8E       CAR RET SUBR.
36     COUT        EQU  $FDED       CHAR OUT SUBR.
37
38      SUBTTL SWEET16 EQUATES
39     ACC         EPZ  $0          SWEET16 ACCUMULATOR.
40     NEWLOW      EPZ  $1          NEW INITIAL LNO.
41     NEWINCR     EPZ  $2          NEW LNO INCR.
42     LNLOW       EPZ  $3          LOW LNO OF RENUM RANGE.
43     LNHI        EPZ  $4          HI LNO OF RENUM RANGE.
44     TBLSTRT     EPZ  $5          LNO TABLE START.
45     TBLNDX1     EPZ  $6          PASS 1 LNO TBL INDEX.
46     TBLIM       EPZ  $7          LNO TABLE LIMIT.
47     SCR8        EPZ  $8          SCRATCH REG.
48     HMEM        EPZ  $8          HIMEM (END OF PRGM).
49     SCR9        EPZ  $9          SCRATCH REG.
50     PRGNDX      EPZ  $9          PASS 1 PROG INDEX.
51     PRGNDX1     EPZ  $A            ALSO PROG INDEX.
52     NEWLN       EPZ  $B          NEXT 'NEW LNO'.
53     NEWLN1      EPZ  $C          PRIOR 'NEW LNO' ASSIGN
54     TBLND       EPZ  $6          PASS 2 LNO TABLE END.
55     PRGNDX2     EPZ  $7          PASS 2 PROG INDEX.
56     CHR0        EPZ  $9          ASCII '0'.
57     CHRA        EPZ  $A          ASCII 'A'.
58     MODE        EPZ  $C          CONST/LNO MODE.
```

```
59    TBLNDX2     EPZ   $B          LNO TBL IDX FOR UPDATE
60    OLDLN       EPZ   $D          OLD LNO FOR UPDATE.
61    STRCON      EPZ   $B          BASIC STR CON TOKEN.
62    REM         EPZ   $C          BASIC REM TOKEN.
63    R13         EPZ   $D          SWEET16 REG 13 (CPR R
64    THEN        EPZ   $D          BASIC THEN TOKEN.
65    LIST        EPZ   $D          BASIC LIST TOKEN.
66    SCRC        EPZ   $C          SCRATCH REG FOR APPEND
67
```

APPLE-II BASIC RENUMBER SUBROUTINE - PASS 1

9:53 A.M., 11/21/1977

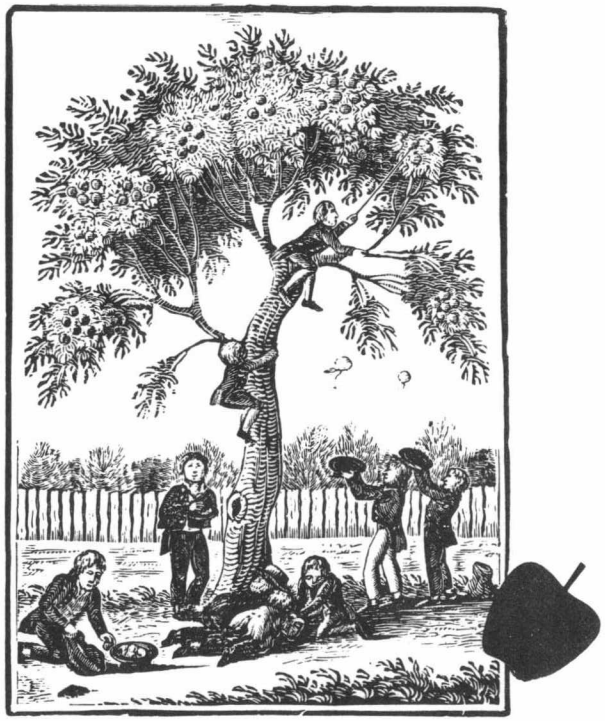
```
                    68     SUBTTL APPLE-II BASIC RENUMBER SUBROUTINE - PASS 1
                    69           ORG   $300
0300: 20 89 F6      70   RENX    JSR   SW16          OPTIONAL RANGE ENTRY.
0303: B0            71           SUB   ACC
0304: 33            72           ST    LNLOW         SET LNLOW=0,
0305: 34            73           ST    LNHI            LNHI=$FFFF
0306: F4            74           DCR   LNHI
0307: 00            75           RTN
0308: 20 89 F6      76   RENUM   JSR   SW16
030B: 18 4C 00      77           SET   SCR8,HIMEM
030E: 68            78           LDD   @SCR8         BASIC HIMEM POINTER
030F: 38            79           ST    HMEM            TO HMEM.
0310: 19 CE 00      80           SET   SCR9,PVL+2
0313: C9            81           POPD  @SCR9         BASIC VAR PTR TO
0314: 35            82           ST    TBLSTRT         TBLSTRT AND TBLNDX1.
0315: 36            83           ST    TBLNDX1
0316: 21            84           LD    NEWLOW        COPY NEWLOW (INITIAL)
0317: 3B            85           ST    NEWLN           TO NEWLN.
0318: 3C            86           ST    NEWLN1
0319: C9            87           POPD  @SCR9         BASIC PROG PTR
031A: 37            88           ST    TBLIM           TO TBLIM
031B: 39            89           ST    PRGNDX          AND PRGNDX.
031C: 29            90   PASS1   LD    PRGNDX
031D: D8            91           CPR   HMEM          IF PRGNDX >= HMEM
031E: 03 46         92           BC    PASS2           THEN DONE PASS 1.
0320: 3A            93           ST    PRGNDX1
0321: 26            94           LD    TBLNDX1
0322: E0            95           INR   ACC           IF < 2 BYTES AVAIL IN
0323: D7            96           CPR   TBLIM           LNO TABLE THEN RETUR
0324: 03 38         97           BC    MERR            WITH 'MEM FULL' MSG.
0326: 4A            98           LD    @PRGNDX1
0327: A9            99           ADD   PRGNDX        ADD LEN BYTE TO
0328: 39           100           ST    PRGNDX          PROG INDEX.
0329: 6A           101           LDD   @PRGNDX1      LINE NUMBER.
032A: D3           102           CPR   LNLOW         IF < LNLOW THEN
032B: 02 2A        103           BNC   P1B             GO TO P1B.
032D: D4           104           CPR   LNHI          IF > LNHI THEN
032E: 02 02        105           BNC   P1A             GO TO P1C.
0330: 07 30        106           BNZ   P1C
0332: 76           107   P1A     STD   @TBLNDX1      ADD TO LNO TABLE.
0333: 00           108           RTN
0334: A5 01        109           LDA   ROH           *** 6502 CODE ***
0336: A6 00        110           LDX   ROL
0338: 20 1B E5     111           JSR   PRDEC         PRINT OLD LNO '->' NE
033B: A9 AD        112           LDA   #$AD            (RO,R11) IN DECIMAL.
033D: 20 ED FD     113           JSR   COUT
0340: A9 BE        114           LDA   #$BE
0342: 20 ED FD     115           JSR   COUT
0345: A5 17        116           LDA   R11H
0347: A6 16        117           LDX   R11L
0349: 20 1B E5     118           JSR   PRDEC
034C: 20 8E FD     119           JSR   CROUT
034F: 20 8C F6     120           JSR   SW16+3        *** END 6502 CODE ***
0352: 2B           121           LD    NEWLN
0353: 3C           122           ST    NEWLN1        COPY NEWLN TO NEWLN1
0354: A2           123           ADD   NEWINCR         AND INCR NEWLN BY
0355: 3B           124           ST    NEWLN           NEWINCR.
0356: 0D           125           NUL                 (WILL SKIP NEXT INST).
0357: D1           126   P1B     CPR   NEWLOW       IF LOW LNO < NEWLOW
0358: 02 C2        127           BNC   PASS1           THEN RANGE ERR.
035A: 00           128   RERR    RTN               PRINT 'RANGE ERR' MSG
035B: 4C 68 EE     129           JMP   RANGERR         AND RETURN.
035E: 00           130   MERR    RTN               PRINT 'MEM FULL' MSG
035F: 4C 6B E3     131           JMP   MEMFULL         AND RETURN.
0362: EC           132   P1C     INR   NEWLN1        IF HI LNO <= MOST REC
0363: DC           133           CPR   NEWLN1          NEWLN THEN RANGE ERR
0364: 02 F4        134           BNC   RERR
                   135
```

APPLE-II BASIC RENUMBER SUBROUTINE - PASS 2

9:53 A.M., 11/21/1977

```
                   136     SUBTTL APPLE-II BASIC RENUMBER SUBROUTINE - PASS 2
0366: 19 B0 00     137   PASS2   SET   CHR0,$B0      ASCII '0'
0369: 1A C1 00     138           SET   CHRA,$C1      ASCII 'A'
036C: 27           139   P2A     LD    PRGNDX2
036D: D8           140           CPR   HMEM          IF PROG INDEX = HIMEM
036E: 03 63        141           BC    DONE            THEN DONE PASS 2.
0370: E7           142           INR   PRGNDX2       SKIP LEN BYTE.
0371: 67           143           LDD   @PRGNDX2      LINE NUMBER.
0372: 3D           144   UPDATE  ST    OLDLN         SAVE OLD LNO.
0373: 25           145           LD    TBLSTRT
0374: 3B           146           ST    TBLNDX2       INIT LNO TABLE INDEX.
0375: 21           147           LD    NEWLOW        INIT NEWLN1 TO NEWLOW.
0376: 1C 00 00     148           SET   NEWLN1,0      (WILL SKIP NEXT 2 INS
                   149           ORG   *-2
0377: 2C           150   UD2     LD    NEWLN1
0378: A2           151           ADD   NEWINCR       ADD INCR TO NEWLN1.
0379: 3C           152           ST    NEWLN1
037A: 2B           153           LD    TBLNDX2       IF LNO TBL IDX = TBLND
037B: B6           154           SUB   TBLND           THEN DONE SCANNING
037C: 03 07        155           BC    UD3             LNO TABLE.
037E: 6B           156           LDD   @TBLNDX2      NEXT LNO FROM LNO TAB
037F: BD           157           SUB   OLDLN         LOOP TO UD2 IF NOT SA
0380: 07 F5        158           BNZ   UD2             AS OLDLN.
0382: C7           159           POPD  @PRGNDX2      REPLACE OLD LNO WITH
0383: 2C           160           LD    NEWLN1          CORRESPONDING NEW L
0384: 77           161           STD   @PRGNDX2
0385: 1B 28 00     162   UD3     SET   STRCON,$28    STR CON TOKEN.
0388: 1C 00 00     163           SET   MODE,0        (SKIPS NEXT 2 INSTR'S)
                   164           ORG   *-2
0389: 67           165   GOTCON  LDD   @PRGNDX2
038A: FC           166           DCR   MODE          IF MODE = 0 THEN UPDA
038B: 08 E5        167           BMI   UPDATE          LNO REF.
038D: 47           168   ITEM    LD    @PRGNDX2      BASIC ITEM.
038E: D9           169           CPR   CHR0
038F: 02 09        170           BNC   CHKTOK        CHECK TOKEN FOR SPECI
0391: DA           171           CPR   CHRA          IF >= '0' AND < 'A' T
0392: 02 F5        172           BNC   GOTCON          SKIP CONST OR UPDAT
0394: F7           173   SKPASC  DCR   PRGNDX2
0395: 67           174           LDD   @PRGNDX2      SKIP ALL NEG BYTES OF
0396: 05 FC        175           BM    SKPASC          STR CON, REM, OR NA
0398: F7           176           DCR   PRGNDX2
0399: 47           177           LD    @PRGNDX2
039A: DB           178   CHKTOK  CPR   STRCON        STR CON TOKEN?
039B: 06 F7        179           BZ    SKPASC          YES, SKIP SUBSEQUEN
039D: 1C 5D 00     180           SET   REM,$5D
03A0: DC           181           CPR   REM           REM TOKEN?
03A1: 06 F1        182           BZ    SKPASC          YES, SKIP SUBSEQUEN
03A3: 08 13        183           BMI   CONTST        GOSUB, LOOK FOR LNO.
03A5: FD           184           DCR   R13
```

```
03A6: FD       185              DCR  R13          (TOKEN $5F IS GOTO)
03A7: 06 0F    186              BZ   CONTST       THEN LNO, LOOK FOR LNO.
03A9: 1D 24 00 187              SET  THEN,$24
03AC: DD       188              CPR  THEN
03AD: 06 09    189              BZ   CONTST       THEN LNO, LOOK FOR LNO.
03AF: F0       190              DCR  ACC
03B0: 06 BA    191              BZ   P2A          EOL (TOKEN $01)?
03B2: 1D 74 00 192              SET  LIST,$74
03B5: BD       193              SUB  LIST         SET MODE = 0 IF LIST
03B6: 09 01    194              BNM1 CONTS2         OR LIST COMMA ($73,#
03B8: B0       195 CONTST       SUB  ACC          CLEAR MODE FOR LNO
03B9: 3C       196 CONTS2       ST   MODE           UPDATE CHECK.
03BA: 01 D1    197              BR   ITEM         CHECK NEXT BASIC ITEM.
               198

                        APPLE-II BASIC APPEND SUBROUTINE
9:53 A.M., 11/21/1977
               199   SUBTTL APPLE-II BASIC APPEND SUBROUTINE
03BC: 20 89 F6 200 APPEND       JSR  SW16
03BF: 1C 4E 00 201              SET  SCRC,HIMEM+2
03C2: CC       202              POPD #SCRC        SAVE HIMEM.
03C3: 38       203              ST   HMEM
03C4: 19 CA 00 204              SET  SCR9,PPL
03C7: 69       205              LDD  #SCR9        SET HIMEM TO PRESERVE
03C8: 7C       206              STD  #SCRC          PROGRAM.
03C9: 00       207              RTN
03CA: 20 DF F0 208              JSR  LOAD         LOAD FROM TAPE.
03CD: 20 89 F6 209              JSR  SW16
03D0: CC       210              POPD #SCRC        RESTORE HIMEM TO SHOW
03D1: 28       211              LD   HMEM           BOTH PROGRAMS
03D2: 7C       212              STD  #SCRC          (OLD AND NEW).
03D3: 00       213 DONE         RTN               RETURN.
03D4: 60       214              RTS
********SUCCESSFUL ASSEMBLY: NO ERRORS
```



```
CROSS-REFERNCE: APPLE-II BASIC RENUMBER/APPEND SUBROUTINES
ACC       0000   0071 0095 0190 0195
APPEND    03BC
CHKTOK    039A   0170
CHR0      0009   0137 0169
CHRA      000A   0138 0171
CONTS2    03B9   0194
CONTST    03B8   0183 0186 0189
COUT      FDED   0113 0115
CROUT     FD8E   0119
DONE      03D3   0141
GOTCON    0389   0172
HIMEM     004C   0077 0201
HMEM      0008   0079 0091 0140 0203 0211
ITEM      038D   0197
LIST      000D   0192 0193
LNHI      0004   0073 0074 0104
LNLOW     0003   0072 0102
LOAD      F0DF   0208
MEMFULL   E36B   0131
MERR      035E   0097
MODE      000C   0163 0166 0196
NEWINCR   0002   0123 0151
NEWLN     000B   0085 0121 0124
NEWLN1    000C   0086 0122 0132 0133 0148 0150 0152 0160
NEWLOW    0001   0084 0126 0147
OLDLN     000D   0144 0157
P1A       0332   0105
P1B       0357   0103
P1C       0362   0106
P2A       036C   0191
PASS1     031C   0127
PASS2     0366   0092
PPL       00CA   0204
PRDEC     E51B   0111 0118
PRGNDX    0009   0089 0090 0099 0100
PRGNDX1   000A   0093 0098 0101
PRGNDX2   0007   0139 0142 0143 0159 0161 0165 0168 0173 0174 0176 0177
PVL       00CC   0080
ROH       0001   0109
ROL       0000   0110
R11H      0017   0116
R11L      0016   0117
R13       000D   0184 0185
RANGERR   EE68   0129
REM       000C   0180 0181
RENUM     0308
RENX      0300
RERR      035A   0134
SCR8      0008   0077 0078
SCR9      0009   0080 0081 0087 0204 0205
SCRC      000C   0201 0202 0206 0210 0212
SKPASC    0394   0175 0179 0182
STRCON    000B   0162 0178
SW16      F689   0070 0076 0120 0200 0209
TBLIM     0007   0088 0096
TBLND     0006   0154
TBLNDX1   0006   0083 0094 0107
TBLNDX2   000B   0146 0153 0156
TBLSTRT   0005   0082 0145
THEN      000D   0187 0188
UD2       0377   0158
UD3       0385   0155
UPDATE    0372   0167
```

STRUBAL: A STRUCTURED BASIC COMPILER

Gentlepeople: *Received: 78 Jan 25*

This letter was inspired by the letter of Neils Laughlin in your issue No. 19 page 19. Mr. Laughlin is describing a proposed structured language for microcomputer use. I agree that the structures proposed by Djikstra and others (DO WHILE, IF-THEN-ELSE, etc.) are good for writing readable programs, yet I look at the sample listing on page 20 and it is far from readable. It took several minutes of reading to figure out what the program was doing. I feel that computer languages should not demand such effort from the reader. To show what I feel a good microcomputer language should look like, I enclosed a listing of the same program, coded in my partner (Jack Hemenway) and my new language: STRUBAL. (STRUctured BAsic Language™) This langauge is implemented as a full compiler which runs on 6800 processors with 16K of memory. The compiler is written in 6800 assembler language and is presently running on a SWTP system with ICOM floppies, although STRUBAL has also run from a dual cassette system. The compiler runs in three passes. Pass one converts the source language input file into an equivalent *assembly-language* output file. Passes two and three are an extended 6800 assembler. The use of assembly langauge as the intermediate file allows the programmer to easily read the intermediate file, to optimize it if desired, and to intersperse his own assembly-language statements wherever desired. This "inline" or "crutch-code" facility gives the programmer great power. STRUBAL itself is a very powerful langauge. There are three data types supported: integer (16-bit), floating-point (10 digit BCD), and character string (bytes). Full math support is provided in several run-time packages. (trigonometry, logarithms, etc.) String functions in the language include string comparisons, string concatenation, variable-length strings, string-to-value and value-to-string conversion, etc. STRUBAL code is fully relocatable and separately compiled program modules may be linked to form a single program. Common data areas, external references, etc. are all handled by a relocating loader which is part of the system. Subroutine calls with named parameters are supported. Sequential and random disk file access is supported. (Tapes have only sequential files, obviously.) STRUBAL has all of the structured-programming forms (WHILE, IF-THEN-ELSE, CASE, etc.), plus other forms taken from BASIC and FORTRAN. (IF-statement, GOTO, GOSUB, FOR-NEXT, CALL, etc.) PEEK and POKE are provided, as is a function to return the address of a variable or label. Any data type may be an array of one or two dimensions. Extensive I/O format capability is provided for either console or disk (tape) file I/O. No line numbers are used, since a full editor is available to build source files and only those lines requiring transfer labels need have labels–and the labels are alphanumeric names, not hard-to-follow numbers.

That was a quick survey of STRUBAL. It is a fast compiler . . . the sample program was fully compiled (3 passes) in less than four minutes. The assembler part of the package can assemble the entire compiler in about 12 minutes. (on floppies) Much of this speed comes from a hash-coded symbol table and a binary-searched keyword table. Keywords are searched on their first three characters, symbols are searched on the first six characters.

Let me make a few notes on the STRUBAL program listing. All lines with an asterisk in the first column are comments and will also appear in the assembly intermediate file. All STRUBAL functions are indicated by a dot. .RAND returns a pseudo-random 16-bit value. Slashes in I/O statements generate carriage return/linefeeds. A percent sign indicates string I/O. STRUBAL does not require the indentation for blocks; it was done for readability. The top line is the command to the ICOM disk system to print the file "&GUSS" from disk one.

Since STRUBAL is a full compiler, all debugging tools used in assembler programming can be used with it. STRUBAL assumes some system monitor (like MIKBUG), but any reasonable monitor can be patched in. It is fully debugged and operational now.

Enough sales-pitch for now. We hope to market STRUBAL soon, and will be selling full source listings. STRUBAL is a lot more complex than a "tiny BASIC", but I feel it returns a lot of power for the investment.

Sincerely yours,
Robert D. Grappel
148 Wood Street
Lexington, MA 02173

```
PRINT,&GUSS:1
* NUMBER-GUESS PROGRAM IN STRUBAL LANGUAGE
* ILLUSTRATES STRUCTURED-PROGRAMMING
* NOTE: RNDNO LIMITED TO RANGE 1-100
*
        INTEGER RNDNO,NGUESS,GUESS
        DSTRING YES,GSTAT
*
        STRING YES='Y'
        STRING GSTAT=YES
*
        WHILE GSTAT .EQ. YES
                RNDNO=.ABS(.RAND(1))/328
                NGUESS=1
                GUESS=-1
                WHILE GUESS .NE. RNDNO
                        INPUT /,'GUESS=',GUESS,/
                        IF GUESS.LT.RNDNO PRINT ' TOO LOW',/
                        IF GUESS.EQ.RNDNO PRINT /,'O.K. IN ',NGUESS,/
                        IF GUESS.GT.RNDNO PRINT ' TOO HIGH',/
                        NGUESS=NGUESS+1
                BLOCK
*
                INPUT /,'PLAY AGAIN? (Y OR N)',%GSTAT
        BLOCK
*
        STOP
        END

!
```

MACROFLOPPY™ AT $695

News Release *Received: 78 Jan 5*

A new series of fully packaged and assembled disk drives priced as low as $695 – including software, S-100 compatible controller and 143,000 byte capacity – has been announced by Micropolis Corporation. Developed for the retail microcomputer market, the new MacroFloppy™ series initially includes two versions:

Model 1041 packages a drive, enclosure, cabling and connectors, disk operating system and disk extended BASIC at a suggested retail price of $695. It is intended for integration into any 8080A or Z-80 microcomputer chassis.

Model 1042, suggested retail of $795, adds a power supply and DC regulators for desktop use.

Delivery is 30 days on both models, and they are factory warranted for 90 days.

Both systems come with complete file protect and built in ROM bootstrap. MacroFloppys are hard-sectored into 16 sectors each 256 bytes long; total tracks per surface is 35. Both offer transfer rates of 250K bits/second at an average rotational latency time of 100 msec. Access time track to track is 30 msec, and recording density 5162 bits per inch.

The disk operating system offered with each MacroFloppy subsystem is a complete package including assembler, file management routines and utilities to support 8080A and Z-80 programs. MacroFloppy DOS will function in microcomputer systems with at least 16K of main memory. The company's disk extended BASIC requires 24K of main storage. Contact Micropolis Corporation, 7959 Deering Ave, Canoga Park, CA 91304, (213) 703-1121.

A Z-80 Tracer

Dear Dr. Dobbs,

Received: 78 Feb 12

Here is a program which I think will be of some value to your readers! Since you have never to my knowledge published a Z-80 program, I thought that I might as well get things off the ground.

This program is a valuable debugging tool. It will move a breakpoint through the object program while executing it. The breakpoint is a restart 070_8 that jumps to a register dump routine. The dump routine will jump to TRACER after displaying the registers. TRACER will then restore the object program op code at the location of the breakpoint. It will then calcluate the location of the next op code to be executed, load the first two bytes into the Tracer program, insert a restart at that location, then restore all the prime registers. It will then return to the object program where it was interrupted and execute the restored op code.

The TRACER program uses the DIGITAL GROUP Z-80 operating system register dump routine, but can be used with any routine to display any type of data desired. The routine must push all prime registers into the stack located at 002000_8 (the stack cannot be used by the object program). The TRACER program starts at 006110_8. Then after storing the stack pointer it will jump to the dump routine at 006134_8. The dump routine must jump back into TRACER at 006154_8. All inputs and outputs are done in the dump routine.

To use TRACER, load a 377_8 (restart jump to 006110_8) at the location where debugging is to begin. Then load B1&B2 in TRACER with the first two bytes of the op code. When the object program is executed the registers will be displayed (or any data desired) when the restart is encountered.

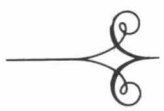
All addressing is in split octal mode. This program will only work on a Z-80 system but will trace 8080 programs since it is a subset of the Z-80 language.

Sincerely,
Arthur W. Cline

USAFSS, Box 18627
San Antonio, TX 78243
(512) 925-2380

```
006110 355 163 166 006  PREDUMP   LD(006166),SP
006114 355 163 151 006            LD(006151),SP
006120 042 132 006                LD(006132),HL
006123 341                        POP HL
006124 053                        DEC HL
006125 061 000 002      RETURN    LD SP,(002000)
006130 345                        PUSH HL
006131 041 *** ***                LD HL,******
006134 303 000 003                JP (DUMP)
006137 041 022 000                LD HL,000022
006142 071                        ADD HL,SP
006143 371                        LD SP,HL
006144 341                        POP HL
006145 321                        POP DE
006146 301                        POP BC
006147 361                        POP AF
006150 061 *** ***                LD SP,******
006153 311                        RET
006154 026 377          START     LD D,377
006156 052 314 007                LD HL,(b2,b1)
006161 355 163 316 007            LD (sp),SP
006165 061 *** ***                LD SP,******
006170 175                        LD A,L
006171 137                        LD E,A
006172 376 375                    CP 375
006174 312 352 006                JP Z,(IX)
006177 376 335                    CP 335
006201 312 352 006                JP Z,(IX)
006204 376 313                    CP 313
006206 312 024 007                JP Z,(2BYTE)
006211 376 355                    CP 355
006213 174                        LD A,H
006214 312 334 006                JP Z,(ESCAPE)
006217 376 351                    CP 351
006221 312 131 007                JP Z,(JP(HL))
006224 001 012 000                LD BC,000012
006225 000                        NOP
006230 041 322 007                LD HL,007322
006233 355 261                    CPIR
006235 257                        XOR A
006236 271                        CP C
006237 173                        LD A,E
006240 302 244 007                JP NZ,(RETTST)
006243 173                        LD A,E
006244 001 011 000                LD BC,000011
006247 355 261                    CPIR
006251 257                        XOR A
006252 271                        CP C
006253 302 014 007                JP NZ,(3BYTE)
006256 173                        LD A,E
006257 001 023 000                LD BC,000023
006262 355 261                    CPIR
006264 257                        XOR A
006265 271                        CP C
006266 302 024 007                JP NZ,(2BYTE)
006271 173                        LD A,E
006272 001 023 000                LD BC,000023
006275 355 261                    CPIR
006277 257                        XOR A
006300 271                        CP C
006301 302 210 007                JP NZ,(CALLTST)
006304 173                        LD A,E
006305 001 007 000                LD BC,000007
006310 355 261                    CPIR
006312 257                        XOR A
006313 271                        CP C
006314 302 112 007                JP NZ,(RST)
006317 173                        LD A,E
006320 001 007 000                LD BC,000007
006323 355 261                    CPIR
006325 257                        XOR A
006326 271                        CP C
006327 302 162 007                JP NZ,(JRTST)
006332 030 101                    JR(1BYTE)
006334 001 012 000      ESCAPE    LD BC,000012
006337 041 075 010                LD HL 010075
006342 355 261                    CPIR
006344 257                        XOR A
006345 271                        CP C
006346 040 031                    JR NZ,(4BYTE)
006350 030 052                    JR (2BYTE)
006352 174                IX      LD A,H
006353 001 031 000                LD BC,000031
006356 041 031 010                LD HL,010031
006361 355 261                    CPIR
006363 257                        XOR A
006364 271                        CP C
006365 040 025                    JR NZ,(3BYTE)
006367 173                        LD A,E
006370 001 013 000                LD BC,000013
006373 355 261                    CPIR
006375 257                        XOR A
006376 271                        CP C
006377 040 023                    JR NZ,(2BYTE)
007001 341                4BYTE   POP HL
007002 053                        DEC HL
007003 345                        PUSH HL
007004 173                        LD A,E
007005 167                        LD (HL),A
007006 043                        INC HL
007007 043                        INC HL
007010 043                        INC HL
007011 043                        INC HL
007012 030 137                    JR(PREEND)
007014 341                3BYTE   POP HL
007015 053                        DEC HL
007016 345                        PUSH HL
007017 173                        LD A,E
007020 167                        LD(HL),A
```

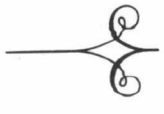
```
007021 043                      INC HL
007022 030 005                  JR 005
007024 341              2BYTE   POP HL
007025 053                      DEC HL
007026 345                      PUSH HL
007027 173                      LD A,E
007030 167                      LD (HL),A
007031 043                      INC HL
007032 043                      INC HL
007033 030 116                  JR (007153)
007035 341              1BYTE   POP HL
007036 053                      DEC HL
007037 345                      PUSH HL
007040 173                      LD A,E
007041 167                      LD (HL),A
007042 043                      INC HL
007043 030 106                  JR (007153)
007045 341               CALL   POP HL
007046 053                      DEC HL
007047 345                      PUSH HL
007050 173                      LD A,E
007051 167                      LD (HL),A
007052 043                      INC HL
007053 136                      LD E,(HL)
007054 043                      INC HL
007055 126                      LD D,(HL)
007056 353                      EX DE,HL
007057 176                      LD A,(HL)
007060 066 377                  LD (HL),377
007062 030 071                  JR(007155)
007064 341                JR    POP HL
007065 053                      DEC HL
007066 345                      PUSH HL
007067 173                      LD A,E
007070 167                      LD (HL),A
007071 043                      INC HL
007072 176                      LD A,(HL)
007073 270                      CP B
007074 372 103 007              JP M,007103
007077 043                      INC HL
007100 205                      ADD L
007101 030 004                  JR (007107)
007103 057                      CPL
007104 107                      LD B,A
007105 175                      LD A,L
007106 220                      SUB B
007107 157                      LD L,A
007110 030 041                  JR (007153)
007112 341                RST   POP HL
007113 053                      DEC HL
007114 345                      PUSH HL
007115 173                      LD A,E
007116 167                      LD (HL),A
007117 356 307                  XOR 307
007121 074                      INC A
007122 157                      LD L,A
007123 046 000                  LD H,000
007125 371                      LD SP,HL
007126 341                      POP HL
007127 030 022                  JR (007153)
007131 341              JP(HL)  POP HL
007132 053                      DEC HL
007133 345                      PUSH HL
007134 173                      LD A,E
007135 167                      LD (HL),A
007136 052 366 001              LD HL,(001366)
007141 030 010                  JR(007153)
007143 341                RET   POP HL
007144 053                      DEC HL
007145 345                      PUSH HL
007146 173                      LD A,E
007147 167                      LD (HL),A
007150 341                      POP HL
007151 341                      POP HL
007152 000                      NOP
007153 176                      LD A,(HL)
007154 162                      LD (HL),D
007155 062 314 007              LD (007314),A
007160 030 113                  JR (END)
007162 173               JRTST  LD A,E
007163 061 374 001              LD SP,001374
007166 062 172 007              LD (007172),A
007171                          POP AF
007172 *** 006                  JR*(007202)
007174 355 173 166 006          LD SP,(006166)
007200 030 222                  JR (2BYTE)
007202 355 173 166 006          LD SP,(006166)
007206 030 254                  JR (JR)
007210 173               CALLTST LD A,E
007211 061 374 001              LD SP,001374
007214 062 223 007              LD(007223),A
007217 361                      POP AF
007220 061 377 377              LD SP,377377
007223 *** 235 007              ***007235
007226 355 173 166 006          LD SP,(006166)
007232 303 014 007              JP (3BYTE)
007235 355 173 166 006          LD SP,(006166)
007241 303 045 007              JP (CALL)
007244 061 374 001       RETTST LD SP,001374
007247 062 256 007              LD (007256),A
007252 361                      POP AF
007253 061 320 007              LD SP,007320
007256 ***                      RET *
007257 355 173 166 006          LD SP,(006166)
007263 303 035 006              JP (1BYTE)
007266 355 173 166 006          LD SP,(006166)
007272 303 143 007              JP (RET)
007275 355 173 316 007      END LD SP,(007316)
007301 043                      INC HL
007302 176                      LD A,(HL)
007303 062 315 007              LD(007315),A
007306 303 137 006              JP (RETURN)
007311 000                      NOP
007312 000                      NOP
007313 000                      NOP
007314 ***                      B1
007315 ***                      B2
007316 ***                      SP L
007317 ***                      SP H
007320 266                      DUMMY SP L
007321 007                      DUMMY SP H
```



```
                                041
                                042
                                052
                                103
                                113
                                123
                                133
                                163
                                173
                                000

                        011
                        031
                        061
                        043
                        053
                        341
                        351
                        371
                        343
                        345
                        000

                064
                065
                160
                161
                162
                163
                164
                165
                167
                106
                116
                126
                136
                146
                156
                176
                206
                216
                226
                236
                246
                256
                266
                276
                000

010031
```

TABLES

```
                317
                327
                337
                347
                357
                367
                000
                020
                030
                040
                050
                060
                070
                000

            302
            312
            322
            332
            342
            352
            362
            372
            303
            304
            314
            324
            334
            344
            354
            364
            374
            315
            000

        006
        016
        026
        036
        046
        056
        066
        076
        323
        333
        306
        316
        326
        336
        346
        356
        366
        376
        000

007322  300
        310
        320
        330
        340
        350
        360
        370
        311
        000
        001
        021
        041
        061
        042
        052
        062
        072
        000
```



A KIM Binary Calculator

BY JOHN EATON
1126 N 2nd
Vincennes, Ind. 47591
Copyright 1977

Binary math has always been a stumbling block for anyone attempting to study the inner workings of a computer. No matter how often you work problems in binary, octal or hexadecimal, you can never work as efficiently as you can in decimal. Most of us when confronted with a problem involving other number systems will have to reach for a pencil and paper in order to obtain a result.

But now there is an easier way to solve binary math problems. With this software your KIM-1 or other 6502 system can do all the work and give you the result in a form that you can understand. The program is a machine language routine that will fit in less than 1K of memory. It communicates with the user through any ASCII I/O device.

The program will work with binary numbers up to 16 bits. They can be inputted to the program in decimal, hexadecimal, octal, binary or as an ASCII literal character. The user may specify whether the results are displayed in decimal, hexadecimal, octal or binary. With these features the program is able to convert numbers between different bases.

Several logic functions are available. The program will take any two 16 bit numbers and perform the operations of AND, OR, NOR, EOR and NAND. You may also perform the arithmetic operations of addition and subtraction. The inverse and two's complement of any 16 bit number can be found.

There are two rotate commands available. The rotate commands treat two 16 bit numbers as a single 32 bit number and can shift it left or right up to 16 bits.

Using the Routines

The program requires a MOS 6502 based system and *must* have a rotate right instruction. If you have one of the earlier processors then you will have to modify the code around address 327E to perform eight ROL's for each ROR. The program is assembled starting at address 3000 but is small enough that it will fit in a KIM-1 without any additional memory. The starting addresses for the user's I/O routines go in the first 9 bytes. You will need a single character input routine, single character output routine and a routine that performs a carriage return and linefeed.

Start the program at address 3010. It should respond with a carriage return and linefeed. The first thing that you should do is set the display mode. Type either DEC, HEX, OCT, or BIN depending on what mode you want the result to be displayed in. The mode that you type will remain in effect until you specify a new one. The program will respond by displaying the current value of its display register.

The ENT command is used to enter numbers into the display register. Type ENT and a space followed by a number. If the number is other than a decimal number, it must be preceded by a % for binary, @ for octal, $ for hex and " for an ASCII literal. ASCII literals must end with a ". The number must be an integer and may be preceded by a "–" if the two's complement of the number is wanted. Once a number is displayed by the program you can use it in various ways. If you type INV or TSC then the inverse or two's complement of the number is displayed.

Several operations require the use of a second number to produce a result. Logical functions are available by typing AND, OR, EOR, NAND or NOR. Each of these commands must be followed by a space and a second number. The number can be in the same format as in the ENT command. The result of the operation will be displayed. The commands ADD and SUB may be used in a similar manner for integer arithmetic.

The rotate commands require a special format. To use them type ROL or ROR followed by an integer number in parenthesis. This number specifies the number of bits that are to be shifted. The parenthesis must be immediately followed by a 16 bit number that is to be shifted into the display to replace the bits that are shifted out.

The program does all binary/decimal conversions with counting loops. It's very easy to write but extremely slow. The worst case time for any conversion is 5 seconds so if you want fast results use a non-decimal display mode.

Isn't it nice to now have a computer that can solve problems that you never would have to solve if you didn't have a computer?

```
0000            DISL                    Display
0001            DISH                    register
0002            DECL                    dec/bin conversion
0003            DECH                    register
0004            BUFL                    Buffer 3 Bytes
0005            BUFH
0006            BUFO
0007            POIN                    input line pointer
0008            MODE                    display mode
0009            SHIFT                   # of bits per char
000A            MASK
000B            SIGN
000C            VEB
000D            VEB+1
000E            VEB+2
0010            LR                      line register 32 bytes

3000 4C 00 00   INPUT   JMP CHAIN       user input routine
3003 4C 00 00   OTPUT   JMP CHAROT      user output routine
3006 4C 00 00   CRLF    JMP CARLF       user carret/lf routine
3009 EA                 NOP
300A 20 03 30   OUT     JSR OTPUT       display number
300D 68         DISPLA  PLA
300E D0 FA              BNE OUT         characters on stack
3010 A9 01      START   LDA#01          Start program here******
3012 85 07              STA POIN
3014 20 06 30           JSR CRLF
3017 C6 07      BACK    DEC POIN
3019 20 00 30   LOOP    JSR INPUT       fetch character in Accum
301C C9 08              CMP#08          backspace?
301E F0 F7              BEQ BACK        yes
3020 A6 07              LDX POIN
3022 95 10              STA LR,X        save character in pg 0
3024 E6 07              INC POIN
3026 C9 0D              CMP#0D          carriage return?
3028 D0 EF              BNE LOOP
302A 20 06 30           JSR CRLF
302D D8                 CLD
302E A0 04              LDY#04
3030 20 95 30           JSR LOAD        load number into buffer
3033 20 D7 30           JSR OP          perform operation
3036 A2 01              LDX#01
3038 B5 00      MOVE    LDA DISL,X      move display register
303A 95 04              STA BUFL,X      into buffer and invert
303C 49 FF              EOR#FF          into dec/bin conversion
303E 95 02              STA DECL,X      register
3040 CA                 DEX
3041 10 F5              BPL MOVE
3043 A9 00              LDA#00
3045 85 06              STA BUFO
3047 48                 PHA             end of stack storage
```

```
3048 A6 08          LDX MODE      display mode
304A D0 27          BNE DIS1      non decimal
304C F8             SED
304D 20 D9 31       JSR SET9      Buffer=999999
3050 38       DEC1  SEC           add 1 to buffer
3051 A0 02          LDY#02
3053 A2 00          LDX#00
3055 B5 04    DEC2  LDA BUFL,X
3057 69 00          ADC#00
3059 95 04          STA BUFL,X
305B E8             INX
305C 88             DEY
305D 10 F6          BPL DEC2
305F E6 02          INC DECL      add 1 to DECL
3061 D0 ED          BNE DEC1
3063 E6 03    DEC3  INC DECH
3065 F0 06          BEQ DEC4      finished counting
3067 20 6A 32       JSR ADD256    add 256 to buffer
306A 4C 63 30       JMP DEC3

306D A0 14    DEC4  LDY#14        lenght= 20 bits
306F A2 04          LDX#04        bits/char= 4
3071 D0 03          BNE DIS2      unconditional branch

3073 BC 65 32 DIS1  LDY TAB2-1,X  fetch lenght
3076 86 09    DIS2  STX SHIFT
3078 D8             CLD
3079 A6 09    DIS3  LDX SHIFT     fetch LSD and save on stack
307B A9 80          LDA#80
307D 20 7E 32 DIS4  JSR ROTATE
3080 88             DEY
3081 30 8A          BMI DISPLA    display all characters
3083 CA             DEX
3084 D0 F7          BNE DIS4
3086 4A       DIS5  LSR A         char in A,shift to lo end
3087 D0 FD          BCC DIS5
3089 C9 0A          CMP#0A        convert to ASCII
308B 90 02          BCC DIS6
308D 69 06          ADC#06
308F 69 30    DIS6  ADC#30
3091 48             PHA           save char on stack
3092 4C 79 30       JMP DIS3

3095 A2 04    LOAD  LDX#04        number load routine
3097 B9 10 00 LOAD1 LDA LR,Y      Y points to start
309A DD D2 30       CMP LOAD8,X   match to %,",@,$

309D F0 15          BEQ LOAD4     match found
309F CA             DEX
30A0 D0 F5          BNE LOAD1
30A2 A2 04          LDX#04        decimal input
30A4 20 97 31       JSR LOADT     input number
30A7 20 E6 31       JSR DECHEX    convert to hex
30AA A5 0B    LOAD2 LDA SIGN      complement if "-"
30AC C9 2D          CMP#"-"
30AE D0 03          BNE LOAD3
30B0 20 58 31       JSR TSCB
30B3 60       LOAD3 RTS
30B4 E0 02    LOAD4 CPX#02        test for ASCII literal
30B6 D0 14          BNE LOAD7
30B8 C8       LOAD5 INY           ASCII section
30B9 A2 07          LDX#07
30BB B9 10 00       LDA LR,Y
30BE C9 22          CMP#"""       last character
30C0 F0 F1          BEQ LOAD3
30C2 0A       LOAD6 ASL A         shift char into buff
30C3 26 04          ROL BUFL
30C5 26 05          ROL BUFH
30C7 CA             DEX
30C8 10 F8          BPL LOAD6
30CA 30 EC          BMI LOAD5
30CC C8       LOAD7 INY           hex,oct and bin
30CD 20 97 31       JSR LOADT
30D0 4C AA 30       JMP LOAD2
30D3 25 22 40 LOAD8+1 .BYTE %,",@,
30D6 24             .BYTE $

30D7 98       OP    TYA           look up command and
30D8 48             PHA           perform
30D9 A0 00          LDY#00
30DB A2 00          LDX#00
30DD 20 F0 30       JSR OP1       match 1st letter
30E0 20 F0 30       JSR OP1       match 2nd letter
30E3 20 F0 30       JSR OP1       match 3rd letter
30E6 68             PLA
30E7 A8             TAY
30E8 A9 31          LDA#31        address hi byte
30EA 48             PHA
30EB B9 16 32       LDA TAB1,X    address lo byte-1
30EE 48             PHA
30EF 60             RTS           jump to operation
30F0 B9 16 32 OP1   LDA TAB1,X    search table for match
30F3 D9 10 00       CMP LR,Y
30F6 F0 08          BEQ OP2       match found
30F8 E8             INX           next entry
30F9 E8             INX
30FA E8             INX
30FB E8             INX
30FC C9 FF          CMP#FF        end of table?
30FE D0 F0          BNE OP1
3100 E8       OP2   INX
3101 C8             INY
3102 60             RTS

3103 20 58 31 SUB   JSR TSCB      routines that perform
3106 A9 75    ADD   LDA#75        operations
3108 D0 19          BNE EXEC
310A A9 31    NAD   LDA#31
310C 48             PHA
310D A9 13          LDA#13        INV address-1
310F 48             PHA
3110 A9 35    AND   LDA#35
3112 D0 0F          BNE EXEC
3114 20 DD 31 INV   JSR SETF
3117 A9 55    EOR   LDA#55
3119 D0 08          BNE EXEC
311B A9 31    NOR   LDA#31
311D 48             PHA
311E A9 13          LDA#13        INV address-1
3120 48             PHA
3121 A9 15    ORR   LDA#15
3123 85 0C    EXEC  STA VEB
3125 A9 00          LDA#00
3127 85 0D          STA VEB+1
3129 A9 60          LDA#60
312B 85 0E          STA VEB+2
312D 18             CLC
312E A2 00          LDX#00
3130 A0 01          LDY#01
3132 B5 04    EXEC1 LDA BUFL,X
3134 20 0C 00       JSR VEB
3137 95 00          STA DISL,X
3139 E8             INX
313A 88             DEY
313B 10 F5          BPL EXEC1
313D 60             RTS

313E A5 04    ENT   LDA BUFL
3140 85 00          STA DISL
3142 A5 05          LDA BUFH
3144 85 01          STA DISH
3146 60             RTS

3147 A9 01    BIN   LDA#01
3149 D0 0A          BNE MOD
314B A9 03    OCT   LDA#03
314D D0 06          BNE MOD
314F A9 04    HEX   LDA#04
3151 D0 02          BNE MOD
3153 A9 00    DEC   LDA#00
3155 85 08    MOD   STA MODE
3157 60             RTS

3158 A2 04    TSCB  LDX#04        two's comp of buff
315A D0 02          BNE TSC1
315C A2 00    TSC   LDX#00        two's comp of display
315E 38       TSC1  SEC
315F 98             TYA
3160 48             PHA
3161 A0 01          LDY#01
3163 B5 00    TSC2  LDA DISL,X
3165 49 FF          EOR#FF        invert byte
3167 69 00          ADC#00        add 1
3169 95 00          STA DISL,X
316B E8             INX
316C 88             DEY
316D 10 F4          BPL TSC2
316F 68             PLA
3170 A8             TAY
3171 60             RTS

3172 20 58 31 ROR   JSR TSCB      rotate right by rotating
3175 A9 1F          LDA#1F        left 32-n times
3177 25 04          AND BUFL
3179 85 04          STA BUFL
317B C6 04    ROL   DEC BUFL
317D A9 3F          LDA#3F
317F 25 04          AND BUFL
3181 48             PHA
3182 C8             INY
3183 20 95 30       JSR LOAD
3186 68             PLA
3187 A8             TAY
3188 A5 01    ROL1  LDA DISH      set C to bit 7 of DISH
318A 0A             ASL A
318B 26 04          ROL BUFL      shift registers left
318D 26 05          ROL BUFH      Y times
318F 26 00          ROL DISL
3191 26 01          ROL DISH
3193 88             DEY
3194 10 F2          BPL ROL1
3196 60             RTS

3197 B9 10 00 LOADT LDA LR,Y      Load routine for all
319A 85 0B          STA SIGN      but ASCII
319C C9 2D          CMP#"-"       test for negative
319E D0 01          BNE LOADT1
31A0 C8             INY
31A1 86 09    LOADT1 STX SHIFT    # of bits per char
31A3 BD D0 31       LDA LOADT5,X
31A6 85 0A          STA MASK
31A8 20 D5 31       JSR SETZ      buff=000000
31AB B9 10 00 LOADT2 LDA LR,Y
31AE C9 30          CMP#30
31B0 90 1E          BCC LOADT5    ending character
31B2 C9 3A          CMP#3A
```

TINY BASIC FOR COSMAC

Dear Sir:

I have read letters in your magazine (and others) seeking a source of basic for the RCA COSMAC CDP 1802 microprocessor. Infinite Incorporated offers tiny basic 3K on paper tape, listing or cassette tape. Their address is: Infinite Incorporated, 1924 Waverley Place, Melbourne, FL 32901.

Yours truly,

Dana S. Majors

28 La Cienega Way
Yuba City, CA 95991

'WHIZ' LOADER/DUMPER

news release *Received: 77 Dec 12*

Shifting Sands Microcomputer Products Corporation has begun shipping WHIZ, a software program for recording and loading MC6800 based programs on the SWTPC M68 computer with MIKBUG and the SWTPC AC-30 cassette interface. WHIZ operates at nine times the standard MIKBUG format 300 baud speed and three times that of the binary format. This higher speed capability is provided without any modifications to the standard computer or AC-30. Load a 1K program in 14 seconds, 4K in 48 seconds, and 8K BASIC in 85 seconds. WHIZ includes an interactive front-end which allows the specification of a header and program start address to be placed on tape for read-back later. For $15.95, WHIZ is supplied on Kansas City standard cassette in MIKBUG format and includes a built-in relocater to place your own copy of WHIZ in RAM memory wherever you wish. WHIZ is optionally available in 2708 EPROM from Shifting Sands Microcomputer Products Corporation, Box 441, Fairborn, OH 45324.

DATAFORCE MAINTENANCE SERVICE FOR COMPUTER VENDORS

News Release *Received: 77 Sept 1*

A national service organization is seeking the business of independent vendors of small business computers including computer stores. The general trouble shooting of machines in this market has been generally viewed as unprofitable and the major drawback to handling the bottom end of small computer lines. But the Dataforce Service Company is now offering to perform warranty or out of warranty work and field service for vendors of small computers and computer kits anywhere in the United States.

Dataforce has 115 service locations in the country and is expanding with the goal of having a service outlet within 50 miles of every computer sold.

A central diagnostic center is at Dataforce's headquarters, 2807-F Oregon Court, Torrance, CA 90503. It opens at 5 a.m. for trouble calls from the East Coast and reamins open until 5 p.m., Monday through Friday. The company stresses its desire to help first time computer users with the simple problems arising from unfamiliarity with their equipment as well as the sophisticated user who needs a highly skilled diagnosis of a computer problem.

Dataforce is first contacted by calling its Torrance diagnostic center. In California the number is (213) 328-2950. Outside the state, call (800) 421-1858. A technician qualifies the nature of the problem and takes the necessary steps to resolve it. When on-site assistance is needed, Dataforce dispatches parts and test equipment from the closest service organization to the trouble.

The company is wholly owned by Randal Data Systems, Inc., but operates independently. It was created to meet the requirements of Randal distributors, outlets for the company's Link 100, Link 200, and Link 500 series, small business computers.

```
31B4 90 06          BCC LOADT3      number
31B6 C9 47          CMP#47          ending character
31B8 B0 16          BCS LOADT5
31BA E9 06          SBC#06
31BC 25 0A    LOADT3 AND MASK       clear unneeded bits
31BE A6 09          LDX SHIFT
31C0 06 04    LOADT4 ASL BUFL       shift buffer X times
31C2 26 05          ROL BUFH
31C4 26 06          ROL BUFO
31C6 CA             DEX
31C7 D0 F7          BNE LOADT4
31C9 05 04          ORA BUFL        combine new character
31CB 85 04          STA BUFL
31CD C8             INY
31CE D0 DB          BNE LOADT2      unconditional
31D0 60       LOADT5  RTS
31D1 01 00 07       .BYTE 01,00,07
31D4 0F             .BYTE 0F

31D5 A9 00    SETZ  LDA#00
31D7 F0 06          BEQ SET
31D9 A9 99    SET9  LDA#99
31DB D0 02          BNE SET
31DD A9 FF    SETF  LDA#FF
31DF 85 04    SET   STA BUFL
31E1 85 05          STA BUFH
31E3 85 06          STA BUFO
31E5 60             RTS

31E6 98       DECHEX TYA            decimal to binary conv
31E7 48             PHA
31E8 F8             SED
31E9 A9 FF          LDA#FF
31EB 85 02          STA DECL
31ED 85 03          STA DECH
31EF E6 02    DH1   INC DECL
31F1 D0 02          BNE DH2
31F3 E6 03          INC DECH
31F5 A2 00    DH2   LDX#00
31F7 A0 02          LDY#02
31F9 18             CLC
31FA B5 04    DH3   LDA BUFL,X
31FC E9 00          SBC#00
31FE 95 04          STA BUFL,X
3200 E8             INX
3201 88             DEY
3202 10 F6          BPL DH3         keep looping until
3204 B0 E9          BCS DH1         buffer is 0000
3206 D8             CLD
```

```
3207 A5 02          LDA DECL        move into buffer
3209 85 04          STA BUFL
320B A5 03          LDA DECH
320D 85 05          STA BUFH
320F A9 00          LDA#00
3211 85 06          STA BUFO
3213 68             PLA
3214 A8             TAY
3215 60             RTS

3216 41 44 44 05    TAB1 consists of 3 ASCII chars and the
321A 41 4E 44 0F         lo byte-1 of the operations add-
321E 42 49 4E 46         ress. Hi byte = 31.
3222 44 45 43 52
3226 45 4E 54 3D
322A 45 4F 52 16
322E 48 45 58 4E
3232 49 4E 56 13
3236 4E 41 44 09
323A 4E 4F 52 1A
323E 4F 43 54 4A
3242 4F 52 52 20
3246 52 4F 4C 7A
324A 52 4F 52 71
324E 53 55 42 02
3252 54 53 43 5B
3256 FF FF FF 3C
325A FF FF FF 3C
325E FF FF FF 3C
3262 FF FF FF 3C
3266 10 10 12 10    TAB2+1 consists of lenghts for non dec dis

326A 18       ADD256 CLC            add 256 to buffer
326B A5 04          LDA BUFL
326D 69 56          ADC#56
326F 85 04          STA BUFL
3271 A5 05          LDA BUFH
3273 69 02          ADC#02
3275 85 05          STA BUFH
3277 A5 06          LDA BUFO
3279 69 00          ADC#00
327B 85 06          STA BUFO
327D 60             RTS

327E 46 06    ROTATE LSR BUFO       perform rotate rights
3280 66 05          ROR BUFH
3282 66 04          ROR BUFL
3284 6A             ROR A
3285 60             RTS
```

Status of the UCSD PASCAL Project

BY KENNETH L. BOWLES
Director IIS

This is a brief report on the current status of the UCSD PASCAL project intended to answer the questions of the hundreds of people who have been writing to us or calling by telephone. It is our intention eventually to reach a steady state in which we can afford to have full time help capable of responding to such inquiries. For the present, we have to apologize once again to those who may have been kept too long waiting for replies.

Kenneth L. Bowles
Institute for Information Systems
University of California, San Diego
La Jolla, CA 92093 (714) 452-4526

Nature of the Project

The project is one of the principal activities of the Institute for Information Systems (IIS). Like other "Organized Research Units", IIS is operated primarily to provide resources and activities within which students and faculty can conduct research and development projects. Within the range of such activities, projects may support instruction and other public services though the more usual activities of an ORU involve only basic research.

Under the IIS we have developed a major software system for stand-alone microcomputers based on the PASCAL language. The initial reason for developing the system was to support instruction activities at UCSD. However, the system is designed for general purpose use, particularly for the development of interactive software, and for software development in general. The system has matured sufficiently that we are distributing copies to outside users at a $200 fee which pays for some of the student part-time assistants who provide support to users and maintain the software. Under prevailing University policies, we are not attempting to recover capital costs from the fees paid by users of our software package. However, a number of interested industrial firms have provided assistance to further the project through unrestricted grants to the Regents of the University of California marked for use by our project. These grants are our principal source of operating funds at the present time.

Since the PASCAL based software system was developed with the intent of supporting long term instructional projects, we have placed very high emphasis on machine independence. We expect the repertoire of instructional software developed to use the underlying system to grow very large. The development costs for the instructional software will eventually dwarf the costs of the hardware on which it operates. Since the industry is introducing new microcomputer designs at a rapid rate, we wanted to be able to move the entire software repertoire to new machines with a minimum of effort. As will be detailed in later sections of this note, our system is now running on 5 dis-similar processors, with more planned in the relatively near future. We are using the Digital Equipment LSI-11 for teaching. Versions for the 8080 and Z80 microprocessors are operational and will be ready to distribute on or about 1 January, 1978.

We intend to continue promoting the use of our PASCAL-based system on as many popular microprocessors as practical for two reasons. First, this should provide IIS with a source of continuing income to pay for student projects. Second, PASCAL with extensions is a superior language for system programming, and we believe that it is in the public interest to assist in the current effort of many people and institutions to promote wider use of PASCAL in place of some of the earlier high level languages. Though PASCAL may have some shortcomings for specific applications when compared to specific proprietary languages, we regard it as by far the best general purpose language now in the public domain.

Our current Research and Development interests include:

a. Methods of making large software systems like ours more readily transportable to new processor architectures.

b. The use of microcomputers as intelligent communications devices to assist humans to work together even when located thousands of miles apart. This interest will eventually involve us in a variety of complex software issues. In the near term it will provide us with an efficient method of supporting users of our software system who are remote from UCSD.

c. Joint use of microcomputers and Keller's Personalized System of Instruction (PSI) as a means of offering high quality college level mass education at lower costs per enrolled student than associated with conventional methods. A published introductory textbook on problem solving using PASCAL, and a library of automated quizzes and record keeping software to go with the textbook, are available to others as a first step in this direction.

d. Exploration of possibilities and software problems associated with new hardware devices or architectures adaptable to the purposes already described. Examples include video disks, low cost X-Y input devices, and low cost strategies for interconnecting many semi-independent microcomputers.

Partly as a matter of self preservation, we have become interested in the problem of *standards* for the PASCAL language. The United States Defense Department and many large industrial corporations have recently decided to use PASCAL as a base language which they would extend, and possibly alter, to create system implementation langauges. Although almost every organization has chosen to extend or alter in slightly different ways, we have found that the intent portrayed in most instances is very similar. In our own implementation, we too found it necessary to extend PASCAL, and in very minor ways to alter the base language as described in Niklaus Wirth's widely read "Report"on the langauge (see Jensen, K. and N. Wirth, *PASCAL User Manual and Report,* Springer Verlag, 1975). We, and many others in the PASCAL User Group, are very much concerned that all this extension and alteration activity will result in PASCAL going the way of BASIC for which hundreds of dialects are in common use. We believe that a chance still exists to gain consensus on a substantial family of PASCAL extensions for system programming, provided that this can be brought about within the next 6 to 12 months. Unless someone does so before us, we intend to convene a summer workshop for representatives of some of the major using organizations in the hope that such a consensus can be reached.

Another ancillary activity of the project has been continuing search for low cost microcomputer hardware of high quality for use in educational institutions – particularly ours. We have been advising and collaborating with EDUCOM regarding establishment of quantity purchase discounts for stand-alone microcomputers. The first microcomputer to be included in

the EDUCOM discount program is the Terak Corporation 8510A, which is based on the Digital Equipment LSI-11. For the current market, the Terak unit's price of $5500 to EDUCOM member institutions is highly competitive. Nevertheless, the rate of new announcements from the industry continues very high, and we believe that it is all but impossible to predict what hardware will provide the best cost/performance tradeoff for as long as even one year in advance. Of necessity, our search has concentrated on stand-alone microcomputers with graphic display capabilities, and with enough main memory and secondary storage to handle the extensive software and course materials with which we are working. We welcome inputs on this subject from other institutions, or from any vendor, and endeavor to keep EDUCOM informed of opportunities that seem advantageous. In addition to educational and communications applications, we are interested in word processing and business applications of the same machines.

The following sections of this status report contain brief detailed summaries covering most of the topics just enumerated. If we haven't answered your questions yet, please try again with a phone call or letter. For those who already have our software system in use, we will soon be providing an automatic Tele-Mail facility on a dial-in basis. This should improve dramatically our ability to keep you informed and to respond when software difficulties arise.

The PASCAL based Software System

Thus far, users who have received our first released system have copies of version I.3, which was completed in mid August this year. We have ourselves been using version I.3c since early October. By the end of the December academic break, we hope to have a version I.4 available for distribution. The most significant generally useful addition since the I.3 release has been the screen-oriented editor. A major package for preparation of CAI programs, following the general philosophy of the University of California Irvine Physics Computer Development Project (PCDP), has been placed in operation on the Terak 8510A microcomputer. Except for some graphics materials within this package, it can be used on a wide variety of CRT screen display devices. Software more specifically oriented to the Terak machine is also available, and includes a character set editor (for the soft character generator), and a bookkeeping package for keeping track of student progress in a large Keller Plan (PSI) class.

The software system is currently executed in a pseudo machine interpreter, which emulates a hypothetical real machine designed to handle PASCAL constructs efficiently. Our pseudo machine is similar in concept to the P-machine distributed by Wirth's group at Zurich, but we have made extensive changes to compress the PASCAL object code into a much smaller space than possible with the Zurich interpreter. The interpreter, and run-time support routines, currently occupy about 8K bytes of main memory. The interpreter is in the native machine language of the host machine, and thus far has been coded by hand using the host's assembly language. All other code in the system is written in extended PASCAL.

While the interpreted object code runs roughly five times slower than native code for the host machine, several factors allow our large system programs to run substantially faster than this would indicate. The strategy of code compression makes it possible to run relatively large programs without time consuming overlays. For example, the complete compiler occupies 20K bytes in a single overlay. Since the system is designed for frequent compile/go cycles associated with instruction, we have added several built-in procedures and functions to handle low level operations needed frequently by the compiler. As a result, the compiler translates PASCAL source code at about 650 lines per minute on an LSI-11 with its clock set to 2.2 MHz. On a 4 MHz Z80, the compile speed will be slightly faster than this.

Interpreter based versions of our system are now running on 5 distinct processors, and two others are close to completion. Those operating include DEC PDP-11's ranging from the LSI-11 to the 11/45, using either floppy disks or RK05 disks for secondary storage. Versions for the 8080 and Z80 are operating in our laboratory, but more of that later. Sperry Univac Minicomputer Operations at Irvine is using the system on the V-75 and related machines. Another group at UCSD has the system running on the Nanodata QM-1. With support from General Automation, a version is close to completion on the GA-440 family of machines. National Semiconductor has an implementation nearly completed on the PACE microcomputer.

The principal modules of the system as it will be distributed in the I.4 release include the following:

PASCAL compiler
File manager (capabilities similar to DEC's PIP)
Screen oriented editor (cursor positioning, immediate updates)
Line oriented editor (similar to DEC's RT-11 Editor)
Debugger (single line execution, reference to variable contents)
SETUP program (adapt system to most ASCII terminals)
BASIC compiler (ANSI standard plus strings)
Operating System and user command interpreter
PASCAL pseudo machine interpreter
Linker program (for linking independently compiled program segments)
Desk Calculator utility program

Users of the Terak 8510A may, on request, also receive copies of the CAI package, and automated quizzes for the introductory textbook, as well as the bookkeeping package.

Documents describing all of the above are available, and part of the release, but not all documents can be considered complete at this time. We distribute source and object code files on separate floppy disks formatted to be compatible with the IBM 3740 standard, with 512 byte blocks laid out in alternate 128 bytes sectors according to DEC's standard. We have occasionally sent copies recorded directly on disk packs for the RK05 drives. All other media are painful or impossible for us to handle, and no promises are made to use them. Those who order the full $200 release package will be sent both the documents, and printed lisitings of the source programs. Copies of the descriptive documents, amounting to approximately 150 pages, may be ordered at $10 each (checks payable to the "Regents of the University of California") to cover printing and handling costs.

Minimum Configuration

In order to use the compiler, you need a total of at least 48K bytes of main memory, including the 8K bytes assigned to the interpreter. We use 56K bytes. Ideally, the interpreter should be completed re-entrant and thus it should be possible to operate the interpreter from Read Only Memory. To date, the ideal has not quite been achieved, as none of our sponsors has yet insisted on that feature.

At present, the system we use with students contains several built-in functions not needed for system development. The aggregate size of these functions is large enough to prevent compiling the compiler itself, or the operating system, even on a 56K byte system. Accordingly, we currently have two versions of the system, one for students, and one for system development. Within the next few months, we plan to add a means of configuring general purpose libraries for the system, and by that means expect to be able to return to a single

version for all purposes. That single version should be practical to use in less than 48K bytes for some purposes.

If you intend to compile on one microcomputer, and to execute object routines on others, the others can get by with as little as 16K bytes of main memory if the operating system is not used. The resident portion of the operating system occupies about 8K bytes itself. This will undoubtedly be reduced as part of the libraries project.

The system is designed to be used with standard IBM compatible floppy disks. Clearly it can be used with other varieties of floppy disks, or with other secondary storage media, with appropriate I/O drivers. The I/O drivers have proven to be one of our principal bottlenecks, and we make no promises in advance about supporting other devices. For DEC PDP-11 machines, the floppy disk drivers are assumed to be compatible with the RX-11, or with the Terak 8510A drives. Hard disks are assumed to be compatible with the RK05.

The system is normally supplied with the assumption that the user has a simple line-oriented ASCII terminal. The SETUP program can be used to configure control codes for more appropriate use of most CRT terminals. Copies of the system supplied to users of the Terak 8510A make fairly extensive use of the special graphics and character generator facilities of that machine.

8080 and Z80 Versions

The Z80 version is now running on the Tektronix 8002 Microprocessor Development Aid system, for which Tektronix has supplied substantial support to the project. The 8080 version uses virtually the same source code as is used on the Z80, with conditional assembly altering certain passages in the source to substitute for a few of the extended Z80 instructions that proved useful.

Release of the 8080/Z80 version of the system of other machines has been held up primarily because of the awkwardness of handling I/O. We currently have a Zilog Development System, a Processor Technology SOL system, and a Computer Power and Light COMPAL-80 system. The floppy disk provisions for each of these machines is non-standard. As a result, we have been forced to down-load programs via serial interfaces to get from the LSI-11 host machines used for development over to the new 8080 or Z80 based host. This has proven to be a very time consuming process, and a serious bottleneck in our work. Moreover, we are somewhat amazed to find that the *assembly* of large programs on these machines runs almost a factor of ten *slower than compilation* of PASCAL programs that carry out similar tasks! Clearly, something has to give if we are to reach the objective of distributing PASCAL systems for more than a few 8080 and Z80 based machines.

The solution to this problem that we now plan to use is based on the extensive market penetration of an operating system called CP/M, which is a product of Digital Research Inc. We have talked with many OEM and hobbyist users of the 8080 and Z80 who wanted to know when we would have the PASCAL system operating under CP/M. We then learned that CP/M is distributed in a package which assumes that most users will write their own I/O drivers. In effect, CP/M establishes a quasi standard for the interface between an 8080/Z80 operating system and its I/O drivers. With thousands of copies working in the field, CP/M seems to be far ahead of the field in this area. Accordingly, we have decided to release the UCSD PASCAL System for 8080 and Z80 users in a form that will work with I/O drivers and bootstrap loaders developed for use with CP/M. This does not mean that our package will run under CP/M. However, if CP/M runs on your machine it should be relatively easy to install the PASCAL system on that machine. We have been in contact with Digital Research on this concept, and they have offered to cooperate. If you do not have CP/M for your machine, the implementation package may be obtained from Digital Research Inc., Box 579, Pacific Grove, CA 93950 for $70. Since CP/M has been implemented on a very wide variety of 8080 and Z80 based machines, there is a high probability that CP/M I/O drivers are already available from Digital Research or someone else for your machine.

Alteration of our present interpreter to match the CP/M I/O calling conventions has proven to be very simple, at least on paper. We expect that some implementors of CP/M will have installed standard console input routines which automatically echo to the standard console printer or display device. This will necessitate a change, since our system uses both echoing and non-echoing console input. At this writing, the exact method to be used is under discussion. Barring some unforseen calamity, copies of our system designed to run with CP/M I/O drivers should be ready for distribution by early January, 1978. The distribution medium will be IBM compatible floppy disks formatted in a manner yet to be finally specified. We will undertake to transform the system for other media and other formats, in general, only if a copy of the necessary hardware is available in our laboratory, and only if funds are available to pay for the extra conversion work.

For many of the 8080 based machines we have seen, the most practical way to install our system will be to use 48K bytes of RAM augmented with 8K bytes of ROM for the interpreter. Any additional RAM or ROM required by the host processor system will also be needed.

PASCAL Extensions and Alterations

We have attempted to implement faithfully as much as possible of PASCAL as defined in Jensen & Wirth's User Manual and Report. The principal extensions to PASCAL embodied in our system are related to STRING variables, Turtle Graphics, handling of disk files, Segment (overlay) Procedures, and several functions for support of the system itself. Alterations include a prohibition against passing procedure, the addition of EXIT (<procedurename>) to effect a normal exit from the procedure named in the parameter, and a change in READ applying to the interactive INPUT and KEYBOARD files. Further details than given in this section are given in our system release documents.

Type STRING is a pre-declared record containing a character count followed by a packed array of characters. Built-in procedures and functions include LENGTH, POS(ition), INSERT, DELETE, COPY (i.e. extract), CONCATenate, SCAN, FILLCHAR, MOVERIGHT, and MOVELEFT. The last four of these also operate on conventional packed arrays of characters.

Turtle Graphics describes a technique originated by Seymour Papert of MIT in which one can either MOVE a cursor (called the "turtle") an arbitrary number of screen units in the current pointing direction, or TURN an arbitrary number of degrees at the current position. A PENCOLOR procedure allows the line drawn by a MOVE to be either WHITE, BLACK, or NONE.

The disk file extensions allow working with fixed length logical records corresponding to any legal <type>, which might typically be a RECORD data structure. GET and PUT operate normally through a window variable of the same <type>. OPENNEW creates a new file, OPENOLD opens a pre-existing file, and CLOSE allows saving or purging a file. SEEK (which will be distributed with the I.4 system for the first time) allows random access to logical records within a file.

SEGMENT Procedures are separately compiled and then linked into the host program using the LINKER. A Segment procedure is only loaded into main memory when it is entered for the first time, and its memory space is deallocated

upon exit from the first invocation.

READ(INPUT , X) is defined by Wirth as X: =INPUT↑; GET(INPUT); which we find to be extremely awkward for interactive use. Our solution is to place the implied GET before the implied assignment in the case of interactive files of type TEXT. READ operates as defined in Wirth's Report for other TEXT files.

PACKED records on our system which fit within 16 bit fields are automatically packed and unpacked without explicit action by the programmer.

Introductory PASCAL Course and Textbook

Many of those inquiring about our system have heard about it through having seen the textbook "Microcomputer Problem Solving Using PASCAL" by the author of this note, published this fall by Springer Verlag. If you haven't seen a copy, they may be obtained from Springer at 175 Fifth Ave., New York City, NY 10010.

The book is the basis for teaching the large attendance introductory computer science course at UCSD. This course comes close to matching the specifications for course CS1 in the recently published curriculum recommendations from ACM's SIGCSE. The approach is non-numerical as far as practical, as a tactic to reach the many students who come to us with inadequate preparation in high school mathematics. The problem solving and programming approach taught is the same as we would teach even if all the problem sets were mathematics oriented. Because many problem examples and illustrations use our string and graphics extensions to PASCAL, the textbook currently assumes that the student will have access to a computer which runs under the UCSD PASCAL system. We will be glad to discuss the possibility of conversion to other software systems, but have very limited resources to apply to such conversions. There are several stand-alone microcomputers now being sold in large quantities on which our system would run, given a small conversion effort, and we would welcome support funds to pay for such conversions.

Software in the form of automated quizzes is available with our system release for those who may wish to teach using the textbook. Each chapter in the book has a list of study goals for the students to achieve. Wherever appropriate, the quizzes test for mastery of the topics enumerated in the goals lists. The quiz programs have been implemented using a set of CAI primitive routines patterned after the well known DIALOG CAI system developed at U.C. Irvine by Alfred Bork and his colleagues.

The introductory course is taught using Keller's Personalized System of Instruction (PSI). PSI has been found to be a more successful method of instruction than any other method commonly used in universities and colleges. This success is achieved, almost completely without conventional lectures, by using experienced students as Learning Assistants called "proctors". The characteristics of this method make us believe that it is possible to offer this course, or others constructed along the same lines, on a packaged basis for use at other institutions. A separate paper describing this possibility in detail called "Microcomputer Based Mass Education" is available from the writer of this status report.

Tele-Mail User Support Facility

We have reached the point where it will be possible for us to begin operating a dial-in computer "mailbox" by early in the winter quarter. We have been using the Terak 8510A machines occasionally as intelligent terminals for exchanging messages via the large B6700 computer operated by the campus computer center. Our own Tele-Mail facility will use its own single telephone number reachable directly from the national dialed telephone network, or internally via the California state government telephone network. Mail subscribers to our software release will be notified when this mailbox facility is ready to be used.

The mailbox will be operated primarily to serve users of our software system. It will provide notices of recent bug corrections, down-loading of program files (either source or object) where appropriate, notices on new additions to the software and new machines on which implementations have been completed, and other useful information from us to the users. It will also serve as a means for us to collect messages from specific users, and to answer them expeditiously, without the hassle of both parties having to be at their telephones at the same time.

Through the use of block transfer software, the mailbox will make relatively efficient use of the dialed telephone network. We would like to begin immediately by offering a dial-in port at 1200 bits per second. However, the present state of confusion in the industry at that speed (which is the fastest one can use with acoustic couplers) leads us to move cautiously. We can and will install a port at 300 bits per second using the standard Bell 103A equivalent conventions. The system will answer an incoming call from an ordinary terminal by providing a brief summary of recent developments. It will otherwise expect a "handshake" from a special file transfer program that we will provide to users of our software package. This program will be the means of interchange based on efficient transfer of messages in the form of complete files. If you wish to send an ordinary text massage to us, you will prepare the message using either of the editors built into the system. Only after the massage is complete will you need to make the telephone connection.

Forthcoming Improvements

As mentioned earlier, our next significant improvement in the software will be a more flexible system allowing libraries of programs. One of the main reasons for doing this will be to allow the software to be configured to make efficient use of main memory in cases where the user does not need all of the built-in facilities. For example, we have no need for turtle graphics when compiling large system programs.

One of the long awaited features of the new library system will be an arrangement allowing mixture of PASCAL procedures with Assembly language routines and/or procedures compiled directly to the native code of the host machine. The necessary assemblers and code generation will come somewhat after the library system is operational. If all goes well, the library system should be ready to distribute during the winter quarter of 1978. The assemblers and native code versions of the compiler will come somewhat later as time for the necessary work permits.

Many people have asked whether we have in mind extensions to support Concurrent PASCAL, or similar facilities to allow independent processes running concurrently. This is something we would like to do eventually, but our current resources do not allow making definite plans in this area.

ADCIS MET IN DALLAS

News Release *Received: 78 Jan 9*

The 1978 Winter Meeting of the Association for the Development of Computer-based Instructional Systems (ADCIS) was held in Dallas, Texas, March 1-4, 1978. For further information, contact the conference host, Carol Luce, Educational Service Center, Region 100, 400, East Spring Valley Rd, Richardson, TX 75080, (214) 231-6301.

ACT

An 8080 Macroprocessor

BY ALEX CECIL
Lawrence Livermore Laboratory

ACT is a macro text interpreter which runs under the CP/M operating system on an Intel 8080 compatable microcomputer. More than 16K bytes of RAM are needed. ACT uses the same algorithm as TRAC [2] ("TRAC" is a registered trademark of Rockford Research Institute) and a set of primitive functions which are similar to those in the TRIX interpreter [1]. ACT also uses the special characters @ { | } in place of TRAC's #(,). ACT has been used to control jobs on a large time sharing system and to format diskette files; it is generally too slow for efficient human interaction.

Logically (but not necessarily psychologically), the syntax is very simple. Program and data are character strings which are handled by the same rules. There are three structures:

```
{. . .}
@{op|arg1|arg2|. . . |argn}
@@{op|arg1|arg2|. . . |argn}
```

The first is a literal quote; the contents are passed on as the value. The second performs the indicated operation on the argument strings and immediately rescans the resulting value. The last form performs the operation and passes the value to the next higher level function. Any op or argument may have any of the three forms. For more information on the scanning algorithm see any of the published descriptions of TRAC.

There is a version of ACT which uses upper case letters for the command names and the special characters ^ [,] ' instead of @{ | }. Also, the meta-character is the exclaim (!).

Execute line

```
act
act<file-name>
act<file-name><options>
```

If a file name is given, then the ACT macros in the file are read first; <file-name> has the general form "d:x.y". The string @ in is initialized to the characters between "act" and the return.

As a special case, <options> may be the three characters ^hh, where ^ is the caret (or up arrow), and hh is a hex page number. Free storage is allocated from the end of the interpreter up to, but not including, page hh or FBASE if ^hh is not given.

Non-disk I/O and debug

@{re|prompt} "read string"
Value is a line of input from the console (via CP/M call #10), or the next string from the input file. Line-feed, return, and tab characters from the input file are ignored; a string is terminated by the meta-character (~) or by the end-of-file. Comments on the input file are enclosed within matching left quotes (`). Input files generally do not end with the meta-character.

@{ps|a|a|. . .} "print string"
Each argument (after the "ps") is printed on a separate line; but there is no CR-LF after the last argument.

@{so|message} "send to Octopus"
The message is sent over the selected Octopus channel. Usually the message does not end with an Octopus end-message character. The echo is lost. (Note: "Octopus" is the name of the LLL time sharing system. The so and ro functions talk to Octopus over a serial interface as an ordinary terminal. Separate documentation illustrates how this hardware dependent function can be patched for different I/O ports.)

@{ro|x|s|f|eom} "read Octopus"
If the Octopus time-out is greater than zero, then a message terminator is sent. This terminator either causes a previous "so" message to be delivered or a log-out in case there was no previous "so" and the terminator is a control-D. Octopus output is then collected in the string @ro until the string X is encountered or until the time-out interval passes since the last character. The value is s if x was found, otherwise f. If s and f are omitted, the value is null. (Note this inconsistency with the functions gc and in.) A null x will always fail to match.

A non-null last argument (eom) will change the Octopus message terminator. The argument eom must be the two hex characters which correspond to the terminator. The terminator is initially a control-D.

@{io|x|x|. . .} "input-output parameters"
Each argument x is a letter (o, t, m) followed by an associated parameter:

oa	selects Octopus channel a. (initial value)
ob	selects Octopus channel b.
tn	sets the Octopus time-out to n seconds. E. g. t120.
mc	redefines the meta-character to be the character c.
p	print: terminal output is thru CP/M, "fw" applies.
d	display: direct output, "fw" does not apply. Note: This option is hardware dependent.

@{fw|n} "form width"
The "print string" function will start a new line after n characters. The initial form width is 80. However, this does not apply if an @{io|p} is in effect.

@{tn} "trace on"
Turn the trace mode on and save the previous mode. Note: An @{io|d} is implicitly given so that the output will wrap around; see the fw function above.

@{tf} "trace off"
Restore the previous trace mode, up to eight levels.

@{nb} "number of blocks"
Value is the number of sixteen word free storage blocks.

@{fs|n|m} "free storage"
Free storage is augmented by the memory locations n up to but not including m; n and m are decimal and must be multiples of 16.

@{vn} "version number"
Value is the ACT version number.

Disc I/O

@ {of | file-name | n | s | f} "open file"
The file name has the general form "d:x.y". If d is omitted, the system drive is selected; lower case characters in the name are mapped to upper case. The number of the next record to read or write is set to n. (Records are numbered sequentially starting at zero; each record consists of 128 characters.) If the open fails the value is f; otherwise the value is s.

Since only one file can be open at a time, the record number of the current file may need to be saved by @{rn}. Also, a currently opened output file must be closed before another open.

@ {mf | file-name | s | f} "make file"
If the named file can be created, the value is s; otherwise the value is f. The current record number is set to zero.

@ {cf | s | f} "close file"
The currently opened file is closed. The value is f if there is a close error, otherwise s.

@ {df} "destroy file"
The current file is destroyed.

@ {rn} "record number"
The value is the record number of the next record to read or write. This number applies to the currently opened file.

@ {wf | string-name | x | s | f} "write file"
Characters from the named string, starting with the first, are written to the currently opened file. If x is null, then the last partial record is not written; instead, the form pointer of the named string is left positioned at the remainder. For example, the output for the next write could be constructed by:
@{ns | name | @@ {in | name} <more output>}.

If x is non-null, then the first character of x is used to pad out the remainder of the record and the record is written. An entire record of pad characters is written if the previous output ended on a record boundary.

The value is f if there is a write error, otherwise s.

@ {rf | string-name | m | s | f} "read file"
The next m records are read from the currently opened file and appended to the named string. The value is f if the physical end of file is reached, otherwise s.

@ {rd} "read dialect"
The value is the text from the currently open file. The file is read starting with the record set by the open file. The input is processed in the same manner as described for the rs function. Thus, one program can load another in the same way as is done when a file name is put on the execute line after "act". Warning: No of or mf commands can be executed until the file has been read.

Control and delay

@ {eq | a | b | s | b | s | ... | b | s} or @ {eq | a | b | s | b | s | ... | f} "equal"
The value is the first s for which the preceding b is equal to a, where "equal" means string equality. If the last argument is not one of a "b-s" pair, then it provides an "else" value.

@ {ge | n | m | s | f} "greater than or equal to"
The value is s if n is numerically greater than or equal to m, otherwise the value is f.

@ {nt | name | s | f} "null test"
The value is s if the named string is null, otherwise the value is f.

@ {ex | x} "exit"
Clear the active and neutral strings; then set the active string to x.

@ {st} "stop"
Return to CP/M.

@ {zz | n} "sleep"
Sleep for n/10 seconds.

String ops and hex-to-characters

@ {ns | name | value | name | value | ...} "name string"
Create one or more strings with the given values.

@ {as | name | x | name | x | ...} "append string"
Append the string x to the named string.

@ {ss | name | arg | arg | ...} "segment strong"
Create arguments in the named string.

@ {cl | name | arg | arg | ...} or @ {name | arg | arg | ...} "call"
Return the named string, starting at the first character, with the actual arguments substituted for those corresponding substrings which were matched by a previous "segment string".

@ {gc | name | n | s | f} "get characters" (4 or more arguments)
Move the form pointer over n characters, or to the end of the form if there are fewer than n characters remaining. The characters passed over are put into the string "@gc". Return s if there are n characters after the form pointer in the named string, otherwise return f.

@ {gc | name | n} "get characters" (3 arguments)
Like gc with 4 or more arguments except that the value of @gc is also the value of this function.

@ {gc | name}
Return the total number of characters in the named form.

@ {in | name | p | s | f} "initial" (4 or more arguments)
Starting at the form pointer, search for the substring p. The characters which are passed over in the search, up to but not including p, are put into the string "@in". If p is found return s, otherwise return f. The form pointer is left positioned after s, or at the end of the form. Null p always fails.

@ {in | name | p} "initial" (3 arguments)
Like in with 4 or more arguments except that the value of @in is also the value of this function.

@ {in | name} "initial" (2 arguments)
Return the characters from the form pointer to the end of the form; also set @in to this value. The form pointer is left past the end of the form.

@ {cr | name} "call restore"
Move the form pointer of the named string to the beginning of the string.

@ {dd | name | name | ...} "delete definitions"
Delete the named strings.

@ {pk | hex} "pack"
Return the characters whose hex values are given.

@ {uk | x} "unpack"
Return the hex form of the string x.

Arithmetic

@ {ad | n | n | ...} "add"
Return the sum of the arguments. Numbers are represented by signed decimal integers whose magnitude is less than 32768.

@ {su | n | m} "subtract"
Return n-m.

@ {ml | n | n | ...} "multiply"
Return the product of the arguments.

@ {dv | n | m} "divide"
Return n/m.

@ {md | n | m} "mod"
Return n mod m.

Special Strings

@ cb "console break"
The value of this string is the last console break character.
@ im "idle macro"
This macro is loaded whenever the active string is empty. The initial value of @im is @{ps|@{rs|.}} , but @im may be changed at any time.

Console "break"

The last character of any console input, other than for the @ rs function, is saved in the string @cb. Moreover, if the input character is an ESC, then you will be prompted to type one of the letters t, n or i which will turn the trace on (t), turn the trace off (n), or reload the idle macro (i). Console input will stop output, or cause an immediate return from the zz or ro functions.

Generating a COM File with a Preloaded ACT Program

A COM file containing the ACT interpreter and a preloaded macro file may be created as follows, assuming a 32K system.

1. Append something like the following to your macro file
```
@{ns|@im|}
   @{fs|26368|29184}
   @{ns|@im|{@{ps|@{rs|.}}}}
   @{start}
}}
@ st
```
where "start" is the name of your top level macro. Note that the fs function adds 6700 hex up to 7200 hex to the free storage list after the program is started (step 4). Free storage normally extends to FBASE which is 7200 hex in a 32K system.
2. Load your program by typing
```
act macro-file ^67
```
where "^" is the caret, or up arrow. Free storage will extend up to 6700 hex; if your program needs more space, then you need more than 32K to make a COM file. The st function at the end of your file will return control to CP/M.
3. Make a COM file with the SAVE command
```
save 103 code.com
```
4. Run the code by typing either of the following
```
code
code options
```
The options will be in the string @in.

Error Messages

DISK EOF: The rs function is trying to read beyond the end of file.

FREE STORAGE IS EMPTY: The combined storage required for named strings, string names, and active and neutral strings has exceeded the allocated memory. At this point, the active and neutral strings are cleared and control given to the @im macro. The second time this happens, control returns to CP/M.

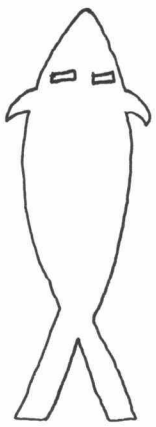
NS STACK OVERFLOW: The number of unprocessed arguments exceeds 250. The level of nesting or the number of arguments is excessive. For example, the number of arguments for a ns should not get much over 200.

References

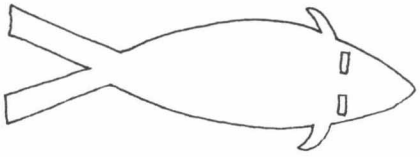
1. Moll, H., "TRIX – An interactive, interpretive language for manipulating strings of characters." *Lawrence Livermore Lab Report* UCID-30100.
2. Moors, C. N., "TRAC, a Procedure describing Language for the Reactive Typewriter." *Comm. ACM 9*, 3 (March, 1966)
3. Moors, C. N., "TRAC, A Text Handling Language." *Proc. ACM 20th Nat. Conf.*, Cleveland, August, 1965.
4. Wegner, Peter, *Programming Languages, Information Structures, and Machine Organization*. McGraw-Hill Book Co., 1968.

Note: Since the listing of ACT is quite long, we are publishing the PL/M listing only in this issue. In the next issue we will print the symbol table and assembler listing along with a table referencing selected line numbers from the PL/M version to the assembler listing – TRW.

```
8080 PLM1 VERS 3.0

00001  1    /* STANDARD LANGUAGE EXTENSIONS.  6/16/75  */
00002  1    DECLARE
00003  1       TRUE           LITERALLY 'OFFH',
00004  1       FALSE          LITERALLY '0',
00005  1       FOREVER        LITERALLY 'WHILE TRUE',
00006  1       FOR            LITERALLY 'DO',
00007  1       ENDFOR         LITERALLY 'END;',
00008  1       FI             LITERALLY 'END;',
00009  1       SELECT         LITERALLY 'DO CASE',
00010  1       OF             LITERALLY ';',
00011  1       ENDSELECT      LITERALLY 'END;',
00012  1       FUNCTION       LITERALLY 'PROCEDURE',
00013  1       ENDFUNCTION    LITERALLY 'END',
00014  1       ENDPROCEDURE   LITERALLY 'END',
00015  1       ENDDO          LITERALLY 'END;',
00016  1       BEGIN          LITERALLY 'DO;';
00017  1
00018  1    100H: START:
00019  1    /* ACT VERSION 31 10/17/77 */
00020  1    /* DON'T FORGET TO CHANGE PF$VN */
00021  1
00022  1    /* TO CHANGE TO UPPER CASE FOR TTY MOD33:
00023  1       1. CHANGE SPECIAL ACT CHARACTERS,
00024  1       2. CHANGE ALPHABET (LIT$A ETC.),
00025  1       3. CHANGE PF$LIST.
00026  1       4. CHANGE META$CHAR TO APOSTROPHE.
00027  1    */
00028  1
00029  1
```



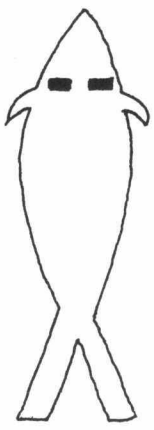
```
00030  1                /*** SPECIAL CHARACTERS ***/
00031  1
00032  1    DECLARE  /* CONTROL CHARACTERS */
00033  1       NUL LITERALLY '00H',
00034  1       SOH LITERALLY '01H',   /* CONTROL-A */
00035  1       ETX LITERALLY '03H',   /* CONTROL-C */
00036  1       EOT LITERALLY '04H',   /* CONTROL-D */
00037  1       ENQ LITERALLY '05H',   /* CONTROL-E */
00038  1       BEL LITERALLY '07H',
00039  1       HT  LITERALLY '09H',
00040  1       LF  LITERALLY '0AH',
00041  1       VT  LITERALLY '0BH',
00042  1       CR  LITERALLY '0DH',
00043  1       SO  LITERALLY '0EH',   /* CONTROL N */
00044  1       SI  LITERALLY '0FH',   /* CONTROL O */
00045  1       DC3 LITERALLY '13H',
00046  1       NAK LITERALLY '15H',   /* CONTROL-U */
00047  1       SYN LITERALLY '16H',
00048  1       ETB LITERALLY '17H',   /* CONTROL-W */
00049  1       CAN LITERALLY '18H',   /* CONTROL-X */
00050  1       EM  LITERALLY '19H',   /* CONTROL-Y */
00051  1       SUB LITERALLY '1AH',   /* CONTROL-Z */
00052  1       ESC LITERALLY '1BH',
00053  1       DEL LITERALLY '7FH',
00054  1       PHI LITERALLY '80H',   /* MARKS AN UNUSED CHAR. POSITION */
00055  1       NIL LITERALLY '81H';   /* MARKS THE END OF A STRING, SEE NILNIL IN FS$GET */
00056  1
00057  1    DECLARE  /* SPECIAL ACT CHARACTERS */
00058  1       FUN$CH       LITERALLY '40H',   /* 'AT' CHARACTER (OR UP-ARROW) */
00059  1       LEFT$DELIM   LITERALLY '7BH',   /* LEFT BRACE (OR SQUARE BRACKET) */
00060  1       ARG$DELIM    LITERALLY '7CH',   /* BAR (OR COMMA) */
00061  1       RIGHT$DELIM LITERALLY '7DH',   /* RIGHT BRACE (OR SQUARE BRACKET) */
00062  1       COMM$DELIM   LITERALLY '60H';   /* GRAVE ACCENT (OR APOSTROPHE) */
00063  1
00064  1    DECLARE  /* ALPHABET (EITHER LOWER OR UPPER CASE) */
00065  1       LIT$A LITERALLY '61H',
00066  1       LIT$B LITERALLY '62H',
00067  1       LIT$C LITERALLY '63H',
00068  1       LIT$D LITERALLY '64H',
00069  1       LIT$E LITERALLY '65H',
00070  1       LIT$F LITERALLY '66H',
00071  1       LIT$G LITERALLY '67H',
00072  1       LIT$H LITERALLY '68H',
00073  1       LIT$I LITERALLY '69H',
00074  1       LIT$J LITERALLY '6AH',
00075  1       LIT$K LITERALLY '6BH',
00076  1       LIT$L LITERALLY '6CH',
00077  1       LIT$M LITERALLY '6DH',
00078  1       LIT$N LITERALLY '6EH',
00079  1       LIT$O LITERALLY '6FH',
00080  1       LIT$P LITERALLY '70H',
00081  1       LIT$Q LITERALLY '71H',
00082  1       LIT$R LITERALLY '72H',
00083  1       LIT$S LITERALLY '73H',
00084  1       LIT$T LITERALLY '74H',
00085  1       LIT$U LITERALLY '75H',
00086  1       LIT$V LITERALLY '76H',
00087  1       LIT$W LITERALLY '77H',
00088  1       LIT$X LITERALLY '78H',
00089  1       LIT$Y LITERALLY '79H',
00090  1       LIT$Z LITERALLY '7AH';
00091  1
00092  1
00093  1                /*** DATA WHICH MUST BE LOCATED MOD 16 ***/
00094  1
00095  1    DECLARE  /* THIS MUST BE THE FIRST NON-LITERAL DECLARE */
00096  1       AT$GC DATA (0,0,FUN$CH,LIT$G,LIT$C,NIL,0,0,0,0),   /* 106-10F */
00097  1       AT$IN DATA (0,0,FUN$CH,LIT$I,LIT$N,NIL),   /* 110-115 */
00098  1       AT$RO DATA (0,0,FUN$CH,LIT$R,LIT$O,NIL,0,0,0,0),   /* 116-11F */
00099  1       AT$CB DATA (0,0,FUN$CH,LIT$C,LIT$B,NIL);   /* 120-125 */
00100  1
00101  1
00102  1                /*** BASIC I/O, SYSTEM CALLS ***/
00103  1
00104  1    DECLARE  /* I/O READY TESTS FOR CONSOLE */
00105  1       /* 'IN' REFERS TO TERMINAL-TO-OCTOPUS DIRECTION */
00106  1       /* 'OUT' REFERS TO OCTOPUS-TO-TERMINAL DIRECTION */
00107  1       CPM$PRINT$FLAG BYTE INITIAL (TRUE),
00108  1       TERM$STATUS     LITERALLY '1',   /* HARDWARE */
00109  1       TERM$DATA       LITERALLY '0',   /* HARDWARE */
00110  1       TERM$IN$READY   LITERALLY 'MON2(11,0)',
00111  1       TERM$OUT$READY LITERALLY 'ROR(MON2(11,0),2)';
00112  1
```



```
00113  1    DECLARE
00114  1       (CON$WIDTH, CON$COLUMN) BYTE INITIAL (80,0),
00115  1       CON$IN BYTE INITIAL (TRUE),   /* CONSOLE INPUT (DISK INPUT IF FALSE) */
00116  1       CONBUF(82) BYTE INITIAL (80,0,1),
00117  1       OCTO$TIMEOUT ADDRESS INITIAL (120),
00118  1       TFCB$PTR ADDRESS INITIAL (5CH),   /* TRANSIENT FILE CONTROL BLOCK */
00119  1       TFCB BASED TFCB$PTR BYTE,
00120  1       TBUFF$PTR ADDRESS INITIAL (80H),   /* TRANSIENT FILE BUFFER */
00121  1       TBUFF BASED TBUFF$PTR BYTE,
00122  1       TBUFF$INDEX BYTE INITIAL (128),
00123  1       END$OF$FILE LITERALLY 'SUB',
00124  1       OCTO$EOM BYTE INITIAL (EOT),
00125  1       META$CHAR BYTE INITIAL (7EH),   /* TILDA OR EXCLAIM */
00126  1       PRELOADED BYTE INITIAL (FALSE),
00127  1       (DEFAULT$DRIVE,PREVIOUS$DRIVE,CURRENT$DRIVE) BYTE;
00128  1
00129  1    DECLARE  /* SOME GENERAL PURPOSE TEMPS */
00130  1       TEMP$PTR ADDRESS,
00131  1       DUMMY BYTE,
00132  1       ITEMP BYTE;
00133  1
00134  1
00135  1                    /*** BASIC I/O ***/
00136  1
00137  1    DECLARE
00138  1       BDOS LITERALLY '5H',
00139  1       FBASEA ADDRESS INITIAL (6H),
00140  1       FBASE BASED FBASEA ADDRESS,
00141  1       EXIT LITERALLY 'CALL MON1(0,0)';
00142  1
00143  1    MON1: PROCEDURE(F,A);
00144  2       DECLARE F BYTE, A ADDRESS;
00145  2       GO TO BDOS;
00146  2       ENDPROCEDURE MON1;
00147  1
00148  1    MON2: FUNCTION(F,A) BYTE;
00149  2       DECLARE F BYTE, A ADDRESS;
00150  2       GO TO BDOS;
00151  2       ENDFUNCTION MON2;
00152  1
00153  1    CRLF: PROCEDURE;
00154  2       CALL MON1(2,CR);
00155  2       CALL MON1(2,LF);
00156  2       CON$COLUMN=0;
00157  2       ENDPROCEDURE CRLF;
00158  1
00159  1    PRINT$CH: PROCEDURE(CH);
00160  2       DECLARE CH BYTE;
00161  2       CALL MON1(2,CH);
00162  2       IF CH = CR THEN DO; CON$COLUMN=0; FI
00163  2       /* NOTE... SHOULD HANDLE NON-GRAPHIC CHARACTERS */
00164  2       IF (CON$COLUMN:=CON$COLUMN+1) = CON$WIDTH THEN DO;
00165  2          CALL CRLF;
00166  3       FI
00167  2       ENDPROCEDURE PRINT$CH;
00168  1
00169  1    DISPLAY$CH: PROCEDURE(CH);
00170  2       /* SEND A CHAR TO TERMINAL. NOTE: HARDWARE DEPENDENT */
00171  2       DECLARE CH BYTE;
00172  2       DO WHILE ROR(INPUT(TERM$STATUS),2); ENDDO
00173  2       OUTPUT(TERM$DATA)=NOT CH;
00174  2       END DISPLAY$CH;
00175  1
00176  1    READ$CH: FUNCTION BYTE;
00177  2       RETURN MON2(1,0);
00178  2       ENDFUNCTION READ$CH;
00179  1
00180  1    CON$BREAK: FUNCTION BYTE;
00181  2       RETURN MON2(11,0);
00182  2       ENDFUNCTION CON$BREAK;
00183  1
00184  1    PRINT$MESS: PROCEDURE(MESS$PTR);
00185  2       DECLARE MESS$PTR ADDRESS;
00186  2       CALL MON1(9,MESS$PTR);
00187  2       CALL CRLF;
00188  2       ENDPROCEDURE PRINT$MESS;
00189  1
00190  1    SELECT$DRIVE: PROCEDURE;
00191  2       IF TFCB(0) = 0 THEN DO;
00192  2          CURRENT$DRIVE=DEFAULT$DRIVE;
00193  3       END; ELSE DO;
00194  2          CURRENT$DRIVE=(TFCB(0) AND 7)-1;
00195  3          TFCB(0)=0;
00196  3       FI
00197  2       IF PREVIOUS$DRIVE <> CURRENT$DRIVE THEN DO;
00198  2          PREVIOUS$DRIVE=CURRENT$DRIVE;
00199  3          CALL MON1(14,CURRENT$DRIVE);
00200  3       FI
00201  2       ENDPROCEDURE SELECT$DRIVE;
00202  1
```

```
00203  1    UPPER$CASE: FUNCTION(CH) BYTE;
00204  2       /* MAP A CHARACTER TO UPPER CASE */
00205  2       DECLARE CH BYTE;
00206  2       IF CH < 61H OR CH > 7AH THEN DO; RETURN CH; FI
00207  2       RETURN CH-20H;
00208  2       ENDFUNCTION UPPER$CASE;
00209  1
00210  1    WAIT$MS: PROCEDURE(N);
00211  2       /* N MILLI-SECOND DELAY (APPROXIMATE) */
00212  2       DECLARE (I,N) ADDRESS;
00213  2       I=0;
00214  2       DO WHILE I < N; CALL TIME(7); I=I+1; ENDDO
00215  2       ENDPROCEDURE WAIT$MS;
00216  1
00217  1    WAIT$TENTH: PROCEDURE(N);
00218  2       /* N SECOND DELAY */
00219  2       DECLARE (I,N) ADDRESS;
00220  2       FOR I=1 TO N;
00221  2          IF CON$BREAK THEN DO; RETURN; FI
00222  3          CALL TIME(255);
00223  3          CALL TIME(255);
00224  3          CALL TIME(135);
00225  3       ENDDO
00226  2       ENDPROCEDURE WAIT$TENTH;
00227  1
00228  1
00229  1                    /*** OCTOPUS I/O PRIMITIVES ***/
00230  1
00231  1    DECLARE
00232  1       /* 'IN' REFERS TO TERMINAL-TO-OCTOPUS DIRECTION */
00233  1       /* 'OUT' REFERS TO OCTOPUS-TO-TERMINAL DIRECTION */
00234  1       OCTO$CHANNEL$A BYTE INITIAL (TRUE),
00235  1       OCTO$A$STATUS LITERALLY '5',
00236  1       OCTO$A$DATA   LITERALLY '4',
00237  1       OCTO$B$STATUS LITERALLY '7',
00238  1       OCTO$B$DATA   LITERALLY '6',
00239  1       OCTO$OUT$READY LITERALLY 'OCTO$STATUS',
00240  1       OCTO$IN$READY  LITERALLY 'ROR(OCTO$STATUS,2)';
00241  1
00242  1    OCTO$STATUS: FUNCTION BYTE;
00243  2       /* RETURN THE TRUE (NOT COMPLEMENTED) OCTOPUS STATUS FOR THE SELECTED PORT */
00244  2       IF OCTO$CHANNEL$A THEN RETURN NOT INPUT(OCTO$A$STATUS);
00245  2       RETURN NOT INPUT(OCTO$B$STATUS);
00246  2       ENDFUNCTION OCTO$STATUS;
00247  1
00248  1    SEND$CH: PROCEDURE(CH);
00249  2       /* SEND A CHARACTER TO THE SELECTED OCTOPUS PORT */
00250  2       DECLARE CH BYTE;
00251  2
00252  2       DO WHILE NOT OCTO$IN$READY; ENDDO
00253  2       IF OCTO$CHANNEL$A THEN DO;
00254  2          OUTPUT(OCTO$A$DATA)=NOT CH;
00255  3       END; ELSE DO;
00256  2          OUTPUT(OCTO$B$DATA)=NOT CH;
00257  3       FI
00258  2       ENDPROCEDURE SEND$CH;
00259  1
00260  1    RECEIVE$CH: FUNCTION BYTE;
00261  2       /* GET A CHARACTER FORM THE SELECTED OCTOPUS CHANNEL */
00262  2       DECLARE I ADDRESS;
00263  2       FOR I=0 TO 65000;
00264  2          IF OCTO$OUT$READY THEN DO;
00265  3             IF OCTO$CHANNEL$A THEN RETURN (NOT INPUT(OCTO$A$DATA)) AND 7FH;
00266  4             RETURN (NOT INPUT(OCTO$B$DATA)) AND 7FH;
00267  4          FI
00268  3       ENDFOR
00269  2       RETURN OFFH;
00270  2       ENDFUNCTION RECEIVE$CH;
00271  1
00272  1
00273  1                 /*** BLOCK AND FREE STORAGE GLOBALS ***/
00274  1
00275  1    DECLARE
00276  1       FS$FIRST$PTR ADDRESS,  /* FREE STORAGE HEAD */
00277  1       BLOCK$LENGTH LITERALLY '16',
00278  1       BLOCK$LAST LITERALLY '15',
00279  1       PTR$MASK     LITERALLY 'OFFFOH',
00280  1       NOT$PTR$MASK LITERALLY 'OFH',
00281  1       (FIRST$GET A$FLAG,FIRST$GETB$FLAG,FIRST$PUTA$FLAG) BYTE,
00282  1       (GET$CHA$CH$PTR,PUT$CHA$CH$PTR,PUT$CHA$CURRENT$BLOCK$PTR) ADDRESS;
00283  1
00284  1
```

```
00285  1                /*** GLOBALS FOR SCAN ALGORITHM ***/
00286  1
00287  1    DECLARE
00288  1       NS$STACK(255) ADDRESS INITIAL (11B,1B),  /* NEUTRAL STACK, 8-BIT INDEX */
00289  1       NS$STACK$INDEX BYTE,  /* INDEX OF CURRENTLY 'OPEN' ARGUMENT */
00290  1       NS$SAVE$INDEX BYTE,  /* INDEX OF FIRST ARGUMENT OF CURRENT FUNCTION */
00291  1       NS$CH$PTR ADDRESS,  /* NEXT NEUTRAL CHARACTER ADDRESS */
00292  1       NS$CH BASED NS$CH$PTR BYTE,
00293  1       NS$ARG$BREAK BYTE,
00294  1       NS$ARG$FLAG       LITERALLY '0000B',  /* LAST FOUR BITS OF NS STACK ENTRY */
00295  1       NS$BEGIN$END$BIT LITERALLY '0001B',  /* MARKS LIMITS OF NEUTRAL STACK */
00296  1       NS$FUN$BIT        LITERALLY '0010B',  /* MARKS FIRST ARG OF FUNCTION */
00297  1       NS$ACTIVE$BIT     LITERALLY '0100B',  /* MARKS ACTIVE FUNCTION */
00298  1       NS$NULL$BEGIN$END LITERALLY '1001B',
00299  1       NS$FUN$MARKER     LITERALLY 'ROR(LOW(NS$STACK(NS$STACK$INDEX)),1)',
00300  1                                   /* NS BEGIN-FUNCTION TEST */
00301  1       NS$LIMIT$MARKER   LITERALLY 'LOW(NS$STACK(NS$STACK$INDEX))',
00302  1                                    /* NS BEGIN-END TEST */
00303  1       NS$ACTIVE$FLAG BYTE,
00304  1
00305  1       AS$BLOCK$PTR ADDRESS INITIAL(0),  /* CURRENT BLOCK IN ACTIVE STRING */
00306  1       AS$BLOCK BASED AS$BLOCK$PTR ADDRESS,
00307  1       AS$CH$PTR ADDRESS,  /* ADDRESS OF NEXT ACTIVE STRING CHARACTER */
00308  1       AS$CH BASED AS$CH$PTR BYTE,
00309  1
00310  1       (SS$CHA$PTR,SS$CHB$PTR,SS$CHC$PTR) ADDRESS, /* SUBSTRING SEARCH PTRS */
00311  1       SS$MASK LITERALLY '11000000B',
00312  1       SS$CHA BASED SS$CHA$PTR BYTE,
00313  1
00314  1       MOVE$PTR      LITERALLY 'TRUE',  /* FLAGS FOR AS$NEXT$CH CALL */
00315  1       DONT$MOVE$PTR LITERALLY 'FALSE',
00316  1       TRACE$FLAG BYTE,  /* 8 BIT 'STACK'. LOW ORDER BIT IS CURRENT */
00317  1
00318  1       (NAME$PTR,ARG$PTR) ADDRESS,  /* GENERAL USE IN PF'S AND TRACE */
00319  1
00320  1       NEXT$ARG LITERALLY 'WHILE NOT LOW(ARG$PTR:=NEXT$NS)',  /* IN PF'S */
00321  1       VALUE$PTR ADDRESS,  /* USED BY PF'S TO RETURN THEIR VALUE */
00322  1       VALUE$CH BASED VALUE$PTR BYTE,
00323  1       (S$PTR,F$PTR) ADDRESS,  /* USED BY PF'S FOR SUCCESS AND FAIL VALUES */
00324  1       NULL$STRING$PTR ADDRESS,  /* ORED WITH NS$BEGIN$END$BIT IN NS STACK  */
00325  1       DEAD$START BYTE INITIAL (TRUE),
00326  1       NO$EX$VALUE BYTE INITIAL (TRUE);
00327  1
00328  1
00329  1                /*** FREE STORAGE AND OTHER BLOCK STRUCTURE UTILITIES ***/
00330  1
00331  1    FS$FREE: PROCEDURE(FREE$THIS$PTR);
00332  2       /* ADD THE INDICATED BLOCK TO THE FREE STORAGE LIST */
00333  2       DECLARE
00334  2          FREE$THIS$PTR ADDRESS,
00335  2          FREE$THIS BASED FREE$THIS$PTR ADDRESS;
00336  2
00337  2       FREE$THIS=FS$FIRST$PTR;
00338  2       FS$FIRST$PTR=FREE$THIS$PTR;
00339  2       ENDPROCEDURE FS$FREE;
00340  1
00341  1    FS$GET: FUNCTION ADDRESS;
00342  2       /* ALLOCATE A BLOCK FROM FREE STORAGE, RETURN ITS ADDRESS */
00343  2       /* SET 'NEXT' FIELD TO END-CHAIN (ZERO HIGH BYTE) */
00344  2       /* AND FILL WITH NILS */
00345  2       DECLARE
00346  2          FS$FIRST BASED FS$FIRST$PTR ADDRESS,
00347  2          GOT$PTR ADDRESS,
00348  2          GOT BASED GOT$PTR ADDRESS,
00349  2          GOTC BASED GOT$PTR BYTE,
00350  2          FIRST$EMPTY BYTE INITIAL (TRUE),
00351  2          NILNIL LITERALLY '8181H';
00352  2
00353  2       IF HIGH(FS$FIRST$PTR) = 0 THEN DO;
00354  2          CALL PRINT$MESS(.'FREE STORAGE IS EMPTY  $');
00355  3          IF FIRST$EMPTY THEN DO;
00356  3             FIRST$EMPTY=FALSE;
00357  4             GO TO START;
00358  4          END; ELSE DO;
00359  3             EXIT;
00360  4          FI
00361  3       END; ELSE DO;
00362  2          GOT$PTR=FS$FIRST$PTR;
00363  3          FS$FIRST$PTR=FS$FIRST;
00364  3          GOT=0;
00365  3          GOT(1)=NILNIL;
00366  3          GOT(2)=NILNIL;
00367  3          GOT(3)=NILNIL;
00368  3          GOT(4)=NILNIL;
00369  3          GOT(5)=NILNIL;
00370  3          GOT(6)=NILNIL;
00371  3          GOT(7)=NILNIL;
00372  3          RETURN GOT$PTR;
00373  3       FI
00374  2       ENDFUNCTION FS$GET;
00375  1
```

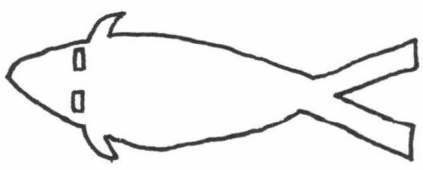


```
00376  1    FS$INIT: PROCEDURE(MIN$ADDRESS,MAX$ADDRESS);
00377  2       /* CREATE FREE STORAGE LIST */
00378  2       /* MAX$ADDRESS MUST BE A MULTIPLE OF BLOCK$LENGTH */
00379  2       DECLARE (MIN$ADDRESS,MAX$ADDRESS,BLK$PTR) ADDRESS;
00380  2
00381  2       FOR BLK$PTR=MIN$ADDRESS TO MAX$ADDRESS-BLOCK$LENGTH BY BLOCK$LENGTH;
00382  2          CALL FS$FREE(BLK$PTR);
00383  3       ENDFOR
00384  2       ENDPROCEDURE FS$INIT;
00385  1
00386  1    FREE$CHAIN: PROCEDURE(BLOCK$PTR);
00387  2       /* FREE ALL BLOCKS IN A CHAIN STARTING WITH BLOCK$PTR */
00388  2       DECLARE (BLOCK$PTR,BLOCK BASED BLOCK$PTR,NEXT$PTR) ADDRESS;
00389  2
00390  2       IF (BLOCK$PTR:=BLOCK$PTR AND PTR$MASK) = NULL$STRING$PTR THEN DO; RETURN; FI
00391  2       DO FOREVER;
00392  2          IF HIGH(BLOCK$PTR) = 0 THEN DO; RETURN; FI
00393  3          NEXT$PTR=BLOCK;
00394  3          CALL FS$FREE(BLOCK$PTR);
00395  3          BLOCK$PTR=NEXT$PTR;
00396  3       ENDDO;
00397  2       ENDPROCEDURE FREE$CHAIN;
00398  1
00399  1    APPEND$CH: FUNCTION(CH,BLOCK$PTR) ADDRESS;
00400  2       /* STARTING WITH BLOCK$PTR (<>0), SEARCH FOR THE LAST BLOCK */
00401  2       /* APPEND CH AND RETURN THE NEXT CHARACTER ADDRESS */
00402  2       DECLARE
00403  2          CH BYTE,
00404  2          (BLOCK$PTR,BLOCK BASED BLOCK$PTR,CH$PTR) ADDRESS,
00405  2          BLOCKC BASED CH$PTR BYTE;
00406  2
00407  2       IF HIGH(BLOCK$PTR) = 0 THEN DO; RETURN 0; FI
00408  2
00409  2       DO WHILE HIGH(BLOCK) > 0;
00410  2          BLOCK$PTR=BLOCK;
00411  3       ENDDO
00412  2
00413  2       CH$PTR=BLOCK$PTR+2;
00414  2       DO WHILE BLOCKC <> NIL;
00415  2          CH$PTR=CH$PTR+1;
00416  3       ENDDO
00417  2
00418  2       BLOCKC=CH;
00419  2       IF (LOW(CH$PTR:=CH$PTR+1) AND NOT$PTR$MASK) = 0 THEN DO;
00420  2          BLOCK,CH$PTR=FS$GET;
00421  3          CH$PTR=CH$PTR+2;
00422  3       FI
00423  2       RETURN CH$PTR;
00424  2       ENDFUNCTION APPEND$CH;
00425  1
00426  1    GET$CHA: FUNCTION(INITIAL$BLOCK$PTR) BYTE;
00427  2       /* RETURNS THE NEXT 8-BIT CHARACTER FROM CHAIN 'A' */
00428  2       /* TO INITIALIZE FOR A NEW CHAIN, SET FIRST$GETA$FLAG TO TRUE */
00429  2       /* CALLER MUST TEST FOR STRING TERMINAL */
00430  2       /* GET$CHA$PTR POINTS TO THE LAST RETURNED CHARACTER */
00431  2       DECLARE
00432  2          (INITIAL$BLOCK$PTR,CURRENT$BLOCK$PTR) ADDRESS,
00433  2          CURRENT$BLOCK BASED CURRENT$BLOCK$PTR ADDRESS,
00434  2          (CH BASED GET$CHA$CH$PTR,NEW$CHAIN) BYTE;
00435  2
00436  2       IF FIRST$GETA$FLAG THEN DO;
00437  2          FIRST$GETA$FLAG=FALSE;
00438  3          CURRENT$BLOCK$PTR=INITIAL$BLOCK$PTR;
00439  3          GET$CHA$CH$PTR=CURRENT$BLOCK$PTR+1;
00440  3       FI
00441  2       IF (LOW(GET$CHA$CH$PTR:=GET$CHA$CH$PTR+1) AND NOT$PTR$MASK) = 0 THEN DO;
00442  2          CURRENT$BLOCK$PTR=GET$CHA$CH$PTR-BLOCK$LENGTH;
00443  3          CURRENT$BLOCK$PTR=CURRENT$BLOCK;
00444  3          GET$CHA$CH$PTR=CURRENT$BLOCK$PTR+2;
00445  3       FI
00446  2       RETURN CH;
00447  2       ENDFUNCTION GET$CHA;
00448  1
00449  1    GET$CHB: FUNCTION(INITIAL$BLOCK$PTR) BYTE;
00450  2       /* RETURNS THE NEXT 8-BIT CHARACTER FROM CHAIN 'B' */
00451  2       /* TO INITIALIZE FOR A NEW CHAIN, SET FIRST$GETB$FLAG TO TRUE */
00452  2       /* CALLER MUST TEST FOR STRING TERMINAL */
00453  2       DECLARE
00454  2          (INITIAL$BLOCK$PTR,CURRENT$BLOCK$PTR,CH$PTR) ADDRESS,
00455  2          CURRENT$BLOCK BASED CURRENT$BLOCK$PTR ADDRESS,
00456  2          (CH BASED CH$PTR,NEW$CHAIN) BYTE;
00457  2
00458  2       IF FIRST$GETB$FLAG THEN DO;
00459  2          FIRST$GETB$FLAG=FALSE;
00460  3          CURRENT$BLOCK$PTR=INITIAL$BLOCK$PTR;
00461  3          CH$PTR=CURRENT$BLOCK$PTR+1;
00462  3       FI
00463  2       IF (LOW(CH$PTR:=CH$PTR+1) AND NOT$PTR$MASK) = 0 THEN DO;
00464  2          CURRENT$BLOCK$PTR=CURRENT$BLOCK;
00465  3          CH$PTR=CURRENT$BLOCK$PTR+2;
00466  3       FI
00467  2       RETURN CH;
00468  2       ENDFUNCTION GET$CHB;
00469  1
```

```
00470  1   PUT$CHA: PROCEDURE(INITIAL$BLOCK$PTR,CHA);
00471  2      /* PUT CHA INTO THE CHAIN 'A' */
00472  2      /* TO INITIALIZE FOR A NEW CHAIN, SET FIRST$PUTA$FLAG TO TRUE */
00473  2      /*    AND SET INITIAL$BLOCK$PTR TO FS$GET */
00474  2      /* PUT$CHA$CH$PTR POINTES TO THE NEXT CHAR TO BE STORED */
00475  2
00476  2      DECLARE
00477  2         INITIAL$BLOCK$PTR ADDRESS,
00478  2         CURRENT$BLOCK BASED PUT$CHA$CURRENT$BLOCK$PTR ADDRESS,
00479  2         (CH BASED PUT$CHA$CH$PTR,NEW$CHAIN,CHA) BYTE;
00480  2
00481  2      IF FIRST$PUTA$FLAG THEN DO;
00482  2         FIRST$PUTA$FLAG=FALSE;
00483  3         PUT$CHA$CURRENT$BLOCK$PTR=INITIAL$BLOCK$PTR;
00484  3         PUT$CHA$CH$PTR=PUT$CHA$CURRENT$BLOCK$PTR+2;
00485  3      FI
00486  2      CH=CHA;
00487  2      IF (LOW(PUT$CHA$CH$PTR:=PUT$CHA$CH$PTR+1) AND NOT$PTR$MASK) = 0 THEN DO;
00488  2         CURRENT$BLOCK,PUT$CHA$CURRENT$BLOCK$PTR=FS$GET;
00489  3         PUT$CHA$CH$PTR=PUT$CHA$CURRENT$BLOCK$PTR+2;
00490  3      FI
00491  2      ENDPROCEDURE PUT$CHA;
00492  1
00493  1   COPY$VALUE: PROCEDURE(SOURCE$PTR,SINK$PTR);
00494  2      /* SOURCE$PTR IS THE ADDRESS OF THE SOURCE CHAIN */
00495  2      /* SINK$PTR IS THE ADDRESS OF AN EMPTY BLOCK */
00496  2      /* ONLY 7-BIT CHARACTERS ARE COPIED */
00497  2      /* NOTE: PF$IN USES THE FINAL VALUE OF GET$CHA$CH$PTR */
00498  2      DECLARE
00499  2         (SOURCE$PTR,SINK$PTR) ADDRESS,
00500  2         CH BYTE;
00501  2
00502  2      FIRST$GETA$FLAG,FIRST$PUTA$FLAG=TRUE;
00503  2      DO WHILE (CH:=GET$CHA(SOURCE$PTR)) <> NIL;
00504  2         IF NOT ROL(CH,1) THEN DO;
00505  3            CALL PUT$CHA(SINK$PTR,CH);
00506  4         FI
00507  3      ENDDO
00508  2      ENDPROCEDURE COPY$VALUE;
00509  1
00510  1   STRING$EQUALITY: FUNCTION(PTA,PTB) BYTE;
00511  2      /* TRUE IF STRINGS CONTAIN THE SAME SEQUENCE OF 7-BIT CHARACTERS */
00512  2      DECLARE
00513  2         (PTA,PTB) ADDRESS,
00514  2         (CHA,CHB) BYTE;
00515  2
00516  2      NEXTA: FUNCTION BYTE;
00517  3         DO WHILE ((CHA:=GET$CHA(PTA)) <> NIL) AND ROL(CHA,1);
00518  3         ENDDO
00519  3         RETURN CHA;
00520  3         ENDFUNCTION NEXTA;
00521  2
00522  2      NEXTB: FUNCTION BYTE;
00523  3         DO WHILE ((CHB:=GET$CHB(PTB)) <> NIL) AND ROL(CHB,1);
00524  3         ENDDO
00525  3         RETURN CHB;
00526  3         ENDFUNCTION NEXTB;
00527  2
00528  2      FIRST$GETA$FLAG,FIRST$GETB$FLAG=TRUE;
00529  2      DO WHILE NEXTA = NEXTB;
00530  2         IF CHA = NIL THEN DO; RETURN TRUE; FI
00531  3      ENDDO
00532  2      RETURN FALSE;
00533  2      ENDFUNCTION STRING$EQUALITY;
00534  1
00535  1   NULL$TEST: FUNCTION(BLOCK$PTR) BYTE;
00536  2      /* 'TRUE' IF THERE ARE NO 7-BIT CHARACTERS BEFORE THE NIL */
00537  2      DECLARE
00538  2         BLOCK$PTR ADDRESS,
00539  2         CH BYTE;
00540  2
00541  2      IF HIGH(BLOCK$PTR) = 0 THEN DO; RETURN TRUE; FI
00542  2      FIRST$GETA$FLAG=TRUE;
00543  2      DO WHILE (CH:=GET$CHA(BLOCK$PTR AND PTR$MASK)) <> NIL;
00544  2         IF NOT ROL(CH,1) THEN DO; RETURN FALSE; FI
00545  3      ENDDO
00546  2      RETURN TRUE;
00547  2      ENDFUNCTION NULL$TEST;
00548  1
00549  1   PRINT$STRING: PROCEDURE(BLOCK$PTR);
00550  2      /* PRINT ALL 7-BIT CHARACTERS IN THE STRING */
00551  2      DECLARE
00552  2         BLOCK$PTR ADDRESS,
00553  2         CH BYTE;
00554  2
00555  2      FIRST$GETA$FLAG=TRUE;
00556  2      DO WHILE (CH:=GET$CHA(BLOCK$PTR AND PTR$MASK))<>NIL AND NOT TERM$IN$READY;
00557  2         IF NOT ROL(CH,1) THEN DO;
00558  3            IF CPM$PRINT$FLAG THEN DO;
00559  4               CALL PRINT$CH(CH);
00560  5            END; ELSE DO;
00561  4               CALL DISPLAY$CH(CH);
00562  5            FI
00563  4         FI
00564  3      ENDDO
00565  2      ENDPROCEDURE PRINT$STRING;
```

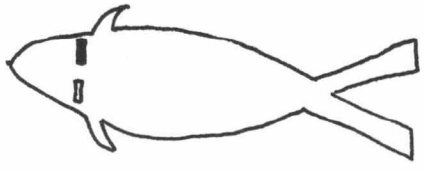
```
00566  1
00567  1
00568  1                    /*** OCTOPUS I/O UTILITIES ***/
00569  1
00570  1    CLEAR$OCTO$OUTPUT: PROCEDURE;
00571  2       CALL WAIT$TENTH(2);
00572  2       DO WHILE OCTO$OUT$READY; DUMMY=RECEIVE$CH; CALL WAIT$TENTH(2); ENDDO
00573  2       ENDPROCEDURE CLEAR$OCTO$OUTPUT;
00574  1
00575  1    SEND$STRING: PROCEDURE(BLOCK$PTR);
00576  2       /* SEND A MESSAGE TO OCTOPUS. (USUALLY WITHOUT END-OF-LINE CHARACTER) */
00577  2       /* IGNORE THE ECHO */
00578  2       DECLARE BLOCK$PTR ADDRESS, CH BYTE;
00579  2       CALL CLEAR$OCTO$OUTPUT;
00580  2       FIRST$GETA$FLAG=TRUE;
00581  2       DO WHILE (CH:=GET$CHA(BLOCK$PTR)) <> NIL;
00582  2          IF NOT ROL(CH,1) THEN DO; CALL SEND$CH(CH); FI
00583  3       ENDDO
00584  2       ENDPROCEDURE SEND$STRING;
00585  1
00586  1    RECEIVE$STRING: FUNCTION(END$MESSAGE$PTR) BYTE;
00587  2       /* RECEIVE AN OCTOPUS MESSAGE. VALUE$PTR WILL ADDRESS IT */
00588  2       /* VALUE IS FALSE IF TIME-OUT BEFORE END-MESSAGE IS RECOGNIZED */
00589  2       /* DOES NOT SEND EOT IF ZERO DELAY */
00590  2       /* NOTE: GET A LOG-OFF IF NULL MESSAGE AND DELAY > 0 */
00591  2       DECLARE
00592  2          (END$MESSAGE$PTR,RESTART$PTR,COUNT,COUNT$MAX,I) ADDRESS,
00593  2          CHA BYTE,
00594  2          CYCLES$PER$SECOND LITERALLY '5000'; /* DEPENDENT ON HARDWARE TIMING */
00595  2
00596  2       MATCH$END: FUNCTION BYTE;
00597  3          FIRST$GETB$FLAG=TRUE;
00598  3          GET$CHA$CH$PTR=RESTART$PTR;
00599  3          DO WHILE (CHA:=GET$CHA(0)) = GET$CHB(END$MESSAGE$PTR);
00600  3             IF CHA = NIL THEN DO; RETURN TRUE; FI
00601  4             IF OCTO$OUT$READY THEN DO; /* DON'T LOSE HIGH BAUD INPUT */
00602  4                COUNT=COUNT$MAX;
00603  5                CALL PUT$CHA(0,RECEIVE$CH);
00604  5             FI
00605  4          ENDDO
00606  3          IF CHA <> NIL THEN DO;
00607  3             GET$CHA$CH$PTR=RESTART$PTR;
00608  4             DUMMY=GET$CHA(0);
00609  4             RESTART$PTR=GET$CHA$CH$PTR;
00610  4          FI
00611  3          RETURN FALSE;
00612  3          ENDFUNCTION MATCH$END;
00613  2
00614  2       IF OCTO$TIMEOUT = 0 THEN DO;
00615  2          COUNT$MAX=2;
00616  3       END; ELSE DO;
00617  2          COUNT$MAX=OCTO$TIMEOUT+1;
00618  3          CALL CLEAR$OCTO$OUTPUT;
00619  3          CALL SEND$CH(OCTO$EOM);
00620  3       FI
00621  2
00622  2       RESTART$PTR=(VALUE$PTR:=FS$GET)+1;
00623  2       FIRST$PUTA$FLAG=TRUE;
00624  2       COUNT=COUNT$MAX;
00625  2       DO WHILE (COUNT:=COUNT-1) <> 0 AND NOT CON$BREAK;
00626  2          FOR I=0 TO CYCLES$PER$SECOND;
00627  3             IF OCTO$OUT$READY THEN DO;
00628  4                COUNT=COUNT$MAX;
00629  5                CALL PUT$CHA(VALUE$PTR,RECEIVE$CH);
00630  5                IF MATCH$END THEN DO; RETURN TRUE; FI
00631  5             FI
00632  4          ENDFOR
00633  3       ENDDO
00634  2       RETURN FALSE;
00635  2       ENDFUNCTION RECEIVE$STRING;
00636  1
00637  1
00638  1                 /*** SYMBOL TABLE ROUTINES ***/
00639  1
00640  1    /* A HASH TABLE, WHICH IS AN ARRAY OF 32 CHAIN POINTERS, IS USED.              */
00641  1    /* EACH ENTRY IS A BLOCK WITH THE FOLLOWING FIELDS: (NO. BYTES IN FIELD)       */
00642  1    /*    NEXT(2), CH-PTR(2), VALUE(2), NAME(2), NAME-TEXT(8)          */
00643  1    /* ST$FIND DOES A LOOKUP AND SETS ST$ENTRY$PTR AND HASH$INDEX .               */
00644  1    /* ST$DELETE DELETES THE PREVIOUSLY FOUND ENTRY AT ST$ENTRY$PTR.              */
00645  1    /* ST$CREATE CREATES AN ENTRY AT HASH$INDEX.                                  */
00646  1    /* ST$ADD$VALUE FILLS IN THE VALUE PTR FOR THE ENTRY AT ST$ENTRY$PTR          */
00647  1
00648  1    DECLARE  /* SYMBOL TABLE STUFF */
00649  1       HASH$TABLE(32) ADDRESS,  /*ADDRESS OF FIRST IN CHAIN OF EQUAL HASH ENTRIES*/
00650  1       HASH$INDEX BYTE,         /* INDEX INTO HASH$TABLE, SET BY ST$FIND */
00651  1       ST$ENTRY$PTR ADDRESS,  /* ADDRESS OF AN ENTRY (BLOCK) */
00652  1       ST$ENTRY BASED ST$ENTRY$PTR ADDRESS,
00653  1       ST$PREVIOUS$PTR ADDRESS, /* ADDRESS OF CHAIN POINTER TO CURRENT ST$ENTRY */
00654  1       ST$PREVIOUS BASED ST$PREVIOUS$PTR ADDRESS, /* HASH INDEX OF PREVIOUS BLOCK */
00655  1       ST$CH$PTR$INDEX  LITERALLY '1',  /* ST$ENTRY(1) IS LAST CH PTR (FORM PTR)*/
00656  1       ST$VALUE$INDEX   LITERALLY '2',  /* ST$ENTRY(2) IS THE VALUE POINTER */
00657  1       ST$NAME$INDEX    LITERALLY '3',  /* ST$ENTRY(3) IS PTR TO REST OF NAME */
00658  1       ST$NAME$OFFSET   LITERALLY '6';  /* ST$ENTRY$PTR+6 IS START OF NAME BLOCK */
```

```
00659  1
00660  1    ST$INIT: PROCEDURE;
00661  2       FOR HASH$INDEX=0 TO LAST(HASH$TABLE); HASH$TABLE(HASH$INDEX)=0; ENDFOR
00662  2       ENDPROCEDURE ST$INIT;
00663  1
00664  1    ST$FIND: FUNCTION(NAME$PTR) BYTE;
00665  2       /* IF FOUND, VALUE IS TRUE AND ST$ENTRY$PTR, ST$PREVIOUS$PTR ARE SET */
00666  2       /* IN ANY CASE, HASH$INDEX IS SET */
00667  2       DECLARE
00668  2          NAME$PTR ADDRESS,
00669  2          (HASH,CH) BYTE,
00670  2          HASH$MASK LITERALLY '11111B'; /* FOR 32 HASH CHAINS */
00671  2
00672  2       HASH=0;  FIRST$GETA$FLAG=TRUE;
00673  2       DO WHILE (CH:=GET$CHA(NAME$PTR)) <> NIL;
00674  2          HASH=HASH+CH;
00675  3       ENDDO;
00676  2
00677  2       IF HIGH(ST$ENTRY$PTR:=HASH$TABLE(HASH$INDEX:=HASH AND HASH$MASK))=0 THEN DO;
00678  2          RETURN FALSE;
00679  3       FI
00680  2       ST$PREVIOUS$PTR=.HASH$TABLE(HASH$INDEX);
00681  2       DO WHILE NOT STRING$EQUALITY(NAME$PTR,ST$ENTRY$PTR+ST$NAME$OFFSET);
00682  2          ST$PREVIOUS$PTR=ST$ENTRY$PTR;
00683  3          IF HIGH(ST$ENTRY$PTR:=ST$ENTRY) = 0 THEN DO; RETURN FALSE; FI
00684  3       ENDDO
00685  2       RETURN TRUE;
00686  2       ENDFUNCTION ST$FIND;
00687  1
00688  1    ST$DELETE: PROCEDURE;
00689  2       /* DELETE THE SYMBOL TABLE ENTRY AT ST$ENTRY$PTR */
00690  2       /* BUT FIRST FREE THE NAME AND VALUE CHAINS */
00691  2       CALL FREE$CHAIN(ST$ENTRY(ST$NAME$INDEX));
00692  2       CALL FREE$CHAIN(ST$ENTRY(ST$VALUE$INDEX));
00693  2       ST$PREVIOUS=ST$ENTRY;
00694  2       CALL FS$FREE(ST$ENTRY$PTR);
00695  2       ENDPROCEDURE ST$DELETE;
00696  1
00697  1    ST$CREATE: PROCEDURE(NAME$PTR);
00698  2       /* CREATE AN ENTRY AT HASH$INDEX, LEAVE ST$ENTRY$PTR SET */
00699  2       /* VALUE PTR TO BE FILLED  IN LATER BY ST$ADD$VALUE */
00700  2       DECLARE NAME$PTR ADDRESS;
00701  2
00702  2       ST$ENTRY$PTR=FS$GET;
00703  2       ST$ENTRY=HASH$TABLE(HASH$INDEX);
00704  2       HASH$TABLE(HASH$INDEX)=ST$ENTRY$PTR;
00705  2       ST$ENTRY(ST$NAME$INDEX)=0;
00706  2       CALL COPY$VALUE(NAME$PTR,ST$ENTRY$PTR+ST$NAME$OFFSET);
00707  2       ST$ENTRY(ST$VALUE$INDEX)=0;
00708  2       ENDPROCEDURE ST$CREATE;
00709  1
00710  1    ST$ADD$VALUE: PROCEDURE(ST$VALUE$PTR);
00711  2       /* FILL IN THE VALUE PTR IN ENTRY AT ST$ENTRY$PTR */
00712  2       /* ALSO RESET FORMPTR */
00713  2       DECLARE ST$VALUE$PTR ADDRESS;
00714  2
00715  2       CALL FREE$CHAIN(ST$ENTRY(ST$VALUE$INDEX));
00716  2       ST$ENTRY(ST$VALUE$INDEX)=ST$VALUE$PTR;
00717  2       ST$ENTRY(ST$CH$PTR$INDEX)=ST$VALUE$PTR+1;
00718  2       ENDPROCEDURE ST$ADD$VALUE;
00719  1
00720  1    ST$REINIT: PROCEDURE;
00721  2       /* DEFINE THE IDLE MACRO $IM = $(PS,$(RS,.))  */
00722  2
00723  2       DECLARE IM DATA
00724  2          (FUN$CH,LEFT$DELIM,LIT$P,LIT$S,ARG$DELIM,
00725  2           FUN$CH,LEFT$DELIM,LIT$R,LIT$S,ARG$DELIM,'.',RIGHT$DELIM,RIGHT$DELIM);
00726  2
00727  2       FIRST$PUTA$FLAG=TRUE;
00728  2       CALL PUT$CHA(TEMP$PTR:=FS$GET,FUN$CH);
00729  2       CALL PUT$CHA(0,LIT$I);
00730  2       CALL PUT$CHA(0,LIT$M);
00731  2       DUMMY=ST$FIND(TEMP$PTR);
00732  2       CALL ST$CREATE(TEMP$PTR);
00733  2       CALL FS$FREE(TEMP$PTR);
00734  2
00735  2       FIRST$PUTA$FLAG=TRUE;
00736  2       TEMP$PTR=FS$GET;
00737  2       FOR ITEMP=0 TO LAST(IM);
00738  2          CALL PUT$CHA(TEMP$PTR,IM(ITEMP));
00739  3       ENDFOR
00740  2       CALL ST$ADD$VALUE(TEMP$PTR);
00741  2       ENDPROCEDURE ST$REINIT;
00742  1
00743  1
```

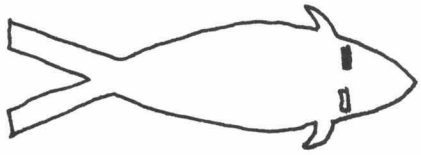


```
00744  1                /*** GENERAL UTILITIES FOR SCAN AND PRIMITIVE FUNCTIONS ***/
00745  1
00746  1    THIS$NS: FUNCTION ADDRESS;
00747  2       DECLARE THIS ADDRESS;
00748  2       THIS=NS$STACK(NS$STACK$INDEX);
00749  2       NS$STACK$INDEX=NS$STACK$INDEX+1;
00750  2       RETURN THIS;
00751  2       ENDFUNCTION THIS$NS;
00752  1
00753  1    NEXT$NS: FUNCTION ADDRESS;
00754  2       RETURN NS$STACK(NS$STACK$INDEX:=NS$STACK$INDEX+1);
00755  2       ENDFUNCTION NEXT$NS;
00756  1
00757  1
00758  1                /*** DEBUG AIDS  ***/
00759  1
00760  1    TRACE$PRINT: PROCEDURE;
00761  2       DO FOREVER;
00762  2          IF (VALUE$PTR:=THIS$NS) THEN DO;
00763  3             CALL CRLF;
00764  4             NS$STACK$INDEX=NS$SAVE$INDEX;
00765  4             RETURN;
00766  4          FI
00767  3          CALL PRINT$STRING(VALUE$PTR); CALL PRINT$CH(5CH);
00768  3       ENDDO
00769  2       ENDPROCEDURE TRACE$PRINT;
00770  1
00771  1    TURN$TRACE$ON: PROCEDURE;
00772  2       TRACE$FLAG=SHL(TRACE$FLAG,1) OR 1;
00773  2       CPM$PRINT$FLAG=TRUE;
00774  2       ENDPROCEDURE TURN$TRACE$ON;
00775  1
00776  1    TURN$TRACE$OFF: PROCEDURE;
00777  2       TRACE$FLAG=SHR(TRACE$FLAG,1);
00778  2       ENDPROCEDURE TURN$TRACE$OFF;
00779  1
00780  1    PROCESS$CON$BREAK: PROCEDURE;
00781  2       DECLARE CH BYTE;
00782  2
00783  2       IF NOT ST$FIND(.AT$CB) THEN DO; CALL ST$CREATE(.AT$CB); FI
00784  2       FIRST$PUTA$FLAG=TRUE;
00785  2       CALL PUT$CHA(VALUE$PTR:=FS$GET,CH:=READ$CH);
00786  2       CALL ST$ADD$VALUE(VALUE$PTR);
00787  2
00788  2       IF CH <> ESC THEN DO; RETURN; FI
00789  2       CALL PRINT$MESS(.' T N I:  $');
00790  2       IF (CH:=READ$CH) = LIT$T THEN DO;   /* 'T' */
00791  2          CALL TURN$TRACE$ON;
00792  3       END; ELSE IF CH = LIT$N THEN DO;   /* 'N' */
00793  2          CALL TURN$TRACE$OFF;
00794  3       END; ELSE IF CH = LIT$I THEN DO;   /* 'I' */
00795  2          GO TO START;
00796  3       FI
00797  2       CALL CRLF;
00798  2       ENDPROCEDURE PROCESS$CON$BREAK;
00799  1
00800  1
00801  1                /*** UTILITIES FOR SCAN ALGORITHM ***/
00802  1
00803  1    AS$NS$INIT: PROCEDURE;
00804  2       /* LOAD THE IDLE MACRO;  RESET THE NEUTRAL STRING */
00805  2       DECLARE
00806  2          CL$IM DATA
00807  2             (FUN$CH,LEFT$DELIM,FUN$CH,LIT$I,LIT$M,RIGHT$DELIM),
00808  2          I BYTE;
00809  2
00810  2       /* RELEASE ACTIVE STRING BLOCKS, THEN SET ACTIVE TO $($IM) */
00811  2       CALL FREE$CHAIN(AS$BLOCK$PTR);
00812  2       AS$BLOCK$PTR=FS$GET;
00813  2       IF NO$EX$VALUE THEN DO;
00814  2          FIRST$PUTA$FLAG=TRUE;
00815  3          FOR I=0 TO LAST(CL$IM);
00816  3             CALL PUT$CHA(AS$BLOCK$PTR,CL$IM(I));
00817  4          ENDFOR
00818  3       END; ELSE DO;
00819  2          CALL COPY$VALUE(VALUE$PTR,AS$BLOCK$PTR);
00820  3          NO$EX$VALUE=TRUE;
00821  3       FI
00822  2       AS$CH$PTR=AS$BLOCK$PTR+2;
00823  2
00824  2       /* RESET NEUTRAL STACK TO EMPTY RELEASING THE ASSOCIATED BLOCKS */
00825  2       NS$STACK$INDEX=0;
00826  2       DO NEXT$ARG;
00827  2          CALL FREE$CHAIN(ARG$PTR);
00828  3       ENDDO
00829  2       NS$STACK$INDEX=0;
00830  2       ENDPROCEDURE AS$NS$INIT;
00831  1
```

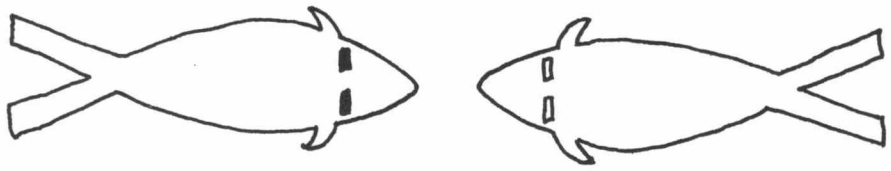
```
00832  1    AS$FREE$BLOCK: PROCEDURE;
00833  2       TEMP$PTR=AS$BLOCK$PTR;
00834  2       AS$BLOCK$PTR=AS$BLOCK;
00835  2       CALL FS$FREE(TEMP$PTR);
00836  2       AS$CH$PTR=AS$BLOCK$PTR+2;
00837  2       ENDPROCEDURE AS$FREE$BLOCK;
00838  1
00839  1    AS$NEXT$CH: FUNCTION(MOVE$PTR$FLAG) BYTE;
00840  2       /* RETURN THE NEXT 7-BIT CHARACTER FROM THE ACTIVE STRING */
00841  2       /* AS$CH$PTR IS THE ADDRESS OF THE NEXT CHAR TO BE RETURNED */
00842  2       /*    OR JUST OFF THE END OF A BLOCK */
00843  2       /* PAST CHARACTERS ARE REPLACED WITH  'PHI' (8-BIT NULL)*/
00844  2       DECLARE
00845  2          (MOVE$PTR$FLAG,CH) BYTE,
00846  2          OLD$BLOCK$PTR ADDRESS;
00847  2
00848  2       NEXT: FUNCTION BYTE;
00849  3          IF (LOW(AS$CH$PTR) AND NOT$PTR$MASK) = 0 THEN DO;
00850  3             CALL AS$FREE$BLOCK;
00851  4             IF HIGH(AS$BLOCK$PTR) = 0 THEN DO;
00852  4                GO TO START;
00853  5             FI
00854  4          FI
00855  3          RETURN AS$CH;
00856  3          ENDFUNCTION NEXT;
00857  2
00858  2       /* SKIP ANY 8-BIT CHARACTERS */
00859  2       DO WHILE ROL(CH:=NEXT,1); AS$CH$PTR=AS$CH$PTR+1; ENDDO
00860  2       IF MOVE$PTR$FLAG THEN DO;
00861  2          AS$CH=PHI;
00862  3          AS$CH$PTR=AS$CH$PTR+1;
00863  3       FI
00864  2       RETURN CH;
00865  2       ENDFUNCTION AS$NEXT$CH;
00866  1
00867  1    START$NEXT$ARGUMENT: PROCEDURE(FLAGS);
00868  2       /* INITIALIZE THE NEXT NEUTRAL STRING STACK ENTRY TO THE NULL STRING */
00869  2       /* ALSO SET FLAGS WHICH INDICATE ACTIVE OR NEUTRAL MODE */
00870  2       DECLARE FLAGS BYTE;
00871  2
00872  2       NS$STACK(NS$STACK$INDEX:=NS$STACK$INDEX+1),NS$CH$PTR=FS$GET OR FLAGS;
00873  2       NS$CH$PTR=(NS$CH$PTR AND PTR$MASK)+2;
00874  2       NS$ARG$BREAK=FALSE;
00875  2       IF NS$STACK$INDEX=LAST(NS$STACK)-4 THEN DO;
00876  2          CALL MON1(9,.'NS STACK OVERFLOW  $');
00877  3          GO TO BDOS;
00878  3       FI
00879  2       ENDPROCEDURE START$NEXT$ARGUMENT;
00880  1
00881  1    NS$ADD$CH: PROCEDURE(CH);
00882  2       /* APPEND CH TO THE ACTIVE STRING. IF NS$ARG$BREAK IS FALSE, THEN */
00883  2       /* NS$CH$PTR IS THE ADDRESS OF THE NEXT CHARACTER TO BE ADDED */
00884  2       DECLARE
00885  2          CH BYTE,
00886  2          (BLOCK$PTR,BLOCK BASED BLOCK$PTR) ADDRESS;
00887  2
00888  2       IF NS$ARG$BREAK THEN DO;
00889  2          NS$CH$PTR=APPEND$CH(CH,NS$STACK(NS$STACK$INDEX) AND PTR$MASK);
00890  3          NS$ARG$BREAK=FALSE;
00891  3       END; ELSE DO;
00892  2          NS$CH=CH;
00893  3          IF(LOW(NS$CH$PTR:=NS$CH$PTR+1) AND NOT$PTR$MASK) = 0 THEN DO;
00894  3             BLOCK$PTR=NS$CH$PTR-BLOCK$LENGTH;
00895  4             BLOCK,NS$CH$PTR=FS$GET;
00896  4             NS$CH$PTR=NS$CH$PTR+2;
00897  4          FI
00898  3       FI
00899  2       ENDPROCEDURE NS$ADD$CH;
00900  1
00901  1    PROCESS$QUOTED$STRING: PROCEDURE;
00902  2       DECLARE (DELIM$LEVEL,LAST$DELIM$WAS$LEFT,CH) BYTE;
00903  2
00904  2       DELIM$LEVEL=0;
00905  2       LAST$DELIM$WAS$LEFT=TRUE;
00906  2       DO FOREVER;
00907  2          IF (CH:=AS$NEXT$CH(MOVE$PTR)) = LEFT$DELIM THEN DO;
00908  3             IF LAST$DELIM$WAS$LEFT THEN DO; DELIM$LEVEL=DELIM$LEVEL+1; FI
00909  4             LAST$DELIM$WAS$LEFT=TRUE;
00910  4          END; ELSE IF CH = RIGHT$DELIM THEN DO;
00911  3             IF LAST$DELIM$WAS$LEFT THEN DO; LAST$DELIM$WAS$LEFT=FALSE;
00912  5             END; ELSE DO; DELIM$LEVEL=DELIM$LEVEL-1;
00913  5             FI
00914  4             IF DELIM$LEVEL = 0 THEN DO; RETURN; FI
00915  4          FI
00916  3          CALL NS$ADD$CH(CH);
00917  3       ENDDO
00918  2       ENDPROCEDURE PROCESS$QUOTED$STRING;
00919  1
```



```
00920  1    PROCESS$BEGIN$FUNCTION: PROCEDURE;
00921  2       DECLARE
00922  2          BLOCK$PTR ADDRESS,
00923  2          (BLOCK BASED BLOCK$PTR,CH) BYTE;
00924  2
00925  2       IF (CH:=AS$NEXT$CH(DONT$MOVE$PTR)) = FUN$CH THEN DO;
00926  2          DUMMY=AS$NEXT$CH(MOVE$PTR);
00927  3          IF (CH:=AS$NEXT$CH(DONT$MOVE$PTR)) = LEFT$DELIM THEN DO;  /* NEUTRAL */
00928  3             DUMMY=AS$NEXT$CH(MOVE$PTR);
00929  4             CALL START$NEXT$ARGUMENT(NS$FUN$BIT);
00930  4          END; ELSE DO;  /* NOT A FUNCTION, JUST TWO FUN CHARACTERS */
00931  3             CALL NS$ADD$CH(FUN$CH);
00932  4             CALL NS$ADD$CH(FUN$CH);
00933  4          FI
00934  3       END; ELSE IF CH = LEFT$DELIM THEN DO;  /* ACTIVE FUNCTION CALL */
00935  2          DUMMY=AS$NEXT$CH(MOVE$PTR);
00936  3          CALL START$NEXT$ARGUMENT(NS$FUN$BIT OR NS$ACTIVE$BIT);
00937  3       END; ELSE DO;  /* NOT A FUNCTION, JUST ONE FUN CHARACTER */
00938  2          CALL NS$ADD$CH(FUN$CH);
00939  3       FI
00940  2       ENDPROCEDURE PROCESS$BEGIN$FUNCTION;
00941  1
00942  1
00943  1                /*** UTILITIES FOR PRIMITIVE FUNCTIONS ***/
00944  1
00945  1    BINARY$TO$STRING: PROCEDURE(N);
00946  2       /* NOTE: VALUE$PTR IS USED BY PROCESS$END$FUNCTION */
00947  2       DECLARE
00948  2          N ADDRESS,
00949  2          D(6) BYTE,
00950  2          (I,K,NEGATIVE) BYTE;
00951  2
00952  2       NEGATIVE=FALSE;
00953  2       I=-1;
00954  2       IF ROL(HIGH(N),1) THEN DO;
00955  2          NEGATIVE=TRUE;
00956  3          N=-N;
00957  3       END; ELSE IF N = 0 THEN DO;
00958  2          I=0;
00959  3          D(0)='0';
00960  3       FI
00961  2
00962  2       DO WHILE N > 0;
00963  2          I=I+1;
00964  3          D(I)=(N MOD 10)+'0';
00965  3          N=N/10;
00966  3       ENDDO
00967  2       IF NEGATIVE THEN DO; I=I+1; D(I)='-'; FI
00968  2
00969  2       VALUE$PTR=FS$GET;
00970  2       FIRST$PUTA$FLAG=TRUE;
00971  2       FOR K=0 TO I;
00972  2          CALL PUT$CHA(VALUE$PTR,D(I-K));
00973  3       ENDFOR
00974  2       ENDPROCEDURE BINARY$TO$STRING;
00975  1
00976  1    STRING$TO$BINARY: FUNCTION(BLOCK$PTR) ADDRESS;
00977  2       DECLARE
00978  2          (BLOCK$PTR,N) ADDRESS,
00979  2          (CH,NEGATIVE) BYTE;
00980  2
00981  2       N,NEGATIVE=0;
00982  2       FIRST$GETA$FLAG=TRUE;
00983  2       DO WHILE (CH:=GET$CHA(BLOCK$PTR AND PTR$MASK)) <> NIL;
00984  2          IF CH = '-' THEN DO;
00985  3             NEGATIVE=TRUE;
00986  4          END; ELSE IF (CH:=CH-'0') < 10 THEN DO;
00987  3             N=SHL(N,3)+N+N+CH;
00988  4          FI
00989  3       ENDDO
00990  2       IF NEGATIVE THEN DO; RETURN -N; FI
00991  2       RETURN N;
00992  2       ENDFUNCTION STRING$TO$BINARY;
00993  1
00994  1    CH$TO$HEX: FUNCTION(CH) BYTE;
00995  2       DECLARE CH BYTE;
00996  2       IF CH <= '9' THEN DO; RETURN CH-'0'; FI
00997  2       IF CH < 47H THEN DO; RETURN CH-37H; FI  /* CH-'A'+10 */
00998  2       RETURN CH-57H;  /* LOWER CASE A-F */
00999  2       ENDFUNCTION CH$TO$HEX;
01000  1
01001  1    HEX$TO$CH: FUNCTION(HALF$BYTE) BYTE;
01002  2       DECLARE HALF$BYTE BYTE;
01003  2       IF HALF$BYTE < 10 THEN DO; RETURN HALF$BYTE+'0'; FI
01004  2       RETURN HALF$BYTE+57H;  /* 57H = 'A'-10 (LOWER CASE A-F) */
01005  2       ENDFUNCTION HEX$TO$CH;
01006  1
```

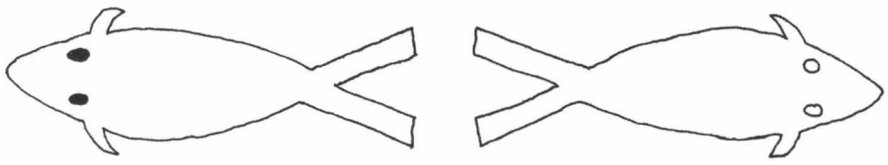


```
01007  1    NUMBER$OF$ARGS: FUNCTION BYTE;
01008  2       /* COUNT THE NUMBER OF ARGS FOR THE CURRENT FUNCTION */
01009  2       DECLARE N BYTE;
01010  2       N=1;
01011  2       DO NEXT$ARG; N=N+1; ENDDO
01012  2       NS$STACK$INDEX=NS$SAVE$INDEX;
01013  2       RETURN N;
01014  2       ENDFUNCTION NUMBER$OF$ARGS;
01015  1
01016  1    SS$MATCH: FUNCTION BYTE;
01017  2       /* SUBSTRING MATCH TEST FOR SS, IN */
01018  2       /* IMPLICIT INPUT:  SS$CHA$PTR (START SEARCH HERE), ARG$PTR (SUBSTRING) */
01019  2       /* IMPLICIT OUTPUT: SS$CHB$PTR, SS$CHC$PTR (BOUNDS OF MATCHED SUBSTRING) */
01020  2       /*    SS$CHB$PTR IS THE CHAR BEFORE THE FIRST MATCHED CHARACTER */
01021  2       /*    SS$CHC$PTR IS THE LAST CHARACTER OF THE MATCH */
01022  2       DECLARE
01023  2          (CHA,CHB) BYTE,
01024  2          RESTART$PTR ADDRESS;
01025  2
01026  2       MATCH$HERE: FUNCTION BYTE;
01027  3          FIRST$GETB$FLAG=TRUE;
01028  3          DO FOREVER;
01029  3             /* GET NEXT 7-BIT CHAR FROM SUBSTRING. IF NIL THEN HAVE MATCHED */
01030  3             DO WHILE ROL(CHB:=GET$CHB(ARG$PTR),1);
01031  4                IF CHB = NIL THEN DO; RETURN TRUE; FI
01032  5             ENDDO
01033  4             /* GET NEXT 7-BIT CHAR FROM STRING. FAIL IF FIND 8-BIT <> PHI */
01034  4             DO WHILE ROL(CHA:=GET$CHA(0),1);
01035  4                IF CHA <> PHI THEN DO; RETURN FALSE; FI
01036  5             ENDDO
01037  4             IF CHA <> CHB THEN DO; RETURN FALSE; FI
01038  4          ENDDO
01039  3          ENDFUNCTION MATCH$HERE;
01040  2
01041  2       NEXT$START: FUNCTION BYTE;
01042  3          GET$CHA$CH$PTR=RESTART$PTR;
01043  3          DUMMY=GET$CHA(0);
01044  3          DO FOREVER;
01045  3             RESTART$PTR=GET$CHA$CH$PTR;
01046  4             IF ROL(CHA:=GET$CHA(0),1) THEN DO;
01047  4                IF CHA = NIL THEN DO; RETURN FALSE; FI
01048  5             END; ELSE DO;
01049  4                RETURN TRUE;
01050  5             FI
01051  4          ENDDO
01052  3          ENDFUNCTION NEXT$START;
01053  2
01054  2       RESTART$PTR=SS$CHA$PTR;
01055  2       IF NULL$TEST(ARG$PTR) THEN DO; RETURN FALSE; FI
01056  2       DO FOREVER;
01057  2          GET$CHA$CH$PTR=RESTART$PTR;
01058  3          IF MATCH$HERE THEN DO;
01059  3             SS$CHB$PTR=RESTART$PTR;
01060  4             SS$CHC$PTR=GET$CHA$CH$PTR;
01061  4             RETURN TRUE;
01062  4          FI
01063  3          IF CHA = NIL THEN DO; RETURN FALSE; FI
01064  3          /* RESTART AT THE NEXT 7-BIT CHAR */
01065  3          IF NOT NEXT$START THEN DO; RETURN FALSE; FI
01066  3       ENDDO
01067  2       ENDFUNCTION SS$MATCH;
01068  1
01069  1
01070  1                /*** PRIMITIVE FUNCTIONS AND PROCESS$END$FUNCTION ***/
01071  1
01072  1    /* FUNCTIONS ARE ENTERED WITH NS$STACK$INDEX CORRESPONDING TO FIRST ARGUMENT */
01073  1    /* VALUE$PTR IS PRESET TO NULL$STRING$PTR FOR DEFAULT VALUE */
01074  1
01075  1    PF$NS: PROCEDURE;
01076  2       /* 'NAME STRING'  $(NS,NAME,VALUE,NAME,VALUE,...) */
01077  2       /* REUSES ANY EXISTING SYMBOL TABLE ENTRIES */
01078  2
01079  2       DO NEXT$ARG;
01080  2          IF NOT ST$FIND(ARG$PTR) THEN DO; CALL ST$CREATE(ARG$PTR); FI
01081  3          IF (ARG$PTR:=NEXT$NS) THEN DO;
01082  3             CALL ST$ADD$VALUE(FS$GET);  /* SUPPLY NULL VALUE IF NO ARGUMENT */
01083  4             RETURN;
01084  4          END; ELSE DO;
01085  3             CALL ST$ADD$VALUE(ARG$PTR);
01086  4             NS$STACK(NS$STACK$INDEX)=NULL$STRING$PTR; /*SO WON'T FREE VALUE*/
01087  4          FI
01088  3       ENDDO
01089  2       ENDPROCEDURE PF$NS;
01090  1
```



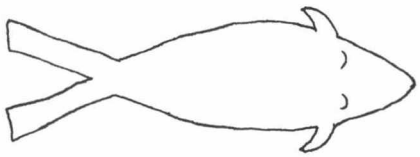
```
01091  1    PF$AS: PROCEDURE;
01092  2       /* 'APPEND TO STRING'  $(AS,NAME,SUFFIX,NAME,SUFFIX,...) */
01093  2       DECLARE CH BYTE;
01094  2
01095  2       DO NEXT$ARG;
01096  2          IF NOT ST$FIND(ARG$PTR) THEN DO;
01097  3             CALL ST$CREATE(ARG$PTR);
01098  4             CALL ST$ADD$VALUE(FS$GET);
01099  4          FI
01100  3
01101  3          IF LOW(ARG$PTR:=NEXT$NS) THEN DO; RETURN; FI
01102  3
01103  3          FIRST$GETA$FLAG=TRUE;
01104  3          IF (CH:=GET$CHA(ARG$PTR)) <> NIL THEN DO;
01105  3             PUT$CHA$CH$PTR=APPEND$CH(CH,ST$ENTRY(ST$VALUE$INDEX));
01106  4             PUT$CHA$CURRENT$BLOCK$PTR=PUT$CHA$CH$PTR AND PTR$MASK;
01107  4             FIRST$PUTA$FLAG=FALSE;
01108  4             DO WHILE (CH:=GET$CHA(0)) <> NIL;
01109  4                CALL PUT$CHA(0,CH);
01110  5             ENDDO
01111  4          FI
01112  3       ENDDO
01113  2       ENDPROCEDURE PF$AS;
01114  1
01115  1
01116  1    PF$SS: PROCEDURE;
01117  2       /* 'SEGMENT STRING'  $(SS,NAME,ARG,ARG,...) */
01118  2       /* NO MORE THAN 64 ARGS */
01119  2       DECLARE (ARG$NUMBER,MORE$FILL) BYTE;
01120  2
01121  2       PUT$CHA$SS: PROCEDURE(CHA);
01122  3          /* SPECIAL PUT$CHA FOR SEGMENT STRING */
01123  3          DECLARE
01124  3             (CH BASED PUT$CHA$CH$PTR,CHA) BYTE,
01125  3             CURRENT$BLOCK$PTR ADDRESS,
01126  3             CURRENT$BLOCK BASED CURRENT$BLOCK$PTR ADDRESS;
01127  3
01128  3          CH=CHA;
01129  3          IF (LOW(PUT$CHA$CH$PTR:=PUT$CHA$CH$PTR+1) AND NOT$PTR$MASK)
01130  3             = 0 THEN DO;
01131  3             CURRENT$BLOCK$PTR = PUT$CHA$CH$PTR-BLOCK$LENGTH;
01132  4             PUT$CHA$CH$PTR=CURRENT$BLOCK+2;
01133  4          FI
01134  3          ENDPROCEDURE PUT$CHA$SS;
01135  2
01136  2       IF NOT ST$FIND(NEXT$NS) THEN DO; RETURN; FI
01137  2       ARG$NUMBER=-1;
01138  2       DO NEXT$ARG;
01139  2          ARG$NUMBER=ARG$NUMBER+1;
01140  3          SS$CHA$PTR=ST$ENTRY(ST$VALUE$INDEX)+1;
01141  3          DO WHILE SS$MATCH;
01142  3             GET$CHA$CH$PTR=SS$CHB$PTR;
01143  4             DUMMY=GET$CHA(0);
01144  4             PUT$CHA$CH$PTR=GET$CHA$CH$PTR;
01145  4             MORE$FILL=(PUT$CHA$CH$PTR <> SS$CHC$PTR);
01146  4             CALL PUT$CHA$SS(SS$MASK OR ARG$NUMBER);  /* INSERT SEG. NUM. */
01147  4             DO WHILE MORE$FILL;
01148  4                MORE$FILL=(PUT$CHA$CH$PTR <> SS$CHC$PTR);
01149  5                CALL PUT$CHA$SS(PHI);
01150  5             ENDDO
01151  4             SS$CHA$PTR=SS$CHC$PTR;
01152  4          ENDDO
01153  3       ENDDO
01154  2       ENDPROCEDURE PF$SS;
01155  1
01156  1    PF$GC: PROCEDURE;
01157  2       /* 'GET CHARACTERS'  $(GC,NAME,N,S,F) */
01158  2       /* IF ONLY TWO ARGS, VALUE IS LENGTH OF THE NAMED STRING */
01159  2       /* IF NO FORM CALLED NAME, THEN VALUE IS NULL */
01160  2       DECLARE
01161  2          (GC$PTR,N$PTR,N,MARK$PTR,FORM$PTR,
01162  2           CURRENT$CH$PTR,BACKUP$TO) ADDRESS,
01163  2          (CH,NEG$COUNT$FAILED) BYTE;
01164  2
01165  2       ANOTHER$CH: FUNCTION BYTE;
01166  3          IF N = 0 THEN DO; RETURN FALSE; FI
01167  3          IF (CH:=GET$CHA(0)) <> NIL THEN DO; RETURN TRUE; FI
01168  3          RETURN FALSE;
01169  3          ENDFUNCTION ANOTHER$CH;
01170  2
01171  2
01172  2       GC$LENGTH: FUNCTION ADDRESS;
01173  3          DECLARE M ADDRESS;
01174  3          M=0;
01175  3          FIRST$GETA$FLAG=TRUE;
01176  3          TEMP$PTR=ST$ENTRY(ST$VALUE$INDEX);
01177  3          DO WHILE (CH:=GET$CHA(TEMP$PTR)) <> NIL AND
01178  3             GET$CHA$CH$PTR <> MARK$PTR;
01179  3             IF NOT ROL(CH,1) THEN DO; M=M+1; FI
01180  4          ENDDO
01181  3          RETURN M;
01182  3          ENDFUNCTION GC$LENGTH;
01183  2
```

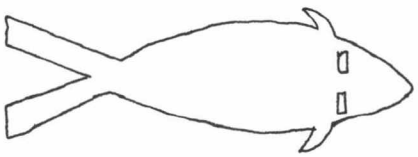
```
01184  2       IF NOT ST$FIND(.AT$GC) THEN DO; CALL ST$CREATE(.AT$GC); FI
01185  2       CALL ST$ADD$VALUE(GC$PTR:=FS$GET);
01186  2
01187  2       NAME$PTR=NEXT$NS;
01188  2       N$PTR=NEXT$NS;
01189  2       S$PTR=NEXT$NS;
01190  2       F$PTR=NEXT$NS;
01191  2
01192  2       IF NOT ST$FIND(NAME$PTR) THEN DO; RETURN; FI
01193  2       IF LOW(N$PTR) THEN DO;  /* RETURN STRING LENGTH IF NO THIRD ARGUMENT */
01194  2          MARK$PTR=0;
01195  3          CALL BINARY$TO$STRING(GC$LENGTH);
01196  3          RETURN;
01197  3       FI
01198  2
01199  2       N=STRING$TO$BINARY(N$PTR);
01200  2       NEG$COUNT$FAILED=FALSE;
01201  2       GET$CHA$CH$PTR,FORM$PTR=ST$ENTRY(ST$CH$PTR$INDEX);
01202  2       /* NEGATIVE COUNT NOT IMPLEMENTED */
01203  2       FIRST$PUTA$FLAG=TRUE;
01204  2       DO WHILE ANOTHER$CH;
01205  2          IF NOT ROL(CH,1) THEN DO;
01206  3             N=N-1;
01207  4             FORM$PTR=GET$CHA$CH$PTR;
01208  4             CALL PUT$CHA(GC$PTR,CH);
01209  4          FI
01210  3       ENDDO
01211  2
01212  2       ST$ENTRY(ST$CH$PTR$INDEX)=FORM$PTR;
01213  2       VALUE$PTR=S$PTR;
01214  2       IF N <> 0 OR NEG$COUNT$FAILED THEN DO; VALUE$PTR=F$PTR; FI
01215  2       IF LOW(S$PTR) THEN DO;  /* RETURN $GC IF NO S OR F ARGS */
01216  2          CALL COPY$VALUE(GC$PTR,VALUE$PTR:=FS$GET);
01217  3       FI
01218  2       ENDPROCEDURE PF$GC;
01219  1
01220  1    PF$IN: PROCEDURE;
01221  2       /* 'INITIAL'  $(IN,NAME,SUBSTRING,S,F)  $IN SET TO 'VALUE' */
01222  2       /* ADD... IF SUBSTRING IS NULL, THEN LOOK FOR THE NEXT NON-NULL SEGMENT */
01223  2       DECLARE IN$PTR ADDRESS, CH BYTE;
01224  2
01225  2       NAME$PTR=NEXT$NS;
01226  2       ARG$PTR=NEXT$NS;  /* USED BY SS$MATCH */
01227  2       S$PTR=NEXT$NS;
01228  2       F$PTR,VALUE$PTR=NEXT$NS;
01229  2
01230  2       IF NOT ST$FIND(.AT$IN) THEN DO; CALL ST$CREATE(.AT$IN); FI
01231  2       CALL ST$ADD$VALUE(IN$PTR:=FS$GET);
01232  2
01233  2       IF NOT ST$FIND(NAME$PTR) THEN DO; RETURN; FI
01234  2       SS$CHA$PTR=ST$ENTRY(ST$CH$PTR$INDEX);
01235  2
01236  2       FIRST$GETA$FLAG=FALSE;
01237  2       FIRST$PUTA$FLAG=TRUE;
01238  2       IF SS$MATCH THEN DO;
01239  2          GET$CHA$CH$PTR=SS$CHA$PTR;
01240  3          DO WHILE GET$CHA$CH$PTR <> SS$CHB$PTR;
01241  3             IF NOT ROL(CH:=GET$CHA(0),1) THEN DO; CALL PUT$CHA(IN$PTR,CH); FI
01242  4          ENDDO
01243  3          VALUE$PTR=S$PTR;
01244  3       END; ELSE DO;
01245  2          GET$CHA$CH$PTR,SS$CHC$PTR=SS$CHA$PTR;
01246  3          FIRST$PUTA$FLAG=TRUE;
01247  3          DO WHILE (CH:=GET$CHA(0)) <> NIL;
01248  3             IF NOT ROL(CH,1) THEN DO;
01249  4                SS$CHC$PTR=GET$CHA$CH$PTR;
01250  5                CALL PUT$CHA(IN$PTR,CH);
01251  5             FI
01252  4          ENDDO
01253  3       FI
01254  2       ST$ENTRY(ST$CH$PTR$INDEX)=SS$CHC$PTR;
01255  2       IF LOW(S$PTR) THEN DO;  /* RETURN $IN IF NO S OR F ARGS */
01256  2          CALL COPY$VALUE(IN$PTR,VALUE$PTR:=FS$GET);
01257  3       FI
01258  2       ENDPROCEDURE PF$IN;
01259  1
01260  1    PF$CR: PROCEDURE;
01261  2       /* 'CALL RESTORE'  $(CR,NAME) */
01262  2       IF NOT ST$FIND(NEXT$NS) THEN DO; RETURN; FI
01263  2       ST$ENTRY(ST$CH$PTR$INDEX)=ST$ENTRY(ST$VALUE$INDEX)+1;
01264  2       ENDPROCEDURE PF$CR;
01265  1
```



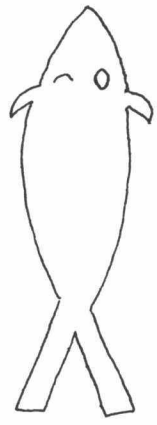
```
01266  1    PF$EQ: PROCEDURE;
01267  2       /* 'EQUAL'  $(EQ,A, B,C, B,C,...,B,C) */
01268  2       /*       OR  $(EQ,A, B,C, B,C,...,B) */
01269  2       DECLARE
01270  2          (ARGA,ARGB,ARGC) ADDRESS;
01271  2
01272  2       ARGA=NEXT$NS;
01273  2       DO FOREVER;
01274  2          IF LOW(ARGB:=NEXT$NS) THEN DO; RETURN; FI
01275  3          IF LOW(ARGC:=NEXT$NS) THEN DO;
01276  3             VALUE$PTR=ARGB;
01277  4             RETURN;
01278  4          END; ELSE IF STRING$EQUALITY(ARGA,ARGB) THEN DO;
01279  3             VALUE$PTR=ARGC;
01280  4             RETURN;
01281  4          FI
01282  3       ENDDO
01283  2       ENDPROCEDURE PF$EQ;
01284  1
01285  1    PF$GE: PROCEDURE;
01286  2       /* 'GREATER THAN OR EQUAL TO'  $(GE,N,M,S,F) */
01287  2       /* '+8000H' IS TO MAP SIGNED HALF-RANGE ONTO FULL-RANGE INTEGERS */
01288  2       IF STRING$TO$BINARY(NEXT$NS)+8000H >= STRING$TO$BINARY(NEXT$NS)+8000H
01289  2       THEN DO; VALUE$PTR=NEXT$NS; RETURN; FI
01290  2       VALUE$PTR=NEXT$NS;
01291  2       VALUE$PTR=NEXT$NS;
01292  2       ENDPROCEDURE PF$GE;
01293  1
01294  1    PF$NT: PROCEDURE;
01295  2       /* 'NULL TEST'  $(NT,NAME,S,F) */
01296  2       NAME$PTR=NEXT$NS AND PTR$MASK;
01297  2       S$PTR,VALUE$PTR=NEXT$NS;
01298  2       F$PTR=NEXT$NS;
01299  2
01300  2       IF ST$FIND(NAME$PTR) THEN DO;
01301  2          IF NOT NULL$TEST(ST$ENTRY(ST$VALUE$INDEX)) THEN DO;
01302  3             VALUE$PTR=F$PTR;
01303  4          FI
01304  3       FI
01305  2       ENDPROCEDURE PF$NT;
01306  1
01307  1    PF$CL: PROCEDURE(STACK$OFFSET);
01308  2       /* 'CALL'  $(CL,NAME,ARG,ARG,...) OR $(NAME,ARG,ARG,...) */
01309  2       DECLARE
01310  2          STACK$OFFSET BYTE,  /* 0 IF FIRST ARG IS 'CL', -1 OTHERWISE */
01311  2          LAST$ARG$NUMBER BYTE,  /* ARG(1),...,ARG(LAST$ARG$NUMBER) */
01312  2          (CH,CHB,SEG$NUMBER) BYTE;
01313  2
01314  2       LAST$ARG$NUMBER=NUMBER$OF$ARGS-STACK$OFFSET-2;
01315  2       NS$STACK$INDEX=NS$STACK$INDEX+STACK$OFFSET;
01316  2       IF NOT ST$FIND(NEXT$NS AND PTR$MASK) THEN DO; RETURN; FI
01317  2       VALUE$PTR=FS$GET;
01318  2       FIRST$GETA$FLAG,FIRST$PUTA$FLAG=TRUE;
01319  2       TEMP$PTR=ST$ENTRY(ST$VALUE$INDEX);
01320  2       DO WHILE (CH:=GET$CHA(TEMP$PTR)) <> NIL;
01321  2          IF NOT ROL(CH,1) THEN DO;
01322  3             CALL PUT$CHA(VALUE$PTR,CH);
01323  4          END; ELSE IF ROL(CH,2) THEN DO;
01324  3             IF (SEG$NUMBER:=(CH XOR SS$MASK)+1) <= LAST$ARG$NUMBER THEN DO;
01325  4                TEMP$PTR=NS$STACK(NS$STACK$INDEX+SEG$NUMBER);
01326  5                FIRST$GETB$FLAG=TRUE;
01327  5                DO WHILE (CHB:=GET$CHB(TEMP$PTR)) <> NIL;
01328  5                   IF NOT ROL(CHB,1) THEN DO; CALL PUT$CHA(VALUE$PTR,CHB); FI
01329  6                ENDDO
01330  5             FI
01331  4          FI
01332  3       ENDDO
01333  2       ENDPROCEDURE PF$CL;
01334  1
01335  1    PF$EX: PROCEDURE;
01336  2       /* 'EXIT'  $(EX,DO-THIS) */
01337  2       VALUE$PTR=NEXT$NS;
01338  2       NO$EX$VALUE=NULL$TEST(VALUE$PTR);
01339  2       GO TO START;
01340  2    ENDPROCEDURE PF$EX;
01341  1
01342  1    PF$PS: PROCEDURE;
01343  2       /* 'PRINT STRING'  $(PS,ARG,ARG,...) */
01344  2       DECLARE NOT$FIRST BYTE;
01345  2       NOT$FIRST=FALSE;
01346  2       DO NEXT$ARG;
01347  2          IF NOT$FIRST THEN DO; CALL CRLF; FI
01348  3          NOT$FIRST=TRUE;
01349  3          CALL PRINT$STRING(ARG$PTR);
01350  3       ENDDO
01351  2       ENDPROCEDURE PF$PS;
01352  1
01353  1    PF$FW: PROCEDURE;
01354  2       /* 'FORM WIDTH'  $(FW,CH-PER-LINE) */
01355  2       CON$WIDTH=STRING$TO$BINARY(NEXT$NS);
01356  2       ENDPROCEDURE PF$FW;
01357  1
```

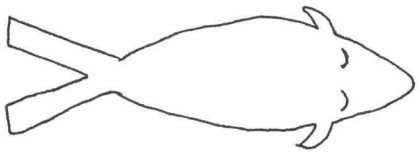
```
01358  1    PF$RS: PROCEDURE;
01359  2       /* 'READ STRING'  $(RS,PROMPT) */
01360  2       /* INPUT FROM CONSOLE OR DISK */
01361  2       DECLARE
01362  2          (K,CH) BYTE,
01363  2          NOT$COMMENT BYTE;
01364  2
01365  2       NEXT$CHAR: PROCEDURE;
01366  3          IF TBUFF$INDEX = 128 THEN DO;
01367  3             TBUFF$INDEX=0;
01368  4             IF MON2(20,.TFCB) <> 0 THEN DO;
01369  4                CALL PRINT$MESS(.'DISK EOF  $');
01370  5                CH=END$OF$FILE;
01371  5                RETURN;
01372  5             FI
01373  4          FI
01374  3          CH=TBUFF(TBUFF$INDEX);
01375  3          TBUFF$INDEX=TBUFF$INDEX+1;
01376  3          ENDPROCEDURE NEXT$CHAR;
01377  2
01378  2       VALUE$PTR=FS$GET;
01379  2       FIRST$PUTA$FLAG=TRUE;
01380  2       IF CON$IN THEN DO;
01381  2          IF NOT LOW(ARG$PTR:=NEXT$NS) THEN DO;
01382  3             CALL PRINT$STRING(ARG$PTR);
01383  4          FI
01384  3          CALL MON1(10,.CONBUF);
01385  3          CALL CRLF;
01386  3          K=1;
01387  3          DO WHILE K <= CONBUF(1);
01388  3             CALL PUT$CHA(VALUE$PTR,CONBUF(K:=K+1));
01389  4          ENDDO
01390  3       END; ELSE DO;  /* DISK INPUT. */
01391  2          NOT$COMMENT=TRUE;
01392  3          DO FOREVER;
01393  3             CALL NEXT$CHAR;
01394  4             IF CH = META$CHAR THEN DO; RETURN; FI
01395  4             IF CH = END$OF$FILE THEN DO; CON$IN=TRUE; RETURN; FI
01396  4             IF CH = COMM$DELIM THEN DO;
01397  4                NOT$COMMENT=NOT NOT$COMMENT;
01398  5             END; ELSE IF CH >= ' ' AND NOT$COMMENT THEN DO;
01399  4                CALL PUT$CHA(VALUE$PTR,CH);
01400  5             FI
01401  4          ENDDO
01402  3       FI
01403  2       ENDPROCEDURE PF$RS;
01404  1
01405  1    PF$PK: PROCEDURE;
01406  2       /* 'PACK' HEX-TO-ASCII  $(PK,HEX-STRING) */
01407  2       /* ALSO CALLED FROM PF$RO */
01408  2       DECLARE (CHA,CHB) BYTE;
01409  2
01410  2       VALUE$PTR=FS$GET;
01411  2       TEMP$PTR=NEXT$NS;
01412  2       FIRST$GETA$FLAG,FIRST$PUTA$FLAG=TRUE;
01413  2       DO FOREVER;
01414  2          /* NOTE: ASSUMES NO 8-BIT CHARACTERS OTHER THAN NIL */
01415  2          CHA=GET$CHA(TEMP$PTR);
01416  3          CHB=GET$CHA(0);
01417  3          IF CHA = NIL OR CHB = NIL THEN DO; RETURN; FI
01418  3          CALL PUT$CHA(VALUE$PTR,SHL(CH$TO$HEX(CHA),4) OR CH$TO$HEX(CHB));
01419  3       ENDDO
01420  2       ENDPROCEDURE PF$PK;
01421  1
01422  1    PF$UK: PROCEDURE;
01423  2       /* 'UNPACK'  ASCII-TO-HEX   $(UK,CH-STRING) */
01424  2       DECLARE CH BYTE;
01425  2
01426  2       IF LOW(ARG$PTR:=NEXT$NS) THEN DO; RETURN; FI
01427  2       FIRST$GETA$FLAG,FIRST$PUTA$FLAG=TRUE;
01428  2       VALUE$PTR=FS$GET;
01429  2       DO WHILE (CH:=GET$CHA(ARG$PTR)) <> NIL;
01430  2          CALL PUT$CHA(VALUE$PTR,HEX$TO$CH(ROL(CH,4) AND 0FH));
01431  3          CALL PUT$CHA(0,HEX$TO$CH(CH AND 0FH));
01432  3       ENDDO
01433  2       ENDPROCEDURE PF$UK;
01434  1
01435  1    PF$SO: PROCEDURE;
01436  2       /* 'SEND TO OCTOPUS'  $(SO,MESSAGE) */
01437  2       CALL SEND$STRING(NEXT$NS AND PTR$MASK);
01438  2       ENDPROCEDURE PF$SO;
```



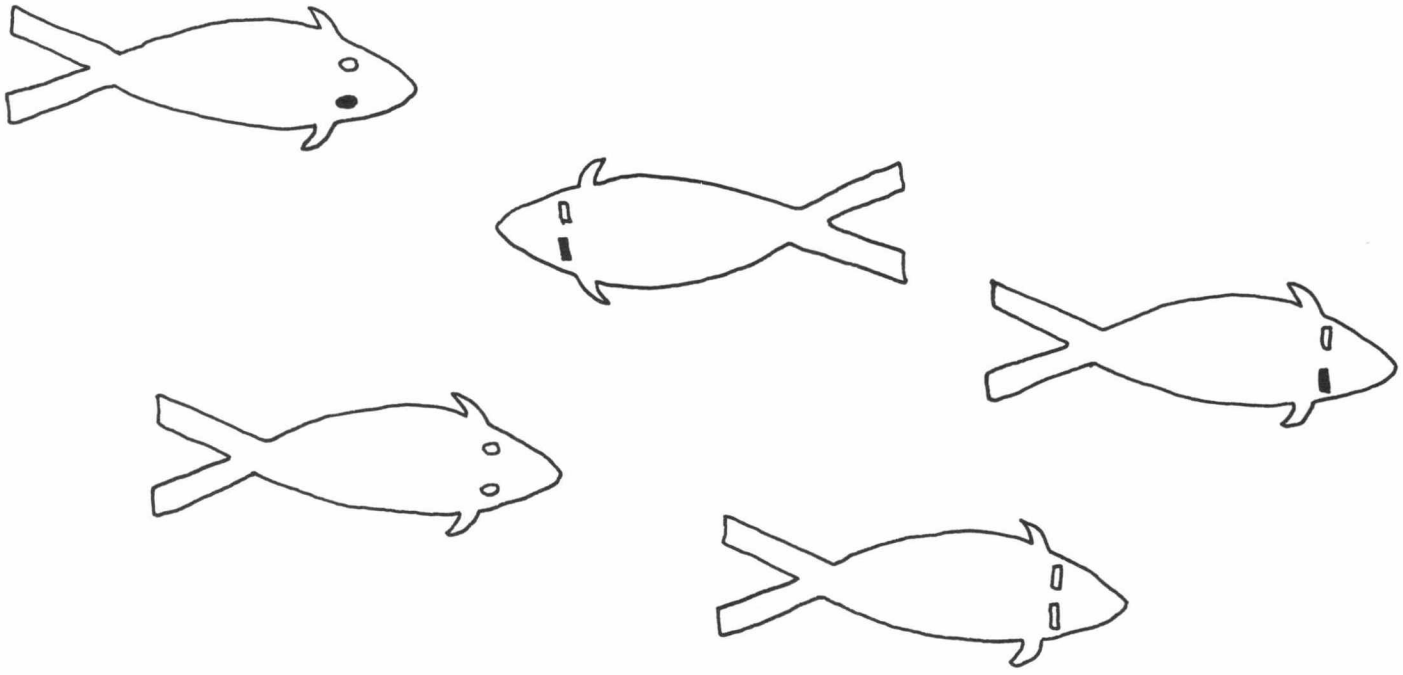


```
01439  1
01440  1    PF$RO: PROCEDURE;
01441  2       /* 'RECEIVE FROM OCTOPUS'  $(RO,END-SENTINEL,S,F,EOM) */
01442  2       /* VALUE$PTR IS SET IN RECEIVE$STRING AND TRANSFERRED TO $RO */
01443  2       DECLARE END$PTR ADDRESS;
01444  2
01445  2       IF NOT ST$FIND(.AT$RO) THEN DO; CALL ST$CREATE(.AT$RO); FI
01446  2       END$PTR=NEXT$NS;
01447  2       S$PTR=NEXT$NS;
01448  2       F$PTR=NEXT$NS;
01449  2       /* SET NEW OCTOPUS MESSAGE TERMINATOR, IF GIVEN */
01450  2       CALL PF$PK;
01451  2       FIRST$GETA$FLAG=TRUE;
01452  2       IF (ITEMP:=GET$CHA(VALUE$PTR)) <> NIL THEN DO;
01453  2          OCTO$EOM=ITEMP;
01454  3       FI
01455  2       CALL FREE$CHAIN(VALUE$PTR);
01456  2
01457  2       IF RECEIVE$STRING(END$PTR) THEN DO;
01458  2          CALL ST$ADD$VALUE(VALUE$PTR);
01459  3          VALUE$PTR=S$PTR;
01460  3       END; ELSE DO;
01461  2          CALL ST$ADD$VALUE(VALUE$PTR);
01462  3          VALUE$PTR=F$PTR;
01463  3       FI
01464  2       ENDPROCEDURE PF$RO;
01465  1
01466  1    PF$DD: PROCEDURE;
01467  2       /* 'DELETE DEFINITION'  $(DD,NAME,NAME,...) */
01468  2       DO NEXT$ARG;
01469  2          IF ST$FIND(ARG$PTR) THEN DO; CALL ST$DELETE; FI
01470  3       ENDDO
01471  2       ENDPROCEDURE PF$DD;
01472  1
01473  1    PF$AD: PROCEDURE;
01474  2       /* 'ADD'  $(AD,N,N,N,...) */
01475  2       /* ZERO OR MORE NUMERICAL ARGUMENTS */
01476  2       DECLARE SUM ADDRESS;
01477  2
01478  2       SUM=0;
01479  2       DO NEXT$ARG;
01480  2          SUM=SUM+STRING$TO$BINARY(ARG$PTR);
01481  3       ENDDO
01482  2       CALL BINARY$TO$STRING(SUM);
01483  2       ENDPROCEDURE PF$AD;
01484  1
01485  1    PF$SU: PROCEDURE;
01486  2       /* 'SUBTRACT'  $(SU,N,M) */
01487  2       /* MUST HAVE EXACTELY TWO NUMERICAL ARGUMENTS */
01488  2
01489  2       CALL BINARY$TO$STRING(
01490  2          STRING$TO$BINARY(NEXT$NS)-
01491  2          STRING$TO$BINARY(NEXT$NS));
01492  2       ENDPROCEDURE PF$SU;
01493  1
01494  1    PF$ML: PROCEDURE;
01495  2       /* 'MULTIPLY'  $(ML,N,N,N,...) */
01496  2       /* ZERO OR MORE NUMERICAL ARGUMENTS */
01497  2       DECLARE PRODUCT ADDRESS;
01498  2
01499  2       PRODUCT=1;
01500  2       DO NEXT$ARG;
01501  2          PRODUCT=PRODUCT*STRING$TO$BINARY(ARG$PTR);
01502  3       ENDDO
01503  2       CALL BINARY$TO$STRING(PRODUCT);
01504  2       ENDPROCEDURE PF$ML;
01505  1
01506  1    PF$MD: PROCEDURE;
01507  2       /* 'MOD'  $(MD,N,M) */
01508  2       /* MUST HAVE EXACTELY TWO NUMERICAL ARGUMENTS */
01509  2
01510  2       CALL BINARY$TO$STRING(
01511  2          STRING$TO$BINARY(NEXT$NS) MOD
01512  2          STRING$TO$BINARY(NEXT$NS));
01513  2       ENDPROCEDURE PF$MD;
01514  1
01515  1    PF$ZZ: PROCEDURE;
01516  2       /* 'SLEEP'  $(ZZ,TENTH SEC) */
01517  2       DECLARE N ADDRESS;
01518  2       CALL WAIT$TENTH(STRING$TO$BINARY(NEXT$NS));
01519  2       ENDPROCEDURE PF$ZZ;
01520  1
```

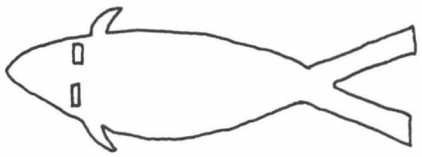




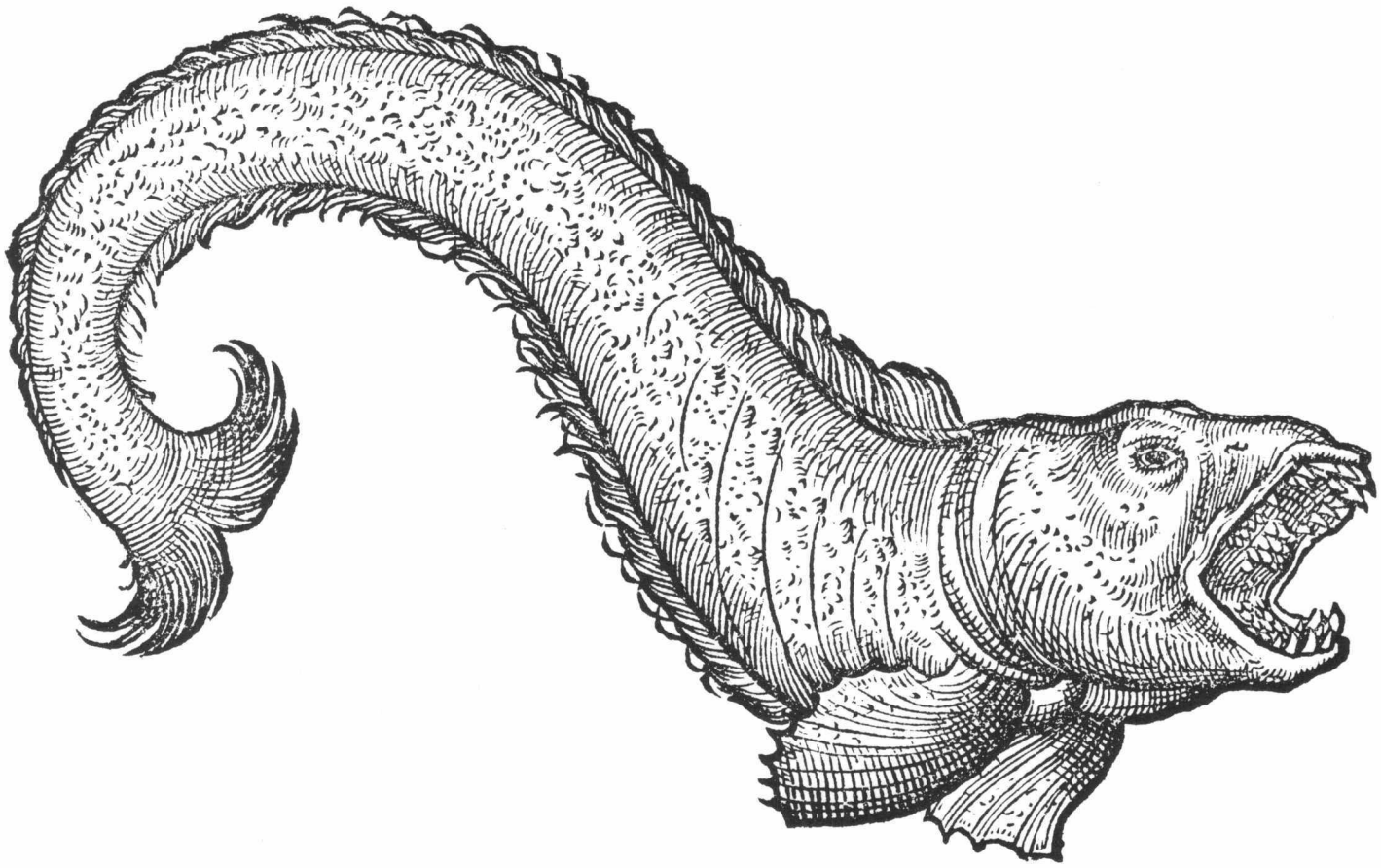
```
01521  1   PF$IO: PROCEDURE;
01522  2      /* 'INPUT-OUTPUT' PARAMETERS  $(IO,'O'OCTO,'T'TIMEOUT,'M'META) */
01523  2      DECLARE CH BYTE;
01524  2
01525  2      DO NEXT$ARG;
01526  2         FIRST$GETA$FLAG=TRUE;
01527  3         IF (CH:=GET$CHA(ARG$PTR)) = LIT$O THEN DO;   /* O: SET OCTO CHANNEL */
01528  3            OCTO$CHANNEL$A=(GET$CHA(0) = LIT$A);
01529  4         END; ELSE IF CH = LIT$T THEN DO;   /* T: SET OCTO TIMEOUT */
01530  3            OCTO$TIMEOUT=STRING$TO$BINARY(ARG$PTR);
01531  4         END; ELSE IF CH = LIT$M THEN DO;   /* M: SET META-CHARACTER */
01532  3            META$CHAR=GET$CHA(0);
01533  4         END; ELSE IF CH = LIT$P THEN DO;   /* P: SET PRINT VIA CP/M */
01534  3            CPM$PRINT$FLAG=TRUE;
01535  4         END; ELSE IF CH = LIT$D THEN DO;   /* D: SET PRINT DIRECT (HARDWARE) */
01536  3            CPM$PRINT$FLAG=FALSE;
01537  4         FI
01538  3      ENDDO
01539  2      ENDPROCEDURE PF$IO;
01540  1
01541  1   PF$DV: PROCEDURE;
01542  2      /* 'DIVIDE'  $(DV,N,M) */
01543  2      /* MUST HAVE EXACTLY TWO NUMERICAL ARGUMENTS */
01544  2
01545  2      CALL BINARY$TO$STRING(
01546  2         STRING$TO$BINARY(NEXT$NS)/
01547  2         STRING$TO$BINARY(NEXT$NS));
01548  2      ENDPROCEDURE PF$DV;
01549  1
01550  1   PF$NB: PROCEDURE;
01551  2      /* 'NUMBER OF (FREE STORAGE) BLOCKS'  $(NB) */
01552  2      DECLARE
01553  2         (BLOCK BASED TEMP$PTR,N) ADDRESS;
01554  2
01555  2      TEMP$PTR=FS$FIRST$PTR;
01556  2      N=0;
01557  2      DO WHILE HIGH(TEMP$PTR) > 0;
01558  2         N=N+1;
01559  3         TEMP$PTR=BLOCK;
01560  3      ENDDO
01561  2      CALL BINARY$TO$STRING(N);
01562  2      ENDPROCEDURE PF$NB;
01563  1
01564  1   PF$FS: PROCEDURE;
01565  2      /* 'FREE STORAGE'  $(FS,FIRST ADDRESS,LAST ADDRESS) */
01566  2      CALL FS$INIT(STRING$TO$BINARY(NEXT$NS),STRING$TO$BINARY(NEXT$NS));
01567  2      ENDPROCEDURE PF$FS;
01568  1
01569  1   PF$VN: PROCEDURE;
01570  2      /* 'VERSION NUMBER'  $(VN) */
01571  2      VALUE$PTR=FS$GET;
01572  2      FIRST$PUTA$FLAG=TRUE;
01573  2      CALL PUT$CHA(VALUE$PTR,'3');
01574  2      CALL PUT$CHA(0,'1');
01575  2      CALL PUT$CHA(0,LIT$A);
01576  2      ENDPROCEDURE PF$VN;
01577  1
01578  1
01579  1
```



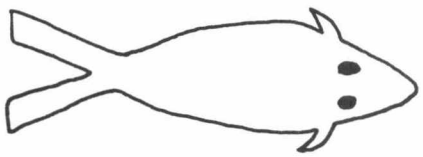
```
01580  1              /*** DISK I/O FUNCTIONS ***/
01581  1
01582  1   /* SEE 'BASIC I/O' ON PAGE 2 FOR GLOBALS */
01583  1
01584  1   SET$FILE$NAME: PROCEDURE(FILE$NAME$PTR);
01585  2      /* EXPAND  D:X.Y  FORMAT TO ELEVEN CHAR FILE NAME AND SELECT DRIVE */
01586  2      DECLARE FILE$NAME$PTR ADDRESS, (I,CH,DRIVE) BYTE;
01587  2
01588  2      TFCB(0)=0;
01589  2      FOR I=1 TO 11; TFCB(I)=' '; ENDFOR
01590  2      I,DRIVE=0; FIRST$GETA$FLAG=TRUE;
01591  2      DO WHILE (I<11) AND ((CH:=GET$CHA(FILE$NAME$PTR)) <> NIL);
01592  2         IF CH = ':' THEN DO;
01593  3            TFCB(0)=DRIVE;
01594  4            I=0;
01595  4         END; ELSE IF CH = '.' THEN DO;
01596  3            I=8;
01597  4         END; ELSE DO;
01598  3            I=I+1;
01599  4            TFCB(I)=UPPER$CASE(CH);
01600  4            DRIVE=CH;
01601  4         FI
01602  3      ENDDO
01603  2      CALL SELECT$DRIVE;
01604  2      ENDPROCEDURE SET$FILE$NAME;
01605  1
01606  1   SET$FILE$RECORD: PROCEDURE(RECORD$NO);
01607  2      /* SET EXTENT AND NR FIELDS IN TFCB */
01608  2      DECLARE RECORD$NO ADDRESS;
01609  2
01610  2      TFCB(12)=SHR(RECORD$NO,7);
01611  2      TFCB(32)=RECORD$NO AND 7FH;  /* SEE 'RANDOM ACCESS' IN CP/M MANUAL */
01612  2      ENDPROCEDURE SET$FILE$RECORD;
01613  1
01614  1   PF$RN: PROCEDURE;
01615  2      /* 'RECORD NUMBER'  $(RN) */
01616  2      CALL BINARY$TO$STRING(SHL(TFCB(12),7) OR TFCB(32));
01617  2      ENDPROCEDURE PF$RN;
01618  1
01619  1   PF$OF: PROCEDURE;
01620  2      /* 'OPEN FILE'  $(OF,FILE-NAME,NEXT-RECORD-NUMBER,S,F) */
01621  2      CALL SET$FILE$NAME(NEXT$NS);
01622  2      CALL SET$FILE$RECORD(STRING$TO$BINARY(NEXT$NS));
01623  2      VALUE$PTR=NEXT$NS;
01624  2      IF MON2(15,.TFCB) = 255 THEN DO;  /* OPEN AND TEST FOR ERROR */
01625  2         VALUE$PTR=NEXT$NS;
01626  3      FI
01627  2      ENDPROCEDURE PF$OF;
01628  1
01629  1   PF$MF: PROCEDURE;
01630  2      /* 'MAKE FILE'  $(MF,FILE-NAME,S,F) */
01631  2      CALL SET$FILE$NAME(NEXT$NS);
01632  2      CALL SET$FILE$RECORD(0);
01633  2      VALUE$PTR=NEXT$NS;
01634  2      IF MON2(22,.TFCB) = 255 THEN DO;
01635  2         VALUE$PTR=NEXT$NS;
01636  3      FI
01637  2      ENDPROCEDURE PF$MF;
01638  1
01639  1   PF$CF: PROCEDURE;
01640  2      /* 'CLOSE FILE'  $(CF,S,F) */
01641  2      VALUE$PTR=NEXT$NS;
01642  2      IF MON2(16,.TFCB) = 255 THEN DO;
01643  2         VALUE$PTR=NEXT$NS;
01644  3      FI
01645  2      ENDPROCEDURE PF$CF;
01646  1
01647  1   PF$DF: PROCEDURE;
01648  2      /* 'DESTROY FILE'  $(DF) */
01649  2      CALL MON1(19,.TFCB);
01650  2      ENDPROCEDURE;
01651  1
01652  1   PF$WF: PROCEDURE;
01653  2      /* 'WRITE FILE'  $(WF,SOURCE-NAME,PAD-CH EMPTY FLAG,S,F) */
01654  2      DECLARE (CH,I,WRITE$STATUS) BYTE;
01655  2
01656  2      STUFF$CH: PROCEDURE;
01657  3         IF WRITE$STATUS <> 0 THEN DO; RETURN; FI
01658  3         TBUFF(TBUFF$INDEX)=CH;
01659  3         IF (TBUFF$INDEX:=TBUFF$INDEX+1) = 128 THEN DO;
01660  3            WRITE$STATUS=MON2(21,.TFCB);
01661  4            TBUFF$INDEX=0;
01662  4            ST$ENTRY(ST$CH$PTR$INDEX)=GET$CHA$CH$PTR;  /* SET FORM PTR */
01663  4         FI
01664  3         ENDPROCEDURE STUFF$CH;
01665  2
01666  2      WRITE$STATUS,TBUFF$INDEX=0;
01667  2      IF NOT ST$FIND(NEXT$NS) THEN DO; RETURN; FI
01668  2      GET$CHA$CH$PTR=ST$ENTRY(ST$VALUE$INDEX)+1;
01669  2      DO WHILE (CH:=GET$CHA(0)) <> NIL;
01670  2         CALL STUFF$CH;
01671  3      ENDDO
01672  2
```



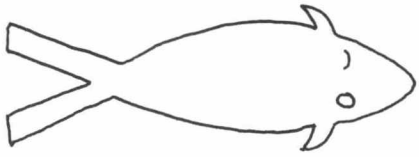
```
01673  2       FIRST$GETB$FLAG=TRUE;
01674  2       IF (CH:=GET$CHB(NEXT$NS AND PTR$MASK)) <> NIL THEN DO;
01675  2          FOR I=0 TO 127; CALL STUFF$CH; ENDFOR  /* GUARANTEE A WRITE WITH PAD CH */
01676  3       FI
01677  2       VALUE$PTR=NEXT$NS;
01678  2       IF WRITE$STATUS <> 0 THEN DO; VALUE$PTR=NEXT$NS; FI
01679  2       ENDPROCEDURE PF$WF;
01680  1
01681  1    PF$RF: PROCEDURE;
01682  2       /* 'READ FILE'  $(RF,SINK-NAME,RECORD-COUNT,S,F)  (F=EOF) */
01683  2       DECLARE RC ADDRESS;
01684  2
01685  2       IF NOT ST$FIND(ARG$PTR:=NEXT$NS) THEN DO;
01686  2          CALL ST$CREATE(ARG$PTR);
01687  3          CALL ST$ADD$VALUE(FS$GET);
01688  3       FI
01689  2
01690  2       RC=STRING$TO$BINARY(NEXT$NS);
01691  2       VALUE$PTR=NEXT$NS;
01692  2       DO WHILE RC > 0;
01693  2          RC=RC-1;
01694  3          IF MON2(20,.TFCB) <> 0 THEN DO;
01695  3             VALUE$PTR=NEXT$NS;
01696  4             RETURN;
01697  4          FI
01698  3          PUT$CHA$CH$PTR=APPEND$CH(TBUFF,ST$ENTRY(ST$VALUE$INDEX));
01699  3          PUT$CHA$CURRENT$BLOCK$PTR=PUT$CHA$CH$PTR AND PTR$MASK;
01700  3          FIRST$PUTA$FLAG=FALSE;
01701  3          FOR TBUFF$INDEX=1 TO 127;
01702  3             CALL PUT$CHA(0,TBUFF(TBUFF$INDEX));
01703  4          ENDFOR
01704  3       ENDDO
01705  2       ENDPROCEDURE PF$RF;
01706  1
01707  1    PF$RD: PROCEDURE;
01708  2       /* 'READ DIALECT'  $(RD) */
01709  2       /* EACH RD WILL GO TO THE NEXT META CHARACTER -  */
01710  2       /* OK IF NO INTERVENING MF OR OF. */
01711  2       CON$IN=FALSE;
01712  2       TBUFF$INDEX=128;
01713  2       CALL PF$RS;
01714  2       ENDPROCEDURE PF$RD;
01715  1
01716  1
```

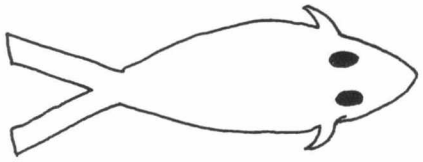


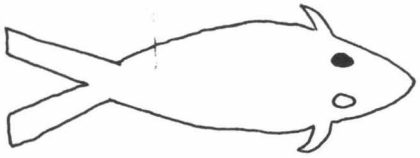
```
01717  1              /*** PROCESS FUNCTION CALL ***/
01718  1
01719  1    PROCESS$END$FUNCTION: PROCEDURE;
01720  2       DECLARE
01721  2          NAME$PTR ADDRESS,
01722  2          NAME BASED NAME$PTR ADDRESS,  /* HOLDS THE FIRST TWO CHARS OF NAME */
01723  2          NAME$CH BASED NAME$PTR BYTE,
01724  2          (BLOCK$PTR,BLOCK BASED BLOCK$PTR) ADDRESS,
01725  2          (CH,FIRST$CH,SECOND$CH) BYTE;
01726  2
01727  2       PF$INDEX: FUNCTION(CHCH) BYTE;
01728  3          /* INDEX OF THE PRIMITIVE FUNCTION WHOSE TWO CHAR NAME IS CHCH */
01729  3          /* NOTE... NAME LIST HAS SECOND CHARACTER FIRST, LIKE ADDRESS */
01730  3          DECLARE
01731  3          CHCH ADDRESS,
01732  3          I BYTE,
01733  3          PF$LIST(40) ADDRESS INITIAL(  /* ORDERED BY FREQUENCY */
01734  3          736EH,   /* NS */
01735  3          7165H,   /* EQ */
01736  3          6E69H,   /* IN */
01737  3          746EH,   /* NT */
01738  3          6567H,   /* GE */
01739  3          7263H,   /* CR */
01740  3          6367H,   /* GC */
01741  3          7373H,   /* SS */
01742  3          6B70H,   /* PK */
01743  3          6461H,   /* AD */
01744  3          7361H,   /* AS */
01745  3          6464H,   /* DD */
01746  3          7573H,   /* SU */
01747  3          6C6DH,   /* ML */
01748  3          7664H,   /* DV */
01749  3          6672H,   /* RF */
01750  3          6677H,   /* WF */
01751  3          666FH,   /* OF */
01752  3          6663H,   /* CF */
01753  3          6E72H,   /* RN */
01754  3          666DH,   /* MF */
01755  3          6664H,   /* DF */
01756  3          6472H,   /* RD */
01757  3          646DH,   /* MD */
01758  3          6B75H,   /* UK */
01759  3          7370H,   /* PS */
01760  3          6F73H,   /* SO */
01761  3          6F72H,   /* RO */
01762  3          6F69H,   /* IO */
01763  3          7865H,   /* EX */
01764  3          626EH,   /* NB */
01765  3          6E74H,   /* TN */
01766  3          6674H,   /* TF */
01767  3          7766H,   /* FW */
01768  3          6E76H,   /* VN */
01769  3          6C63H,   /* CL */
01770  3          7372H,   /* RS */
01771  3          7A7AH,   /* ZZ */
01772  3          7366H,   /* FS */
01773  3          7473H); /* ST */
01774  3
01775  3          FOR I=0 TO LAST(PF$LIST);
01776  3             IF CHCH = PF$LIST(I) THEN DO; RETURN I; FI
01777  4          ENDFOR;
01778  3          RETURN LENGTH(PF$LIST);  /* NO MATCH */
01779  3          ENDFUNCTION PF$INDEX;
01780  2
01781  2       NS$STACK(NS$STACK$INDEX+1)=NULL$STRING$PTR OR NS$NULL$BEGIN$END;
01782  2       DO WHILE NOT NS$FUN$MARKER;  /* FIND THE BEGINNING OF THE CURRENT FUNCTION*/
01783  2          NS$STACK$INDEX=NS$STACK$INDEX-1;
01784  3       ENDDO;
01785  2
01786  2
```

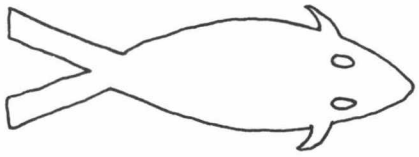


```
01787  2        IF NS$LIMIT$MARKER THEN DO;  /* NO 'BEGIN' FOR THIS 'END' */
01788  2           GO TO START;
01789  3        END; ELSE DO;  /* DO THE FUNCTION WHOSE NAME IS AT NS$STACK$INDEX */
01790  2           NS$SAVE$INDEX=NS$STACK$INDEX;
01791  3           NS$ACTIVE$FLAG=NS$STACK(NS$STACK$INDEX) AND NS$ACTIVE$BIT;
01792  3           IF CON$BREAK THEN DO; CALL PROCESS$CON$BREAK; FI
01793  3           IF TRACE$FLAG THEN DO; CALL TRACE$PRINT; FI
01794  3           /* THE PRIMITIVE FUNCTION NAME IS TAKEN FROM THE FIRST TWO BYTES */
01795  3           /* OF THE FIRST ARGUMENT, REGARDLESS OF THE CONTENTS */
01796  3           NAME$PTR=NS$STACK(NS$STACK$INDEX) AND PTR$MASK;
01797  3           VALUE$PTR=NULL$STRING$PTR;
01798  3           /* TEST FOR TWO CHAR FUNCTION NAME AND FIRST CHAR A-Z */
01799  3           FIRST$CH=NAME$CH(2); SECOND$CH=NAME$CH(3);
01800  3           IF SECOND$CH = NIL OR NAME$CH(4) <> NIL OR
01801  3              FIRST$CH < LIT$A OR FIRST$CH > LIT$Z THEN DO;  /* NOT A PF */
01802  3              CALL PF$CL(-1);
01803  4           END; ELSE DO;
01804  3              SELECT PF$INDEX(NAME(1)) OF
01805  4                 CALL PF$NS;
01806  5                 CALL PF$EQ;
01807  5                 CALL PF$IN;
01808  5                 CALL PF$NT;
01809  5                 CALL PF$GE;
01810  5                 CALL PF$CR;
01811  5                 CALL PF$GC;
01812  5                 CALL PF$SS;
01813  5                 CALL PF$PK;
01814  5                 CALL PF$AD;
01815  5                 CALL PF$AS;
01816  5                 CALL PF$DD;
01817  5                 CALL PF$SU;
01818  5                 CALL PF$ML;
01819  5                 CALL PF$DV;
01820  5                 CALL PF$RF;
01821  5                 CALL PF$WF;
01822  5                 CALL PF$OF;
01823  5                 CALL PF$CF;
01824  5                 CALL PF$RN;
01825  5                 CALL PF$MF;
01826  5                 CALL PF$DF;
01827  5                 CALL PF$RD;
01828  5                 CALL PF$MD;
01829  5                 CALL PF$UK;
01830  5                 CALL PF$PS;
01831  5                 CALL PF$SO;
01832  5                 CALL PF$RO;
01833  5                 CALL PF$IO;
01834  5                 CALL PF$EX;
01835  5                 CALL PF$NB;
01836  5                 CALL TURN$TRACE$ON;
01837  5                 CALL TURN$TRACE$OFF;
01838  5                 CALL PF$FW;
01839  5                 CALL PF$VN;
01840  5                 CALL PF$CL(0);
01841  5                 CALL PF$RS;
01842  5                 CALL PF$ZZ;
01843  5                 CALL PF$FS;
01844  5                 BEGIN PRELOADED=TRUE; EXIT; END;  /* ST */
01845  5                 CALL PF$CL(-1);  /* NO MATCH, ASSUME 'CL' */
01846  5              ENDSELECT
01847  4           FI
01848  3           VALUE$PTR=VALUE$PTR AND PTR$MASK;
01849  3
01850  3           /* RELEASE ARGUMENTS FOR THE FUNCTION WHICH WAS JUST DONE */
01851  3           /* AND LEAVE NULL PTR + NS-END FLAGS IN PLACE OF THESE ARGS */
01852  3           /* BUT DON'T RELEASE AN ARG IF IT IS THE VALUE */
01853  3           NS$STACK$INDEX=NS$SAVE$INDEX;
01854  3           DO WHILE NOT NS$LIMIT$MARKER;
01855  3              IF VALUE$PTR <> (ARG$PTR:=NS$STACK(NS$STACK$INDEX)) THEN DO;
01856  4                 CALL FREE$CHAIN(ARG$PTR);
01857  5              FI
01858  4              NS$STACK(NS$STACK$INDEX)=NULL$STRING$PTR OR NS$NULL$BEGIN$END;
01859  4              NS$STACK$INDEX=NS$STACK$INDEX+1;
01860  4           ENDDO
01861  3
01862  3           /* SEE IF THIS AS$BLOCK CAN BE RELEASED */
01863  3           IF (LOW(AS$CH$PTR) AND NOT$PTR$MASK) = 0 THEN DO;
01864  3              CALL AS$FREE$BLOCK;
01865  4           END; ELSE IF (AS$CH = NIL) AND (HIGH(AS$BLOCK) > 0) THEN DO;
01866  3              CALL AS$FREE$BLOCK;
01867  4           FI
01868  3
```









```
01869  3          /* PROCESS THE VALUE CHAIN */
01870  3          NS$STACK$INDEX=NS$SAVE$INDEX-1;
01871  3          NS$ARG$BREAK=TRUE;
01872  3          IF VALUE$CH(2) = NIL THEN DO;
01873  3             CALL FREE$CHAIN(VALUE$PTR);
01874  4          END; ELSE DO;
01875  3             IF NS$ACTIVE$FLAG>0 THEN DO; /*LINK VALUE CHAIN TO HEAD OF ACTIVE*/
01876  4                BLOCK$PTR=VALUE$PTR;
01877  5                DO WHILE HIGH(BLOCK) > 0;  /* FIND END OF VALUE CHAIN */
01878  5                   BLOCK$PTR=BLOCK;
01879  6                ENDDO
01880  5                BLOCK=AS$BLOCK$PTR;
01881  5                AS$BLOCK$PTR=VALUE$PTR;
01882  5                AS$CH$PTR=AS$BLOCK$PTR+2;
01883  5             END; ELSE DO;  /* ADD VALUE TO END OF CURRENT (NEUTRAL) ARGUMENT */
01884  4                FIRST$GETA$FLAG=TRUE;
01885  5                DO WHILE (CH:=GET$CHA(VALUE$PTR)) <> NIL;
01886  5                   CALL NS$ADD$CH(CH);
01887  6                ENDDO
01888  5                CALL FREE$CHAIN(VALUE$PTR);
01889  5             FI
01890  4          FI
01891  3       FI
01892  2       ENDPROCEDURE PROCESS$END$FUNCTION;
01893  1
01894  1
01895  1             /*** INITIALIZATION AND SCAN LOOP ***/
01896  1
01897  1    DECLARE CH BYTE;
01898  1
01899  1    PRESET$ATIN: PROCEDURE;
01900  2       IF NOT ST$FIND(.AT$IN) THEN DO; CALL ST$CREATE(.AT$IN); FI
01901  2       CALL  ST$ADD$VALUE(VALUE$PTR:=FS$GET);
01902  2       FIRST$PUTA$FLAG=TRUE;
01903  2       ITEMP=2;
01904  2          DO WHILE ITEMP <= TBUFF(0);
01905  2          CALL PUT$CHA(VALUE$PTR,TBUFF((ITEMP:=ITEMP+1)-1));
01906  3       ENDDO
01907  2       ENDPROCEDURE PRESET$ATIN;
01908  1
01909  1    PRESET$DRIVES: PROCEDURE;
01910  2       DEFAULT$DRIVE,PREVIOUS$DRIVE,CURRENT$DRIVE=MON2(25,0);
01911  2       ENDPROCEDURE PRESET$DRIVES;
01912  1
01913  1    IF DEAD$START THEN DO;
01914  1       DEAD$START=FALSE;
01915  2       TRACE$FLAG=0;
01916  2       AS$BLOCK$PTR=0;
01917  2       FS$FIRST$PTR=0;
01918  2       TEMP$PTR=FBASE;
01919  2       IF TFCB(17) = 5EH THEN DO; /* 'ACT FILE  HH', HH=HEX PAGE NO. BEYOND FS */
01920  2          TEMP$PTR=256*(DOUBLE(SHL(CH$TO$HEX(TFCB(18)),4) OR CH$TO$HEX(TFCB(19))));
01921  3       FI
01922  2       CALL FS$INIT(.MEMORY,TEMP$PTR);
01923  2       CALL ST$INIT;
01924  2       CALL ST$REINIT;
01925  2       NULL$STRING$PTR=FS$GET;
01926  2       FOR NS$STACK$INDEX=1 TO LAST(NS$STACK);
01927  2          NS$STACK(NS$STACK$INDEX)=NULL$STRING$PTR OR NS$NULL$BEGIN$END;
01928  3       ENDFOR
01929  2       CALL PRESET$DRIVES;
01930  2       IF TFCB(1) <> ' ' THEN DO;  /* DISK INPUT */
01931  2          CALL PRESET$ATIN;
01932  3          IF MON2(15,.TFCB) <> 255 THEN DO;
01933  3             CALL SELECT$DRIVE;
01934  4             CON$IN=FALSE;
01935  4             TFCB(32)=0;
01936  4          FI
01937  3       FI
01938  2    END; ELSE IF PRELOADED THEN DO;
01939  1       PRELOADED=FALSE;
01940  2       CALL PRESET$DRIVES;
01941  2       CALL PRESET$ATIN;
01942  2       CON$IN=TRUE;
01943  2    FI
01944  1
01945  1    CALL AS$NS$INIT;
01946  1
01947  1    DO FOREVER;
01948  1       IF (CH:=AS$NEXT$CH(MOVE$PTR)) = LEFT$DELIM THEN DO;
01949  2          CALL PROCESS$QUOTED$STRING;
01950  3       END; ELSE IF CH = ARG$DELIM THEN DO;
01951  2          CALL START$NEXT$ARGUMENT(NS$ARG$FLAG);
01952  3       END; ELSE IF CH = FUN$CH THEN DO;
01953  2          CALL PROCESS$BEGIN$FUNCTION;
01954  3       END; ELSE IF CH = RIGHT$DELIM THEN DO;
01955  2          CALL PROCESS$END$FUNCTION;
01956  3       END; ELSE DO;
01957  2          CALL NS$ADD$CH(CH);
01958  3       FI
01959  2    ENDDO
01960  1    EOF
NO PROGRAM ERRORS
```

SAM 76 – Additional comments

Dear Jim:

I just got around to reading your issue 10, volume 2, and was pleased to see the article by Frits Van Der Wateren which describes an implementation of Strachey's GPM for the 6800.

GPM - SAM76

The first purpose of this letter is to point out to the relatively uninitiated that the SAM76 language, described in your first issue of volume 3 has its roots in GPM, and that the structure and philosophy are virtually identical between the two languages.

Mimicking

The main difference between the two languages is the scope of available resident functions, as well as certain features of SAM76 which enable precise user definition in the SAM76 language of functions which might also be defined in terms of assembly language instructions - in other words "exact mimicking capability".

This feature is important to permit portability or transferability of user programs or, as I prefer to call them, scripts. Should some user script make use of a function not resident in some one else's SAM76 interpreter, it is relatively simple to create a user defined function identical in its functioning to the resident function originally used.

SAM76 - 6800

A version of the SAM76 language coded for the 6800 is in process of development by the "notorious" Wayne Loufbrow youngster; it is expected that this version will be functionally identical to the 8080/Z80 implementation, thus permitting absolute transferability of user scripts.

Addendum to SAM76 description

The second purpose of this letter is to respond to some queries from users as to the problem of formatting user script for appearance and legibility, and yet not cause these formatting characters to become part of the evaluation process.

Because of the celerity with which you published the description of the SAM76 language, I was not able to make the appropriate addition which responds to this problem; this addition is described below, as it will appear in the forthcoming "SAM76 Language Handbook".

The table under the heading "Quoting or Protection" on page 20 should be ammended by the addition of the "IC" warning character as illustrated by the excerpt shown below.

AS	,	77	Argument separator
QC	@	100	Quote single character
IC	`	140	Ignore single character

The following specification for this warning character should be concatenated, with scissors and glue, at the end of the SAM76 language description just ahead of the birds on page 22 of your magazine.

SIERRA DIGITAL'S PDP8 X8 CROSS ASSEMBLER SERIES

News Release *Received: 77 Dec 30*

Sierra Digital Systems has announced the addition of four new microprocessor cross-assemblers to its X8 cross-assembler series for the Digital Equipment Corporation PDP8 mincomputer. The X8 series cross-assemblers now cover the Z80, 1802, SC/MP and 8048 microprocessors in addition to the previous 6502, 6800, 8080, F8 and 2650 version.

By using an X8 series cross-assembler, assembly language programs are converted into object code which may be loaded into a microprocessor system, or put into ROM of PROM memory. X8 cross-assemblers feature a Universal Assembler Format of common assembler directives and techniques. This standardization of features combined with a quantity discount schedule makes the series especially attractive to users of more than one microprocessor type. New special package offers at very substantial discounts are available to Educational Institutions for microprocessor course development.

Sierra Digital's X8 cross-assemblers run in 8K words of memory under the OS/8 operating system. The X8 assemblers are written in PDP8 assembly language to provide very fast operation and minimum memory requirements. Pseudo-ops and runtime options provide for conditional assembly and extensive listing control. Generated object code may be output in the microprocessor's standard loader format, or BNPF for ROM generation.

Each cross-assembler is priced at $400 and distributed in PDP8 binary format on paper tape, Dectape, or Dec floppy diskette. Source files are also available for an additional $250. Sierra Digital Systems is located at 13905 Rancheros Drive, Reno, Nevada 89511, (702) 329-9548.

Ignoring next character

Oftentimes the user wishes to arrange his script for convenient legibility using "nulin", "tabs" or other characters; this is strictly for formatting purposes. Under normal circumstances these characters will be part of the expressions, and be returned in the value string unless deleted in the scanner.

One way of achieving this is to place these characters in unused arguments of primitives such as the third argument of an "os" expression. This could be a problem if at some later time the additional unused arguments acquire a purpose.

Under initialized conditions the "`" character is used to signify that the immediately next ensuing character is to be deleted in the scanner.

Final erratum

Another bright youngster, Karl Nicholas, pointed out to me that expression "k" under < skim > on page 23 was unnecessarily complicated, and suggested the following simplification which after being checked out showed the correctness of his observation:

```
< k >
%pt,,[1]/%i/![1]/
```

Closing comments

Plaudits and bouquets to your staff who did a great job of laying out in an esthetic and artistic manner the rather complicated material you received from me.

I hope that the growing body of users of the SAM76 language will contribute algorithms and scripts to your pages, and in that manner provide an escape from the "basic" syndrome.

Cordially,

Ancelme Roichel

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 24 | April 1978 | Volume 3, Issue 4

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

List Processing Unravelled: Pointers, Chains, Rings and Trees *J. M. Terry* 4

Exit This Reviewer *Jef Raskin* 19
DDJ's Reviewer puts his mouth where his money is

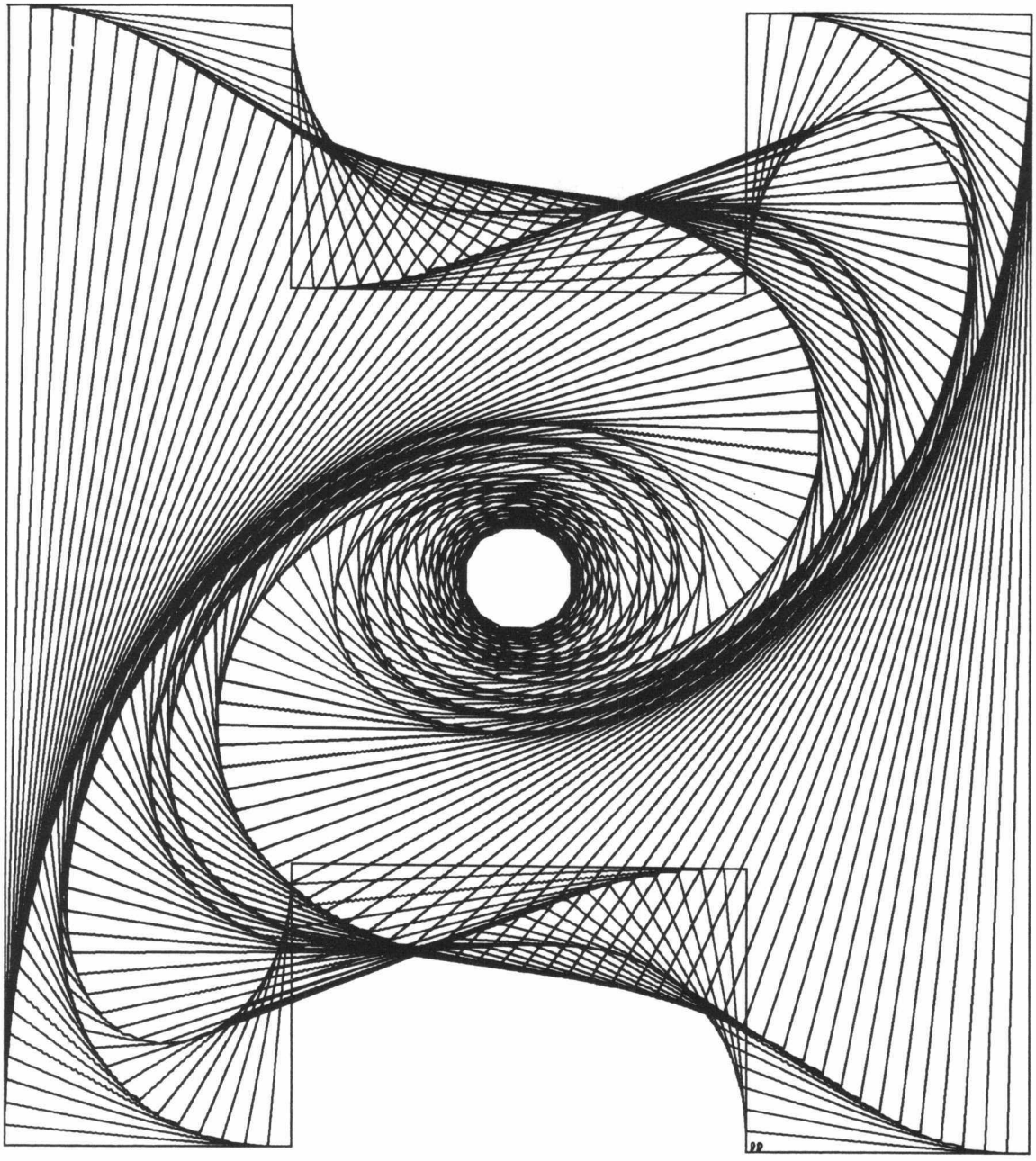
SOFTWARE

The Oznaki "Life" *Harvey A. Cohen* 10
An interactive version for the Poly 88

A 1K Debugger for the Z-80 *Les Hancock* 12

A Line-Oriented PILOT Editor *David J. Morgan* 22

ACT: An 8080 Macroprocessor *Alex Cecil* 27
Assembly listing of PL/M version published in previous DDJ



© David Dameron 1979

UNIX FOR THE 8080

Dear Jim, *Received: 78 Feb 17*

I've been bitten by the bug. The computer bug, that is. Two years ago I literally stumbled into a computer for the first time. And–BAAM!! Now I own one. And it's continually expanding, taking big bytes out of my wallet.

I've always wanted a computer that would do what the 11/70's would do: control everything, and be easy and fun to use. Until recently I couldn't have begun to do that. But after a summer with an 11/70 and Bell Lab's Unix, I might just be able to.

Exactly what I want my IMSAI to do is tough to say, but here goes: EVERYTHING. More precisely, I want it to support several terminals on a time sharing basis, and time share with the same benefits Unix provides. The computer should be able to perform processes without my supervision, and poll many inputs and take actions on its own when required. I plan to someday have it running my house, from the toaster to the alarm clock to the security devices. Sounds incredible, but when completed, it should be fantastic.

Anyone who has worked on Unix knows sort of what I have to do. But for those who don't, a short explanation. Everything the computer has to do will be kept on floppy disk and accessible by the computer. The two terminals (soon three) will interrupt the computer for input, and when a line is complete, it will be put into the process table. When the computer comes to this command, it will load it from the disk, and enter it again into the process table to be executed in due time. (The terminal is then free to be used again, and can have more than one process running under its name at any one time.) Each process in the table will be given equal amounts of the computer's time, say 1/10 second apiece. When a process is complete, it will jump to a routine and have its name removed from the table. In between processes, the computer will poll predefined input ports to see if anything has come up and needs to be dealt with, like a burglar, or a process that has been 'sleeping' for a while and needs to be awakened. Each process would have its own stack and storage area for its registers when another process is being executed. Also included in the shell (a Unix word meaning the part of the computer program that oversees all of these actions, and regulates time) would be such luxuries as piping, terminal to terminal conversation, and the old late night weapons, enable and disable (terminals); However, it is important to remember that the program will not have any assemblers, editors, or languages in it. Everything will be kept on disk and only loaded when needed. This allows for great flexibility, as the control for any piece of hardware can be loaded on the disk, and not take up valuable RAM.

This version of Unix will give my micro (and yours too!) part of the power that minis and macros have, and make it much more useful around the house. Aside from the questions I'm about to ask, the only major problem will be the organization. It will be very easy to get tied up in jumps and loops and calls. It will have to be very efficient on both RAM and operating time (its going to run slow anyway.) If I take my time about it and think it through *completely* before I begin to write, it shouldn't be very difficult.

There are some problems I can already forsee;

1) The 8080 has eight interrupts (or restarts) at its disposal. I would like more. Anybody know how?
2) In order to conserve RAM, each program on disk will have to be entirely relocatable. The best method I have thought of yet is to insert before each addressing command one of the unused hex bytes. While loading, when the computer comes across this byte, it will adjust the following address by the amount the new SA is apart from the base SA, and leave out the flagging command when it loads it into RAM. If someone has a better idea, let me know.
3) As the operating system stands now, every process will be given equal time. Is it worthwhile to set up a priority system so that important software can run faster than other software?
4) Should I use the Z80 instead of the 8080? The Z80 would clear up some relative addressing problems, but leaves no unused commands to use as a flag in the relocate operation. Anybody who can convince me in either direction wins my thanks.
5) What should I call it? Unix for the 8080? Trivial question.
6) And finally, is anybody interested in such a program? Anybody want to make a few suggestions?

In case you're interested in what kind of system I have, it consists of an IMSAI 8080, 32K RAM, IMSAI PIC board w/ programmable interrupts, IMSAI MIO board, two Polymorphic video displays w/ keyboards, a Peripheral Vision Floppy Disk & interface, a high speed dot matrix printer, Cromemco's analog I/O w/ two joysticks, an optical papertape reader, and lots of software (tank game, Star Wars game, HAM interface, theatrical lighting control, etc. all of my own writing).

I plan to spend this next summer writing Unix for the 8080 and associated programs. I'd like to hear from anybody on this program, especially anyone from the Delaware area (where I live), or the Northwest Connecticut area where I go to school.

Keep up the good work at *DDJ*,

Pierre duPont
Salisbury School
Salisbury, CT 06068

We're *interested in such a program and we'll be happy to publish suggestions and ideas pertinent to your project... and maybe you'll send* us *the results. Happy coding!* –*TRW*

STAR WHA-A-A-T?

Gentlemen: *Received: 78 March 9*

Long, long ago in a galaxy far, far away, there was born a new product line of computers. The company was an established retail outlet starting amidst strong competition, but this line had an edge over most others: it was prebuilt and tested.

This company has had its complement of renegades. Unfortunately the renegades occupied the big offices with the long titles on the doors. These outcasts secreted away documents on a space satellite. What documents you ask? All documents on Hardware and Software. Yes friends, it is indeed a crime. And what's worse – the satellite came crashing down to Earth! Already the renegades are being hunted down by the DETH STAR. (*D*ocumentation in *E*ntirety, *T*ruth, *H*onesty), controlled by General Competishun. This Texas based, handy dandy radio hut had better get off its *AS*cii before Darth Vector points his blinking cursor and erases you.

Signed,
Jeffrey Greek
O. B. 1
(Overwrought and befuddled)

4730 Cloister Ave. NW
Canton, Ohio 44709

LIST PROCESSING UNRAVELED: POINTERS, CHAINS, RINGS & TREES

BY J. M. TERRY
P. O. Box 3945
Lubbock, TX 79412

There is more to learning programming than the memorization of instruction sets or statement types. A distinction is often made between two different sorts of skill acquisition, "horizontal" and "vertical". For instance, learning BASIC as a first language is "vertical". After that, it takes a "horizontal" effort for the programmer to learn how expressions are written in FORTRAN. It takes a different kind of effort to learn the use of the ALLOCATE and FREE statements in PL/1 because there are no equivalent statements in BASIC. When we acquire new concepts, a "vertical" effort is needed. Most of the following concepts are beyond BASIC. The steps aren't difficult, but are definitely "vertical".

Sooner or later, we are going to need powerful, elegant ways to handle records. Those of us interested in robotics, languages, and various other Artifical Intelligence (AI) applications will want to connect information in complicated ways. Systems buffs will want to manipulate individual programs (tasks) as objects. Business software certainly can make use of efficient record handling.

Pointers

From assembler language on up, data items are associated with names. For instance, when a variable is assigned a value, that value is put in the location with that name. Thus,

LET S = 12

in BASIC means that the place named "S" is to contain "12". When the program is loaded, the variable's *name* is equivalent to its *address*. If, in this BASIC, variables are eight bytes long, then the eight bytes representing "S" are "pointed to" by some address. Individual instructions may use different conventions for arriving at it, but all instructions referring to "S" must reference this address. Notice that two bytes is always the maximum needed to point anywhere inside the addressing range of a 64K MPU. So, "S" is the same as some two-byte address. If "SOCIAL_SECURITY" had been used instead of "S", its equivalent address would still be two bytes.

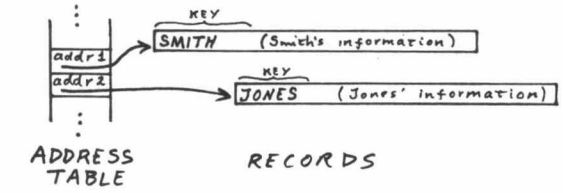
Consider this array:

DIM ALPHA(5)

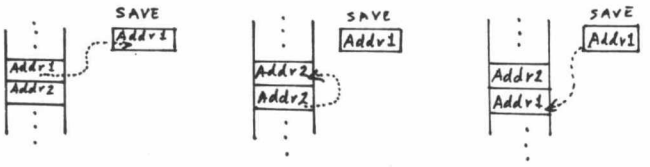
This defines five strings of eight bytes each in our BASIC. We want to sort them using a *bubble-sort*. If the first of a pair of values is larger than the second, we save the first, move the second to the first's position, and put the saved value in the second position, thus swapping the values. In this way, we pass through all four pairs repeatedly until none are swapped. The smaller values will "bubble" up to their proper places. Notice that every time we swap a pair we must move eight bytes three times. Now, suppose that we had the *addresses* of the eight-byte strings in another array. We might construct an assembler program which referenced the strings using this address array. We could perform the sort on the address array. Each 'swap' would involve just two bytes for each of the three moves, instead of eight bytes for each move (Fig. 1).

There are many other powerful uses of pointers, but these will do for what follows.

Fig. 1: Sorting Using Record Addresses



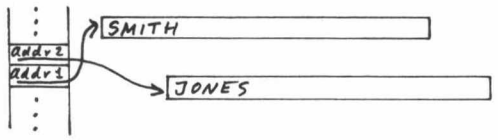
A. Jones' record should come before Smith's.



B. Move 'Addr1' to SAVE

C. Move 'Addr2' to previous pointer

D. Restore current pointer from SAVE



E. Pointers to the records are now in the correct order. The records themselves have not been moved.

Memory Management

There are many times when it is desirable to postpone defining the location of data in memory. Suppose you want to write a general routine for updating personnel records, but each job description uses a different format and size, because each involves different details. You have hopes of using some of the memory for interactive programs. If you allow for the largest record all the time, it may mean that certain programs will have to stay on peripherals to be loaded when needed, slowing the system. You would like a better way.

If your operating system has a *memory-manager*, then all you have to do is wait until the storage to hold a record is needed. When it learns the size needed, your program can pass this information to the memory-manager. A *pointer* will be returned indicating where the allocation of memory begins. There is a second, complementary function of the memory-manager: he must be able to *free* memory as well as *allocate* it, or, sooner or later, he will run out.

Memory-managers are used whenever allocation is postponed until the program is actually executing. In addition to its importance in processing of variable-length records, this is desirable or necessary for *recursive* programs, *interactive interpreters*, *loaders*, and *lists*. A *list*, by the way, is a data set whose size can change during program execution.

Chains

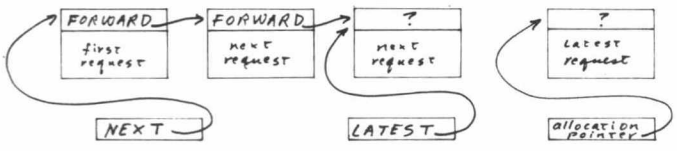
When we interact with our systems through terminals, we say we are processing in *real-time*. This is contrasted with *batch* processing, in which a procedure is first defined completely, then executed. Systems with which people interact have fewer uses for which memory can be defined when they are loaded than batch programs do. An interactive program may not learn what memory it needs until some point in the interaction with the user.

In many systems it is better to process requests on a "first come, first served" basis. When a terminal interrupts an ongoing task, its request is not processed immediately. Instead, a very short routine is invoked which saves the request. The processor then returns to the interrupted task. When this task is finished, the next request is begun. These requests are being stored in a *First-In-First-Out* (FIFO) queue. The "enqueuing" algorithm allocates enough space for the request and its parameters (Fig. 2a). The pointer to the new allocation is stored in the FORWARD field of the element pointed to by LATEST. This stored pointer *links* the allocation. Then the new allocation's pointer is stored in LATEST to make finding the end easier.

When the current task is finished, it is time to *de-queue* a request. CURRENT points to the information for the request just finished. The FORWARD field of the element that CURRENT points to indicates the information for the next request. We are through with the current element. We move its FORWARD field to SAVE. The contents of CURRENT are passed to the memory-manager in a *free-memory* call. SAVE is moved to CURRENT. This pointer is passed to the request handler, which begins the task specified in the data element pointed to (Fig. 2b).

If we didn't compare CURRENT and LATEST, we wouldn't know if FORWARD contained a pointer or meaningless garbage. Besides, how do we know if the queue is empty? A *null* value for pointers simplifies everything. This could be anything, but "0000" (hexadecimal) or some address in ROM might do for an MPU system. It is easy to remember that zeroes are null, and since low memory often is reserved for interrupts, it isn't likely that the ability to allocate from that address will ever be missed. Whenever an element is added to the end of a chain, its FORWARD pointer should be set to NULL. The handling routines can easily tell when a list is empty or when an end is reached by testing for NULL.

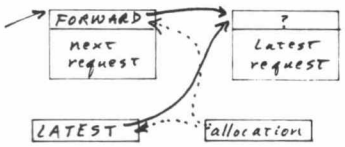
Fig. 2: FIFO Queue Manager



chained requests

newly allocated and initialized request element

A. Enqueue—Immediate request handling—create new request element and store request in it.

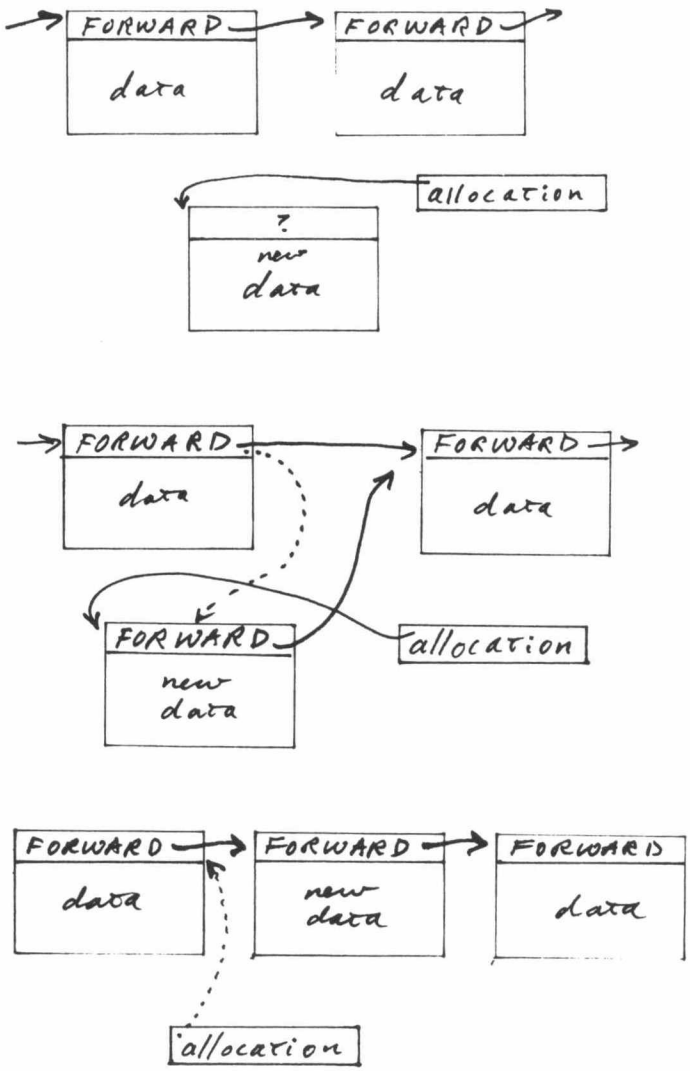


B. De-queue—Move pointer to new element into FORWARD field of latest element, then to LATEST pointer

The list described in Figs. 2a and 2b is called a *one-way* linked list, or a *one-way chain*. The *links* are the FORWARD fields. We used LAST to point to the latest entry, but that is not strictly necessary. As long as we know where the chain starts, we can advance, link by link, until a NULL indicates the end.

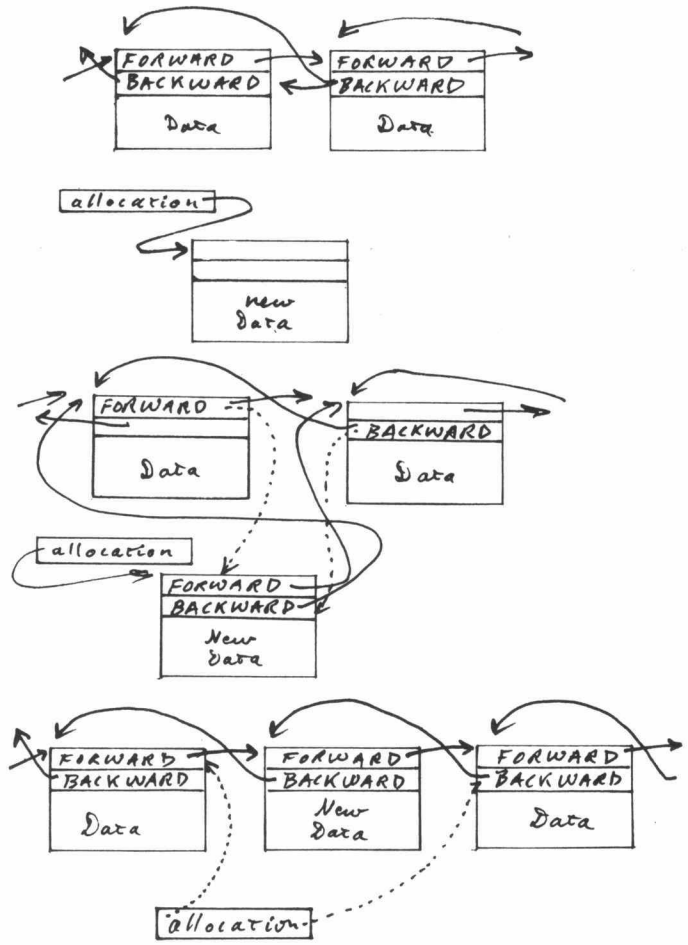
Often we will want to *insert* or *delete* in the middle of a chain. For instance, we may be using a chain to hold data we wish to keep in alphabetical order. To *search*, we follow the chain out link by link, comparing the key field of each element with the name to be added (Fig. 3a). The pointer returned with the allocation for the new element must be stored in the element preceding the inserted element. We don't know which element that will be until we find a key field with a higher sort value (that is, a key later in the alphabet) than the name to be inserted. Therefore, we must save a pointer to the last element every time we advance another element. If each element contains a BACKWARD pointer, search and insertion is simplified (Fig. 3b). This structure is called a *two-way* linked list, or *two-way chain*, naturally enough.

Fig. 3a: Insertions



A. Insertions in a one-way chain

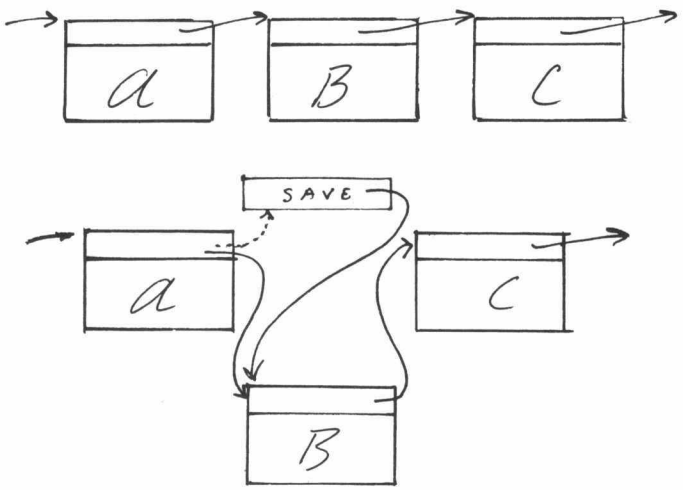
Fig. 3b: Insertions

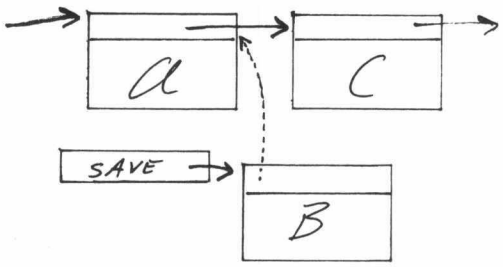


B. Insertions in a two-way chain

Deletions are similar to insertions, only backwards. Pointers are altered to point around the deleted element. A pointer to the deletion must be passed to the memory-manager in order to free it; otherwise that allocation can't be used again (Fig. 4a, 4b).

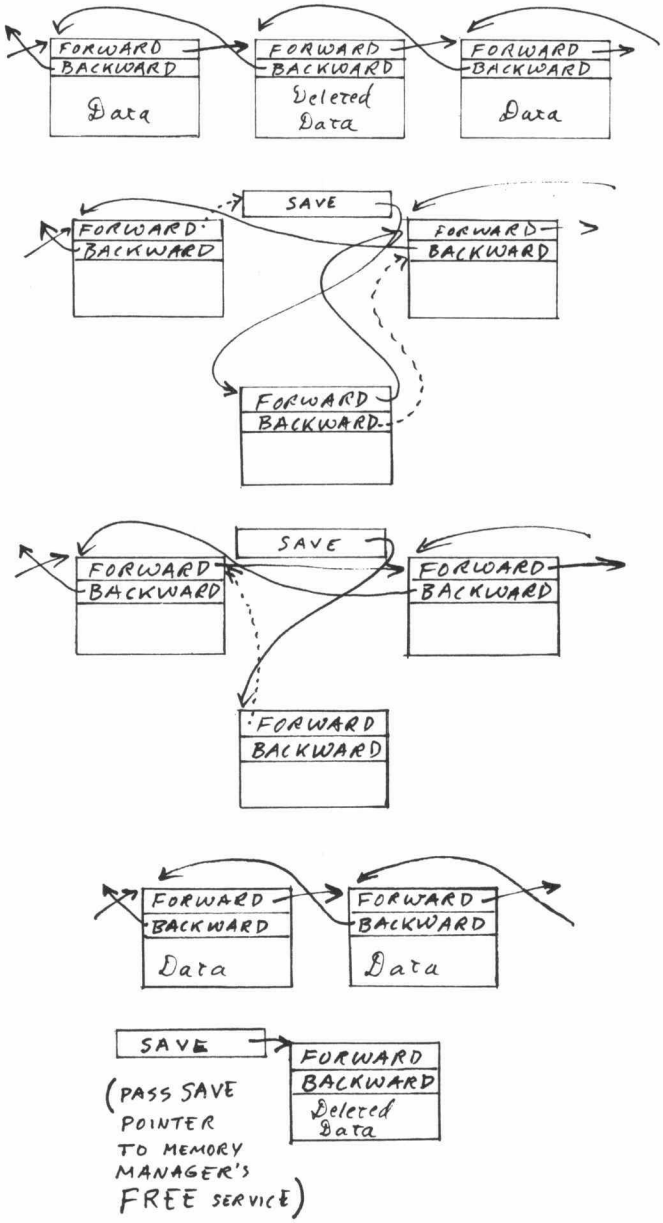
Fig. 4a





Deletions in a one-way chain

Fig 4b: Deletions in a two-way chain

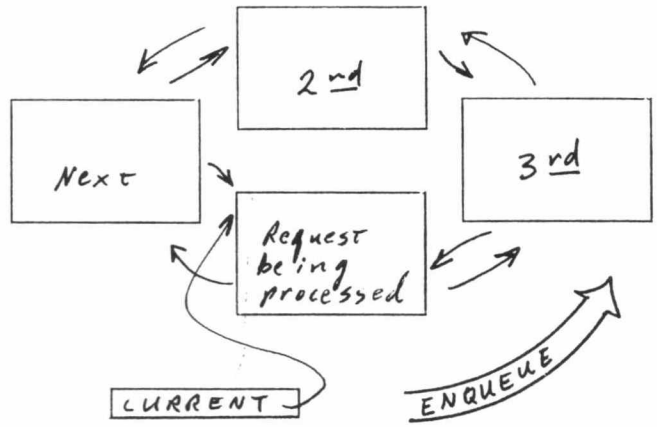


Rings

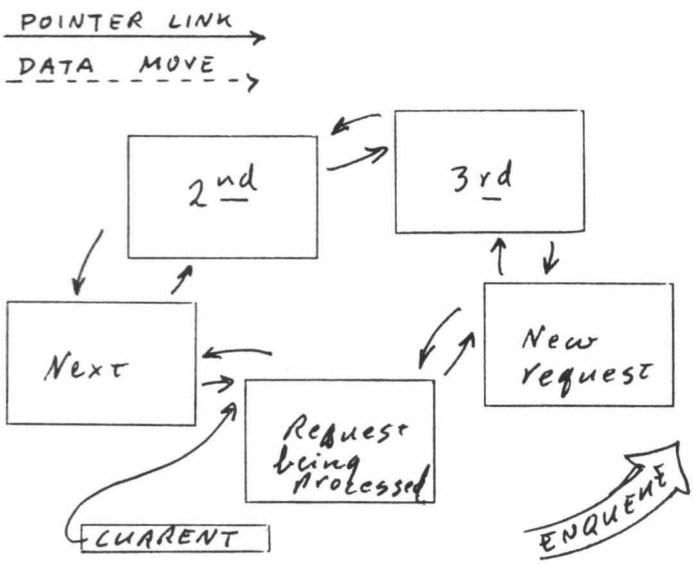
There are times it is more convenient not to have an end in a chain. If the (null) last element of a chain is linked to the first, a *circular* list, or *ring*, results. Suppose you are designing an operating system to support up to sixteen terminals for computer-assisted instruction (CAI) applications. You need information on each user currently logged on the system. You want your monitor to poll each terminal in turn, but only if it is logged on. Simply let the user data reside in a circular list. Each LOGON procedure will insert an element describing terminal and user, and each LOGOFF will delete the proper element. When a user surrenders MPU control, polling will proceed in one direction. The CURRENT pointer is the only pointer needed by the polling routine. It always points to the data for the next terminal to be handled.

Two-way circular lists are especially nice in FIFO queue management. Instead of chaining device requests and holding two pointers (or searching out to the chain end to make additions to a chain), we may build a ring from a fixed CURRENT element in one direction (Fig. 5). When the CURRENT element is deleted, the next element appears from the other direction.

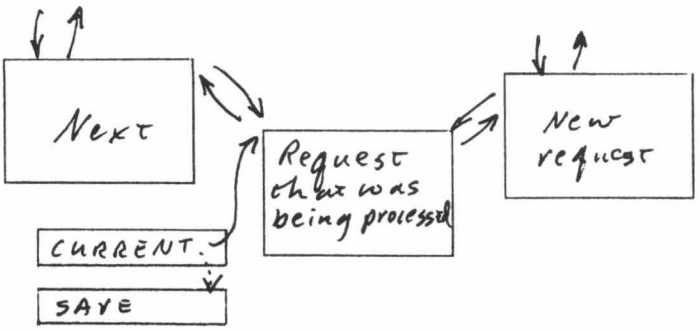
Fig. 5: 2-way FIFO Ring



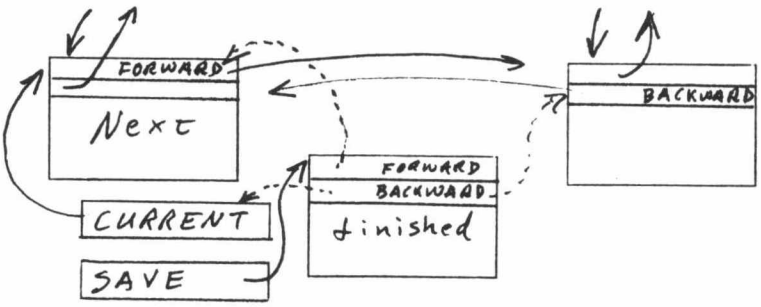
A. Ring before interrupt



B. New request enqueued



C. Current request processing is finished

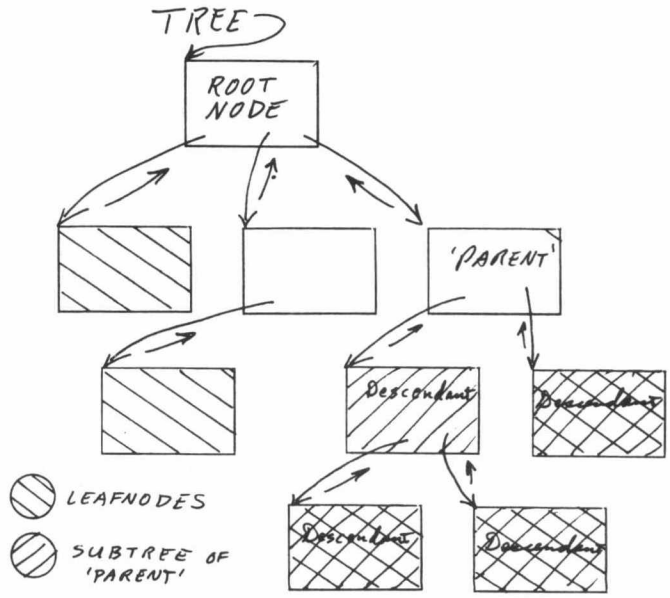


D. Next request is selected; request just finished is then freed

Trees

If an element contains more than one link (not counting backward pointers), a tree structure results. The tree's data elements are called *nodes*. The starting node is called the *root-node* (see Fig. 6). In moving within a tree, down means moving away from the root-node towards peripheral nodes, and up means moving towards the root-node. This is, perhaps unfortunately, exactly the opposite of nature's convention.

Fig. 6



However, as in real trees, the most peripheral nodes are called *leaves* or *leaf nodes*. A "family tree", such as you might have in an old family bible, has this structure. The leaves are the latest generation. Not surprisingly, the higher nodes of a tree are called *parents* and are said to have *children* or *descendant* nodes. A *subtree* refers to some parent node and its descendants.

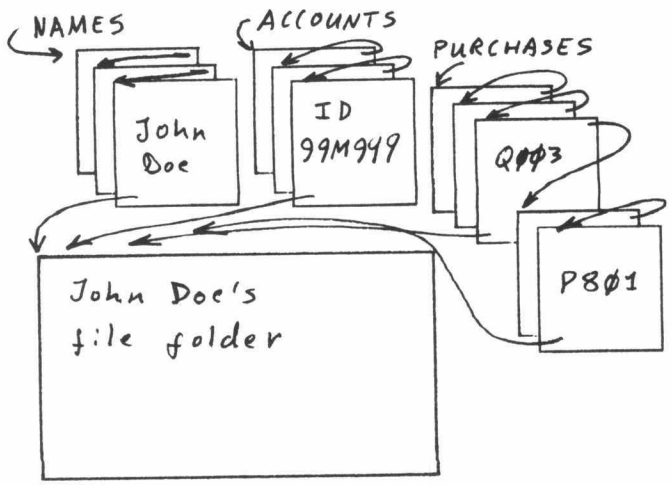
Moving down a tree involves making decisions; moving up simply means remembering where you came from. CAI routines often use trees. A node representing a question and its answers is linked to other nodes. The choosing of one of the answers causes a particular descendant question to be referenced. Another subtree from this node is selected in the same way. This continues until the user reaches a leafnode. Different students might be asked entirely different questions, except for the first root-node question, of course.

Storage and retrieval using a tree can be very fast, especially for large data bases. This is because fewer tests are required in searching. If every node has no more than two descendants, a single test may narrow the search to half of the remaining nodes. Trees are useful for managing symbol tables, efficient use of diskette storage, and maintaining ordered data which is constantly being changed. The many uses of trees really deserve separate elaboration.

Other Stuff

It is often better for list elements to contain a pointer to data instead of the data itself. If the data to be linked is already in memory, just allocate enough space for linkage pointers and a *data-pointer*, insert this element in the list, and store the address of the data in the data-pointer. This has the advantage of allowing several lists to reference the same data, thus saving the duplication of data and conserving memory. Carrying this idea one step further, each link may contain a *key*. Several different sorts of key can be referenced in several lists. Suppose an art store wishes to be able to reference a customer file by customer name and account number, and also wants to be able to keep track of which customer had purchased any particular work of art. Lists for names, ID's, and sold artworks could be maintained in order to reference pointers to information on the customer they match. Such a referencing scheme for JOHN DOE, whose ID is 99M999, and who is the owner of two works catalogued P801 and Q003, is shown in Fig. 7. The lists from which the links came could be chains, trees, rings, or compounds of these.

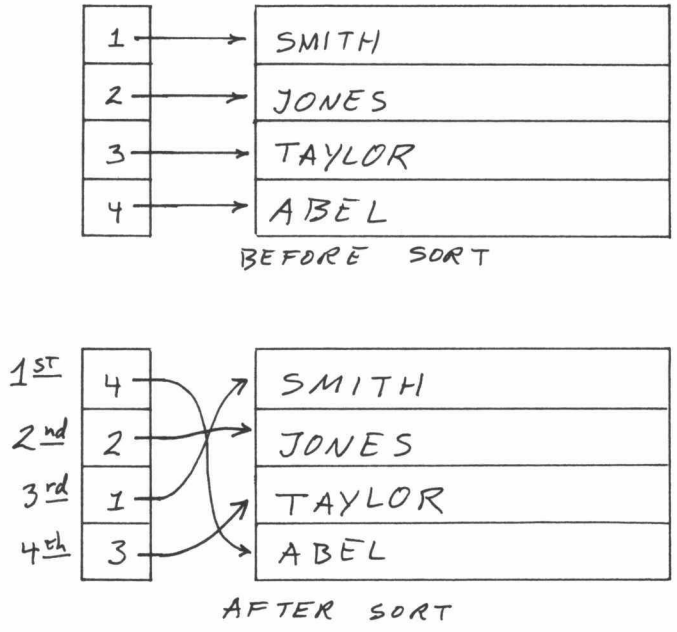
Fig. 7



Multiple References to the Same Data Using Several Lists

It may not be easy to apply all of these ideas immediately in your own systems. However, even if you are just learning BASIC, take heart! There are a few tricks that will carry over. Remember the bubble-sort? A one-dimensional array (a vector) can be used to store index values to another array which contains the records to be sorted. Swap the indices in the first array instead of moving the data. After the sort, the first item will be in some position in the data array, and that position will be indicated by the first element of the first, index, array. Similarly, the second element in the index array will indicate the position of the second item in the sort sequence, and so on (Fig. 8).

Fig. 8



Sorting Using Subscripts

Now, with all of these new tools, you should be just about ready to start designing that great new system that's been in the back of your mind all these months.

Happy linking!

MAILING LIST PACKAGE FOR NORTHSTAR

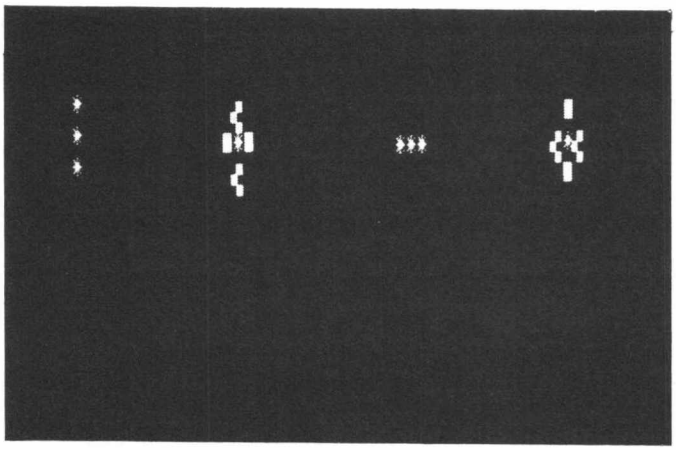
News Release ***Received: 78 Feb 22***

The Comprehensive Mailing List Program Package #ML-1NS, is a modular program set which enables the user to start and effectively maintain one or more mailing lists. Operations include: Add, Delete, Search, Sort, Auto-Sort, and Sequential Printout. Features include: user-selectable defaults for ease of entry, user-selectable number of labels across page for different printers and label sheets, and user-selectable 3 or 4 line address for each independent entry. The program set is written for convenience and ease of use. Available with complete documentation and North Star diskette (SWTPC version available soon) for only $25 PPD. Documentation package only is $4.50 PPD, fully refundable with order for diskette. Order from: Williams Radio and TV, Inc., Computer Division, 2062 Liberty Street, P. O. Box 3314, Jacksonville, FL 32206; (904) 358-3707.

THE OZNAKI "LIFE"

An interactive version of LIFE for the POLY-88 VDM
INTRODUCING THE NAKI CONCEPT FOR TV GRAPHICS

BY HARVEY A. COHEN
OZNAKI Project
La Trobe University
Bundoora 3083
Australia



Here is a version of Conway's classic (i.e. 1969) game of LIFE that has fairly advanced interactive facilities – one can play God with aplomb – yet the program is less than 200 decimal bytes long, and when running uses only a few bytes for a stack and the 1K of memory in a Polymorphic Systems VDM. (I eliminated the need for a 1K next generation store by "marking" the screen itself – indicating which live cells will die in the next generation, and which dead cells will become alive – without interfering with the neighbour count algorithm.)

The key idea in running the LIFE is that there is a little screen creature – a NAKI – which can be commanded to move about seeding or unseeding the screen. A more sophisticated NAKI might drag a picture (e.g. a rocket ship) about the screen, but that's not called for here.

When this game is started the player is presented with a clear TV screen. On pressing the E key, a NAKI appears on the upper-left hand corner of the screen. The NAKI can be driven about (and off) the screen using the commands:

N	NORTH!
S	SOUTH!
E	EAST!
W	WEST!

The NAKI will populate ("seed") the screen cell it currently occupies in obedience to the commands

L	LIFE	Deposit an asterisk
K	KILL	Blank cell

Just what's underneath the NAKI is not visible until the NAKI moves on, or LIFE is started. The whole screen can be cleared with the C command. When you are satisfied with the way you have populated the world, you may grant it LIFE – an autonomous existence determined by savage laws – by keying in the command G.

G	GO

If you want to "single-step" – instead of G use the command H:

H	GO ONE GENERATION THEN HALT

The pattern is steady for about two seconds, then the cells are marked for change. A dead cell in which life is to appear in the next generation is marked by a solid blob. While a living cell that is going to die is marked with a three-pronged graphic character. (The display of the marked generation could be eliminated by removing the corresponding CALL DELAY in the program.) About 2 seconds later, the next generation appears. Here is what happens to a set of traffic lights.

Even though the Birth-Death agony sequence was started on the G command, the display will halt on the next generation by making ANY keystroke; once the display halts, this keystroke is obeyed if it is a valid NAKI or LIFE command. Elsewise, once the display has halted, the NAKI is sitting just off the screen, and may be brought back onto the screen to reseed or kill at your whim.

For those who haven't met LIFE before, the following details may be useful. Each of the 1024 cells (character positions) on the screen has 8 neighbours. Cells with 2 live neighbours are stable, whilst if a cell has exactly 3 live neighbours it will hold life in the next generation. Elsewise the next generation in that cell is dead. These are the usual rules for LIFE, as per Marvin Gardner's articles in the *Scientific American* in October 1970, and February 1971. Gardner's articles include pictures of the evolution of several interesting LIFE "worlds", including gliders and glider guns, pulsars and pentadecathlons, and a host of stable patterns. In this implementation the screen is large enough to display a glider gun, but the gliders ejected bunch up and destroy each other on the screen boundary. Conway's LIFE is played on an infinite lattice. In this LIFE, off screen neighbours are always dead, and there is a crude wrap-around on the screen sides.

Note that this program will not corrupt any location in memory other than the actual program storage space, less than 10 bytes of stack, and the VDM of course.

The program can be adapted to use the Processor Technology VDM, at the price of a modest increase in length, which is then needed to properly count off-screen neighbours as dead. A non-interactive version of LIFE on the P.T. VDM, written by Marvin Wizenread, was printed in *DDJ* November 1976. Unfortunately, Wizenread's program, if unaltered, will corrupt any memory within 10K or the top of the VDM.

I wrote this version of LIFE to provide a demonstration piece that proves that pretty interesting things could be done in tomorrow's schools using 2 chip microcomputers. The 2 chips of this mythical computer would be a CPU with 1K RAM and 1K PROM like the Intel MCS-80, and a video chip.

```
                OZNAKI  PROJECT  CROSS  ASSEMBLER  OUTPUT
                    INTERACTIVE LIFE ON POLY88
                    LIFE    P88     16-AUG-77 13:53

                        TITLE   INTERACTIVE LIFE ON POLY88
                        SEARCH CROSS
3000                    LOC 3000
                F800    VDM=   F800
                  F8    VDMU= F8
                  AA    ALIVEC=AA
                  7F    DEADC=7F
3000 2100FC     CLEAR:  LXI     H,VDM+400
3003   3EF7             MVI     A,VDMU-1
3005   367F     BLANK:  MVI     M,DEADC
3007     2B             DCX     H
3008     BC             CMP     H
3009 C20530             JNZ     BLANK
300C     E5     READY:  PUSH    H
300D 01C0FF             LXI     B,FFC0  ;NORTH ADDRESS INCREMENT
                                        ;B IS ALSO CURSOR CHARACTER
3010 114000             LXI     D,40    ;SOUTH ADDRESS INCREMENT
3013     4E     READIN: MOV     C,M     ;REAL CHAR STORED IN C
                                        ;CORRUPTS BC
3014     7E             MOV     A,M
3015     70             MOV     M,B
3016 CD200C             CALL    WH0
                  C20           WH0=C20
                                        ;;CHARACTER INPUT
3019     71             MOV     M,C     ;RESTORE CHARACTER TO SCREEN
301A   0EC0             MVI     C,C0    ;RESTORE BC AS NORTH INCR
301C   FE4E             CPI     "N"
301E CC9730             CZ      NORTH
3021   FE53             CPI     "S"
3023 CC9930             CZ      SOUTH
3026   FE45             CPI     "E"
3028 CC9B30             CZ      EAST
302B   FE57             CPI     "W"
302D CC9D30             CZ      WEST
3030   FE4B             CPI     "K"
3032 CC9F30             CZ      KILL
3035   FE4C             CPI     "L"
3037 CCA230             CZ      LIVE
303A   FE43             CPI     "C"
303C CA0030             JZ      CLEAR
303F   FE48             CPI     "H"
3041 CCB130             CZ      ONEGEN
3044   FE47             CPI     "G"
3046 C21330             JNZ     READIN
3049     E1             POP     H       ;SETS VDM=1 IN CURRENT GENERATION POINTER
304A             LIFE:
304A     AF     NO:     XRA     A       ;INITIALISE CELL COUNTER
304B     86             ADD     M
304C     09             DAD     B       ;TO GO NORTH
304D     86             ADD     M
304E     23             INX     H       ;TO GO EAST
304F     86             ADD     M
3050     23             INX     H       ;TO GO EAST
3051     86             ADD     M
3052     19             DAD     D       ;TO GO SOUTH
3053     86             ADD     M
3054     19             DAD     D       ;TO GO SOUTH
3055     86             ADD     M
3056     2B             DCX     H       ;TO GO WEST
3057     86             ADD     M
3058     2B             DCX     H       ;TO GO WEST
3059     86             ADD     M
305A     23             INX     H       ;TO GO EAST
305B     09             DAD     B       ;TO GO NORTH
305C   E607             ANI     7       ;WHATS IN BOTTOM 3 BITS?
305E   FE06             CPI     6       ;WERE THERE 2 LIVE?
3060 C48830             CNZ     MARK    ;DO NOTHING IF 2 NEIGHBOURS
3063     7C             MOV     A,H     ;TEST CODE
3064   E604             ANI     4
3066 CA4A30             JZ      NO      ;IF NOT FINISHED CONTINUE
3069 CDA530             CALL    DELAY
                        ;;BIRTH DEATH AGONY SEQUENCE
306C     2B     C1:     DCX     H
306D     7E             MOV     A,M
306E     B7             ORA     A
306F CCA230             CPE     LIVE
3072 CC49F30            CPO     KILL
3075     7C             MOV     A,H
3076   FEF7             CPI     VDMU-1
3078 C26C30             JNZ     C1
                        ;;DELAY FOR VIEWING OF MARKED LOCATIONS
307B CDA530             CALL    DELAY
                        ;;DISPLAY CAN BE STOPPED AND ALTERED
307E 3A0C0C             LDA     KBUFF
3081     B7             ORA     A       ;IF A CHARACTER HAS BEEN TYPED
3082 CA0C30             JZ      READY   ;GO AND PROCESS IT
3085 C34A30             JMP     LIFE    ;ELSE CONTINUE LIFE
                        ;;ROUTINES THAT MARK CELLS FOR POSSIBLE ALTERATION
3088   FE01     MARK:   CPI     1       ;ARE THERE EXACTLY 3 LIVE NEIGHBOURS?
308A CA9230             JZ      LMARK
308D   3E7F     DMARK:  MVI     A,DEADC
308F     A6             ANA     M
3090     77             MOV     M,A     ;CELL NOW MARKED FOR DEATH
3091     C9             RET
                  80            LIVEMARK=80
3092   3E80     LMARK:  MVI     A,LIVEMARK
3094     B6             ORA     M
3095     77             MOV     M,A     ;CELL NOW MARKED FOR LIFE
3096     C9             RET
                        ;;
                        ;;ROUTINES FOR SEEDING THE SCREEN
                        ;;
3097     09     NORTH:  DAD     B
3098     C9             RET
3099     19     SOUTH:  DAD     D
309A     C9             RET
309B     23     EAST:   INX     H
309C     C9             RET
309D     2B     WEST:   DCX     H
309E     C9             RET
309F   367F     KILL:   MVI     M,DEADC
30A1     C9             RET
30A2   36AA     LIVE:   MVI     M,ALIVEC
30A4     C9             RET
                        ;;DELAY ROUTINE
                        ;;SAVES ALL EXCEPT F FLAGS
30A5     C5     DELAY:  PUSH    B
30A6     47             MOV     B,A
30A7     05             DCR     B
30A8     0B     DELOOP: DCX     B
30A9     B8             CMP     B
30AA     C5             PUSH    B       ;WASTE FURTHER 10 CYCLES
30AB     C1             POP     B       ;ANOTHER 10 CYCLES
30AC C2A830             JNZ     DELOOP
30AF     C1             POP     B
30B0     C9             RET
30B1 210C0C     ONEGEN: LXI     H,KBUFF
30B4     34             INR     M       ;SET VALID DATA FLAG
30B5     23             INX     H
30B6     34             INR     M       ;SET NULL COMMAND "I" IN CHAR BUFFER
30B7     3D             DCR     A       ;PUTS "G" IN A
30B8     C9             RET
                        ;;NOTE ON COMPUTATION OF NUMBER OF LIVE  NEIGHBOURS
                        ;;
                ;;After initialising the accumulator to zero,
                ;;all 8 neighbours are added in turn to A.The contents of accumulator
                ;;are then
                ;;      A=l*ALIVEC + d*DEADC
                ;;      where l = number of live neighbours
                ;;      and   d = number of dead neighbours
                ;;      Note  l + d = 8
                ;;      and ALIVEC = A0, while DEADC = 7F in Hex
                ;;      Hence   A = l*28 + 8*DEADC
                ;;      After anding with 7,
                ;;              A = l*3 (mod 8)
                ;;      so      A = 6 for 2 live neighbours
                ;;              A = 1 for 3 live neighbours
                ;;              A = 3,4,7,2,5,0,0 for 1,4,5,6,7,8,0
                ;;                  live neighbours
                ;;Off VDM squares hold FF and therefore are read as DEAD.

                        END

NO ERRORS DETECTED

PROGRAM BREAK IS 000000
ABSLUTE BREAK IS 030271
CPU TIME USED 00:02.812

6K CORE USED


                OZNAKI  PROJECT  CROSS  ASSEMBLER  OUTPUT
                    INTERACTIVE LIFE ON POLY88
                    LIFE    P88     16-AUG-77 13:53
                    SYMBOL TABLE

                A       7
                ALIVEC  AA
                B       0
                BLANK   3005
                C       1
                C1      306C
                CLEAR   3000
                D       2
                DEADC   7F
                DELAY   30A5
                DELOOP  30A8
                DMARK   308D
                EAST    309B
                H       4
                KBUFF   0C0C
                KILL    309F
                LIFE    304A
                LIVE    30A2
                LIVEMA  80
                LMARK   3092
                M       6
                MARK    3088
                NO      304A
                NORTH   3097
                ONEGEN  30B1
                READIN  3013
                READY   300C
                SOUTH   3099
                VDM     F800
                VDMU    F8
                WEST    309D
                WH0     0C20
```

A 1K DEBUGGER for the Z-80

BY LES HANCOCK
305 Lexington Ave 7-C
New York, NY 10016

The 1K Debugger is really a 0. 95117188K debugger. This version is assembled to start at 074061 octal. That puts it at the top of 16K, up out of the way of most programs. The code must be assembled to end at a page boundary (***/377 octal, **FF hex). The debugger is meant to run on a Digital Group Z-80 system with ZE audio ROM, TVC-64 video interface and TVC-64 monitor.

This debugger provides the following features:

- Display the contents of every Z-80 register as it was at the moment the debugger was invoked.
- Display the current contents of any memory location, using extended addressing, register indirect addressing or indexed addressing.
- Change the contents of any register buffer.
- Change the contents of any memory location in RAM, using the three addressing modes mentioned above.
- Load the Z-80 registers from register buffers and jump to any address, with or without enabling interrupts and with or without setting a breakpoint.
- Load the Z-80 registers from register buffers and jump to the most recently set breakpoint, with or without enabling interrupts and with or without setting a new breakpoint.
- Jump to Op Sys program/dump, with automatic return to the debugger.
- Swap contents of register buffers with Z-80 alternate registers.
- Display any 8-bit value in binary notation.
- Give the two's complement of any 8-bit value.
- Add or subtract two 16-bit values in split octal or hex.
- Display or set mode (hex or octal).
- Clear TV screen.

This is a simple debugger, and using it should be fairly straight-forward even for new programmers. Get an assembly listing of the code to be debugged so you'll know the address of every instruction. Load TVC-64 Op Sys, debugger and target program. Create a new Op Sys option to jump to the debugger, then use execute and breakpoint commands to run your program piecemeal. Whenever it reaches a breakpoint the program will return to the debugger; try the various display commands to see exactly what's been happening in the CPU and in memory. You may want to load new values, then jump back into the target code. You always have access to the system program/dump (via the ? command) to do extensive reprogramming or to display large blocks of contiguous memory.

When you jump to the debugger it resets the stack point, disables interrupts, and inserts the debugger's start address at 001025–001026. (This last step vectors RST 070 back to the debugger.) Then it loads a set of buffers with the contents of the Z-80 registers, removes any breakpoints that may have been set, and waits for input. If you want to re-enable interrupts, use the XI and ZI commands; if you want to keep the stack pointer at its original value, use SP=*nn*. (This paragraph may not make sense first time around. Reread it after looking over the list of commands.)

The input prompt is a six-space indentation, a la APL. You can type in up to 20 characters per line, including blanks and the carriage return. (Blanks are ignored when a command is decoded.) Use the RUB key to erase typos. If necessary, rub right back to the beginning of the line – you can't go past it.

The error message is a question mark.

A few words of caution are in order at this point. When you look at the load commands you'll see that there's a big difference between changing the contents of register buffers and the contents of memory. If you give the command A=015, what you're doing is loading 015 into a buffer, which will in turn be loaded into Z-80 register A when you give an execute command. But if you say (001001)=015, you really and immediately put the value 015 into memory location 001001. Think twice.

When you set a breakpoint at a certain address, the debugger copies the byte it finds there into a buffer and replaces it with 377, which is the RST 070 instruction. This restart is vectored back to the debugger, which will take out the 377 and restore the original byte. Be sure that the breakpoint address you give points to the first byte of an executable instruction in the target program, or that program will surely bomb. If it does, for this or any other reason, then control won't return to the debugger, and you must remember that the instruction at the breakpoint address is RST 070 and not the original article.

The Z (or ZI) command allows you to jump to the most recently set breakpoint, which means the last breakpoint set *since you loaded the debugger*. Be aware of that, and of the fact that if you jump to a breakpoint and set a new breakpoint at the same address, you'll bounce right back.

The debugger prefers to work in octal. You can switch it to hex with the M command, but it will reset itself to octal mode when you come back from a breakpoint. If you enter a hex value in octal mode, you'll get an error message; but an octal number may be accepted when you're working in hex, with incorrect results.

1K Debugger Summary

Conventions :

Punctuation marks, math symbols and capital letters must be typed at the keyboard just as they appear here.

Lower-case n stands for any 8-bit value, nn for any 16-bit value, d for any 8-bit two's-complement displacement. When the debugger is in octal mode these values will be given in octal notation, when in hex in hex.

Brackets indicate a choice. For example, { ± } means that you may enter either a plus sign or a minus sign. (But you must use one or the other.)

Square brackets enclose optional entries. For instance, H [L] means that you may enter either HL or just H.

Input :

The debugger prompts for input by indenting six spaces. The rubout character (377) backspaces the cursor and erases the previous entry. You can type in up to 20 characters per line, including blanks and carriage return. When a command is decoded the blanks are ignored. The 21st character creates an error condition, and all input is scrubbed.

COMMANDS

1. SCREEN CLEAR COMMAND

!	Clears TV screen.

2. MODE COMMANDS

2.1 Display Mode

M	Returns H if mode is hex, 0 if mode is octal

2.2 Set Mode

M{0 / H}	0 sets octal mode, H sets hex

3. ARITHMETIC COMMANDS

3.1 Binary Display

N n	Displays n in binary notation

3.2 Two's Complements

T n	Returns the two's complement (NEG)

3.3 Addition and Subtraction

± nn, nn	Using reverse Polish notation, finds the sum of two 16-bit values or the difference between them. The comma may be any non-blank character. Note: octal arithmetic is done in *split* octal, where 000377 + 000001 = 001000

4. DISPLAY COMMANDS

4.1 8-bit Register Display

{A / F / I / R}	Returns the value currently stored in the specified register buffer. Command 3.1 may be used to interpret F

4.2 16-bit Register Display

{H [L] / B [C] / D [E] / S [P] / IX / IY}	Returns the value currently stored in the specified 16-bit register buffer. Note that L, C and E can't be displayed as single registers

4.3 Memory Display

4.3.A Extended addressing

(nn)	Returns the value currently stored at memory address nn

4.3.B Register indirect addressing

({HL / BC / DE / SP}) = n	Returns the value currently stored at memory address specified by the contents of the 16-bit register buffer

4.3.C Indexed addressing

({IX / IY} + d)	Returns the value stored at memory address specified by contents of the 16-bit buffer, offset by the two's -complement displacement d. The plus sign may be any nonblank character.

5. LOAD COMMANDS

5.1 Load 8-bit Register

{A / F / I} = n	Loads the value n into the specified register buffer.

5.2 Load 16-bit Registers

{HL / BC / DE / SP / IX / IY} = nn	Loads the value nn into the specified 16-bit register buffer

5.3 Load Memory

5.3.A Extended addressing

(nn) = n	Loads the 8-bit value into memory at the address specified by the 16-bit value

5.3.B Register indirect addressing

({HL / BC / DE / SP})	Loads the 8-bit value into memory at the address specified by the contents of the of the 16-bit register buffer

5.3.C Indexed addressing

({IX / IY} + d) = n	Loads the 8-bit value into memory at the address specified by contents of the 16-bit buffer, offset by the two's-complement displacement d. The plus sign may be any nonblank character.

6. EXECUTE COMMANDS

6.1 Execute from a Given Address

X [I] nn	Loads the Z-80 registers from the debugger's register buffers and jumps to address nn. If the command is XI, interrupts are enabled

6.2 Execute from Last Breakpoint

Z [I]	Loads the Z-80 registers from the debugger's register buffers and jumps to the most recently set breakpoint. If no breakpoint has been set, control passes back to the debugger. If the command is ZI, interrupts are enabled

7. BREAKPOINT COMMANDS

7.1 Set Breakpoint and Execute from a Given Address

X [I] nn, nn	Loads the Z-80 registers from the debugger's register buffers and jumps to address specified by the first 16-bit value, with breakpoint set at address specified by the second 16-bit value. If the command is XI, interrupts are enabled

7.2 Set New Breakpoint and Execute from Last Breakpoint

Z [I] , nn	Loads the Z-80 registers from the debugger's register buffers and jumps to the most recently set breakpoint, with a new breakpoint set at address specified by the 16-bit value. If the commands are ZI. Interrupts are enabled

7.3 Display Last Breakpoint

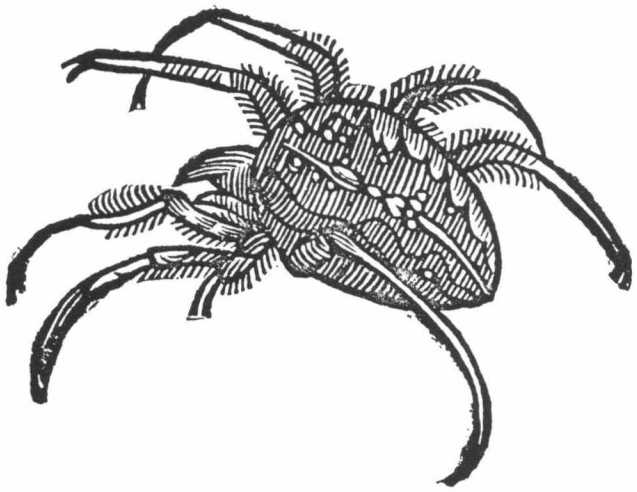
E	Displays the most recently set breakpoint, the one to which Z or ZI will jump

8. DUMP COMMAND

?	Loads the Z-80 registers from the debugger's register buffers and jumps to address 003000. All the usual monitor program/dump functions are available, but the command R will return control to the debugger and not to the monitor

9. REGISTER SWAP COMMAND

&	Loads the Z-80 registers from the debugger's register buffers, then swaps them with alternate registers and returns control to the debugger



FOOTNOTES TO THE CODE

1: *Line 180.* The TVC-64 monitor vectors RST 070 to the address contained at 001025 – 001026. We want that to be the debugger's start address–in this recension, 074061.

2: *Line 270.* The byte at 001247 is a mode code. If it's ASCII 'H', the current mode is hex; if not, octal. Subroutines NUMOUT and FINDMO, and the mode-set-or-display routine SHOWMO, make use of this Digital Group convention.

3: *Line 370.* When the debugger is initialized, FLAG and FLAG + 1 contain a jump instruction as shown. As soon as you set a breakpoint, that instruction is replaced by a NOP and a load-HL-immediate. ZBP and ZBP + 1 always contain the address of the last breakpoint set. (It's initialized to the program's start address.) BPBUF contains the byte located at (ZBP). When you come back from a breakpoint HL is loaded with the breakpoint address, A is loaded with the byte to be replaced in the target code, the byte is put back where it belongs, and the instructions at FLAG and FLAG +1 are reset to JR NN0.

4: *Lines 430, 1070.* Users of TVC-64 return from the system dump by jumping to the address at 003376 – 003377. This is normally 005000, the monitor's entry point. The debug command '?' will substitute the address NN0. So you can go straight into the excellent system dump, use all its features, then return to the debugger without disturbing the contents of the register buffers.

5: *Line 3290.* This subroutine was cribbed from Dave Bryant's PHIMON listing, published by the Digital Group. (PHIMON is DGI's Phi-Deck monitor.) I changed the code to work with TVC-64, but the hardware is a mystery to me, so I've left out comments. (This is another way of saying that I don't know how the subroutine works.)

The TVC-64 support routines used by the 1K Debugger are as follows:
TV–prints ASCII byte in the accumulator on TV screen
SPACE–prints one blank space
ERASE–clears TV
KEYBD–gets a byte from the keyboard
NUMOUT–prints the value in register E as two hex digits or three octal digits, depending on what's in 001247. (See note #2.)

```
074061                  0100 * * * * * * * * * * * * * * * * * *
074061                  0110 * 1K DEBUGGER, BY LES HANCOCK *
074061                  0120 *     VERSION  1/11/1978      *
074061                  0130 * * * * * * * * * * * * * * * * * *
074061                  0131 ERASE  EQU  000346     Erases TV screen
074061                  0132 TV     EQU  000372     Prints 1 character
074061                  0133 SPACE  EQU  000370     Prints 1 blank
074061                  0134 NUMOUT EQU  002106     Prints byte as hex or oct.
074061                  0135 KEYBD  EQU  001250     Gets 1 byte from keyboard
074061                  0136 *
074061                  0137 * LOAD BUFFERS FROM REGISTERS
074061                  0138 *
074061 355 163 301 076  0140 ST     LD   (SBUF),SP  Save stack pointer
074065 061 061 074      0150        LD   SP,ST      Set stack pointer to ST
074070 363              0160 ST2    DI               Disable interrupts
074071 042 270 076      0170        LD   (HBUF),HL  Save HL registers
074074 041 061 074      0180        LD   HL,ST      See note #1
074077 042 025 001      0190        LD   (001025),HL
074102 335 345          0200        PUSH IX         Save IX
074104 341              0210        POP  HL
074105 042 250 076      0220        LD   (XBUF),HL
074110 375 345          0230        PUSH IY         Save IY
074112 341              0240        POP  HL
074113 042 254 076      0250        LD   (YBUF),HL
074116 365              0260        PUSH AF
074117 076 317          0270        LD   A,'O'      See note #2
074121 062 247 001      0280        LD   (001247),A
074124 355 127          0290        LD   A,I        Save I register
074126 062 263 076      0300        LD   (IBUF),A
074131 355 137          0310        LD   A,R        Save R register
074133 062 306 076      0320        LD   (RBUF),A
074136 341              0330        POP  HL         Save A and flags
074137 042 257 076      0340        LD   (FBUF),HL
074142 355 103 273 076  0350        LD   (ZBUF),BC  Save BC
074146 355 123 276 076  0360        LD   (DBUF),DE  Save DE
074152                  0361 *
074152                  0362 * HANDLE RETURN FROM BKPT
074152                  0363 *
074152 030 014          0370 FLAG   JR   NN0        See note #3
074154 061 074          0380 ZBP    DW   ST
074156 072 354 077      0390 UNLOAD LD   A,(BPBUF)
074161 167              0400        LD   (HL),A
074162 041 030 014      0410        LD   HL,014030
074165 042 152 074      0420        LD   (FLAG),HL
074170 041 000 005      0430 NN0    LD   HL,005000  See note #4
074173 042 376 003      0440        LD   (003376),HL
074176 315 346 000      0450 N0     CALL ERASE
074201                  0451 *
074201                  0452 * GET A COMMAND & STORE
074201                  0453 * IN THE INPUT BUFFER
074201                  0454 *
074201 006 006          0460 PRMPT  LD   B,6         Print 6 blanks
074203 315 307 076      0470        CALL LINE           as a prompt
074206 006 024          0480 SET    LD   B,20D      Set line-length counter
074210 041 377 077      0490        LD   HL,INBUF   Set pointer
074213 315 315 076      0500 ENTR   CALL KEYIN      Get byte from keyboard
074216 376 215          0510        CP   215        Is it c/r?
074220 040 012          0520        JR   NZ,NXT1    No, continue
074222 167              0530        LD   (HL),A     Yes, put into buffer
074223 076 046          0540        LD   A,38D          and print blanks to
074225 200              0550        ADD  B              the end of the
074226 107              0560        LD   B,A            64-character
074227 315 307 076      0570        CALL LINE           line
074232 030 042          0580        JR   DECODE     Then decode command
074234 376 377          0590 NXT1   CP   377        Is it rubout?
074236 040 015          0600        JR   NZ,STOR
074240 054              0610        INC  L          Yes, back up pointer
074241 050 343          0620        JR   Z,SET      Buffer empty, restart
074243 054              0630        INC  L          Back up again
074244 004              0640        INC  B          Back up counter
074245 315 331 076      0650        CALL BACKS      Backspace cursor
074250 315 372 000      0660        CALL TV
074253 030 016          0670        JR   CONTNU
074255 167              0680 STOR   LD   (HL),A     Store byte in buffer
074256 315 372 000      0690        CALL TV         Echo on TV screen
074261 020 010          0700        DJNZ CONTNU     Line not full, OK
074263 006 046          0710        LD   B,38D      Line full, error
074265 315 307 076      0720        CALL LINE
074270 303 313 075      0730        JP   ERR
074273 055              0740 CONTNU DEC  L          Bump pointer
074274 030 315          0750        JR   ENTR       Return for another byte
074276                  0751 *
074276                  0752 * DECODE COMMANDS
074276                  0753 *
074276 375 041 000 100  0760 DECODE LD   IY,INBUF+1 Set pointer
074302 315 124 077      0770        CALL GET        Get a non-blank char.
074305 376 241          0780        CP   '!'        Is it '!'?
074307 312 176 074      0790        JP   Z,NO       Yes, go back to NØ
074312 376 324          0800        CP   'T'        Is it 'T'?
074314 312 224 077      0810        JP   Z,TWOS     Yes, go to TWOS
074317 376 315          0820        CP   'M'        Is it 'M'?
074321 312 152 077      0830        JP   Z,MODEIS   Yes, go to MODEIS
074324 376 316          0840        CP   'N'        Is it 'N'?
074326 312 317 077      0850        JP   Z,BINOUT   Yes, go to BINOUT
074331 376 332          0860        CP   'Z'        Is it 'Z'?
074333 040 014          0870        JR   NZ,ANEXT
074335 052 154 074      0880        LD   HL,(ZBP)    Yes, load jump buffer
074340 042 304 076      0890        LD   (JUMP),HL  with last breakpoint
074343 315 136 077      0900        CALL INTRPT     Enable interrupts (?)
074346 303 206 076      0910        JP   BKPTO      Jump to bkpt. routine
074351 376 305          0920 ANEXT  CP   'E'        Is it 'E'?
074353 040 020          0930        JR   NZ,BNEXT
074355 052 154 074      0940        LD   HL,(ZBP)   Yes, get bkpt. into HL
074360 134              0950        LD   E,H        Get high byte into E
074361 315 106 002      0960        CALL NUMOUT     Print in hex or octal
074364 135              0970        LD   E,L        Get low byte into E
074365 315 106 002      0980        CALL NUMOUT     Print in hex or octal
074370 006 072          0990        LD   B,58D      Fill line
074372 303 325 075      1000        JP   ER
074375 376 330          1010 BNEXT  CP   'X'        Is it 'X'?
074377 312 167 076      1020        JP   Z,XSCAN    Yes, do execute routine
075002 376 250          1030        CP   '('        Is it '('?
075004 312 333 075      1040        JP   Z,REGND    Yes, go to mem. routine
075007 376 277          1050        CP   '?'        Is it '?'?
075011 040 017          1060        JR   NZ,N4
075013 041 000 003      1070        LD   HL,003000  Yes, see note #4
075016 042 304 076      1080        LD   (JUMP),HL
075021 041 170 074      1090        LD   HL,NN0
075024 042 376 003      1100        LD   (003376),HL
075027 303 246 076      1110        JP   GO
075032 376 246          1120 N4     CP   '&'        Is it '&'?
075034 302 055 075      1130        JP   NZ,N5
075037 041 050 075      1140        LD   HL,HERE    Yes, store HERE in jump
075042 042 304 076      1150        LD   (JUMP),HL  buffer; jump to load-
075045 303 246 076      1160        JP   GO         &-go routine, which
075050 331              1170 HERE   EXX              returns HERE; swap
075051 010              1180        EX   AF,AF'     registers and return
075052 303 070 074      1190        JP   ST2        to start of debugger
075055 376 301          1200 N5     CP   'A'        Is it 'A'?
075057 040 005          1210        JR   NZ,N55
075061 041 260 076      1220        LD   HL,ABUF    Yes, load HL with ABUF,
075064 030 050          1230        JR   FND        do 8-bit register routine
075066 376 306          1240 N55    CP   'F'        Is it 'F'?
075070 040 005          1250        JR   NZ,N6
075072 041 257 076      1260        LD   HL,FBUF    Yes, load HL and do
075075 030 037          1270        JR   FND        8-bit routine
075077 376 322          1280 N6     CP   'R'        Is it 'R'?
075101 040 005          1290        JR   NZ,N7
075103 041 306 076      1300        LD   HL,RBUF
075106 030 026          1310        JR   FND
075110 376 311          1320 N7     CP   'I'        Is it 'I'?
075112 040 066          1330        JR   NZ,N9
075114 375 345          1340        PUSH IY         Yes, save pointer & get
075116 315 124 077      1350        CALL GET        next non-blank char.
075121 375 341          1360        POP  IY         Restore pointer
075123 376 330          1370        CP   'X'        Is new character 'X'?
075125 050 041          1380        JR   Z,XIP      Yes, go to XIP
075127 376 331          1390        CP   'Y'        Is it 'Y'?
075131 050 042          1400        JR   Z,YIP      Yes, go to YIP
075133 041 263 076      1410        LD   HL,IBUF    Assume that it's 'I'
```

```
075136                          1411 *
075136                          1412 * FOR 8-BIT REGISTERS
075136                          1413 *
075136 315 124 077              1420 FND     CALL GET        Is next non-blank
075141 376 275                  1430         CP   '='        character '='?
075143 040 007                  1440         JR   NZ,SHO     No, do display routine
075145 315 014 077              1450         CALL NEXNUM     Get octal or hex byte
075150 167                      1460         LD   (HL),A     into 8-bit register
075151 303 201 074              1470         JP   PRMPT      buffer; get next command
075154 376 215                  1480 SHO     CP   215        Is it c/r?
075156 040 133                  1490         JR   NZ,ERR     No, error
075160 136                      1500         LD   E,(HL)     Load E with buffer byte
075161 315 106 002              1510 SEND    CALL NUMOUT     Print in hex or octal
075164 006 075                  1520         LD   B,61D      Fill line
075166 030 135                  1530         JR   ER
075170 041 250 076              1540 XIP     LD   HL,XBUF    Load HL with XBUF and
075173 030 050                  1550         JR   FIND       go to 16-bit routine
075175 041 254 076              1560 YIP     LD   HL,YBUF    Load HL with YBUF and
075200 030 043                  1570         JR   FIND       go to 16-bit routine
075202 376 310                  1580 N9      CP   'H'        Is it 'H'?
075204 040 005                  1590         JR   NZ,N10
075206 041 270 076              1600         LD   HL,HBUF    Yes, load HL with HBUF
075211 030 032                  1610         JR   FIND       and go to 16-bit routine
075213 376 304                  1620 N10     CP   'D'        Is it 'D'?
075215 040 005                  1630         JR   NZ,N11
075217 041 276 076              1640         LD   HL,DBUF
075222 030 021                  1650         JR   FIND
075224 376 302                  1660 N11     CP   'B'        Is it 'B'?
075226 040 005                  1670         JR   NZ,N12
075230 041 273 076              1680         LD   HL,ZBUF
075233 030 010                  1690         JR   FIND
075235 376 323                  1700 N12     CP   'S'        Is it 'S'?
075237 302 242 077              1710         JP   NZ,MATH    No, assume math routine
075242 041 301 076              1720         LD   HL,SBUF    Yes, load HL with SBUF
075245                          1721 *
075245                          1722 * FOR 16-BIT REGISTERS
075245                          1723 *
075245 006 002                  1730 FIND    LD   B,2        Are either of next 2
075247 315 124 077              1740 LP0     CALL GET        chars. c/r?  If so,
075252 376 215                  1750         CP   215        go to display routine
075254 050 017                  1760         JR   Z,SHOW
075256 020 367                  1770         DJNZ LP0
075260 006 002                  1780         LD   B,2        Get 2 octal or hex
075262 043                      1790         INC  HL         bytes and load them
075263 315 014 077              1800 LP1     CALL NEXNUM     into 16-bit register
075266 167                      1810         LD   (HL),A     buffer
075267 053                      1820         DEC  HL
075270 020 371                  1830         DJNZ LP1
075272 303 201 074              1840         JP   PRMPT      Get next command
075275 006 002                  1850 SHOW    LD   B,2        Print contents of
075277 043                      1860         INC  HL         16-bit register
075300 136                      1870 LP2     LD   E,(HL)     buffer
075301 315 106 002              1880         CALL NUMOUT
075304 053                      1890         DEC  HL
075305 020 371                  1900         DJNZ LP2
075307 006 072                  1910         LD   B,58D      Fill line
075311 030 012                  1920         JR   ER
075313                          1921 *
075313                          1922 * ERROR MESSAGE
075313                          1923 *
075313 061 061 074              1930 ERR     LD   SP,ST      Reset stack pointer
075316 076 277                  1940         LD   A,'?'
075320 315 372 000              1950         CALL TV         Print '?'
075323 006 077                  1960         LD   B,63D      Fill line
075325 315 307 076              1970 ER      CALL LINE
075330 303 201 074              1980         JP   PRMPT      Get next command
075333                          1981 *
075333                          1982 * DECODE MEMORY REF COMMANDS
075333                          1983 *
075333 375 345                  1990 REGND   PUSH IY         Save pointer
075335 006 003                  2000         LD   B,3        Get 3 non-blank
075337 315 124 077              2010 GET3    CALL GET            characters
075342 020 373                  2020         DJNZ GET3
075344 375 341                  2030         POP  IY         Restore pointer
075346 376 251                  2040         CP   ')'        If 3rd char. is ')',
075350 050 021                  2050         JR   Z,N14      it's register indirect
075352 315 124 077              2060         CALL GET        If not, is 1st char.
075355 376 311                  2070         CP   'I'            an 'I'?
075357 050 061                  2080         JR   Z,N18      Yes, indexed addressing
075361 315 017 077              2090         CALL MIDNUM     No, extended addressing
075364 107                      2100         LD   B,A        Get two hex or octal
075365 315 014 077              2110         CALL NEXNUM         bytes into BC and
075370 117                      2120         LD   C,A            jump to reg. ind.
075371 030 141                  2130         JR   INDR1          routine
075373 315 124 077              2140 N14     CALL GET        Get non-blank char.
075376 376 310                  2150         CP   'H'        Is it 'H'?
076000 040 005                  2160         JR   NZ,N15
076002 041 270 076              2170         LD   HL,HBUF    Yes, load HL with HBUF
076005 030 117                  2180         JR   INDR       and jump to reg. ind.
076007 376 302                  2190 N15     CP   'B'        Is it 'B'?
076011 040 005                  2200         JR   NZ,N16
076013 041 273 076              2210         LD   HL,ZBUF
076016 030 106                  2220         JR   INDR
076020 376 304                  2230 N16     CP   'D'        Is it 'D'?
076022 040 005                  2240         JR   NZ,N17
076024 041 276 076              2250         LD   HL,DBUF
076027 030 075                  2260         JR   INDR
076031 376 323                  2270 N17     CP   'S'        Is it 'S'?
076033 040 005                  2280         JR   NZ,N18
076035 041 301 076              2290         LD   HL,SBUF
076040 030 064                  2300         JR   INDR
076042 315 124 077              2310 N18     CALL GET        Get another character
076045 376 330                  2320         CP   'X'        Is it 'X'?
076047 050 006                  2330         JR   Z,XX       Yes, IX-indexed addressing
076051 376 331                  2340         CP   'Y'        Is it 'Y'?
076053 050 007                  2350         JR   Z,YY       Yes, IY-indexed
076055 030 234                  2360         JR   ERR        Neither, you blew it
076057 041 250 076              2370 XX      LD   HL,XBUF    Load HL with XBUF
076062 030 003                  2380         JR   XY
076064 041 254 076              2390 YY      LD   HL,YBUF    Load HL with YBUF
076067 315 124 077              2400 XY      CALL GET        Get "+" (or whatever)
076072 315 014 077              2410         CALL NEXNUM     Get 2's-comp. dis-
076075 116                      2420         LD   C,(HL)     placement; get contents
076076 043                      2430         INC  HL             of
076077 106                      2440         LD   B,(HL)         register
076100 305                      2450         PUSH BC             buffer
076101 341                      2460         POP  HL             into HL
076102 006 000                  2470         LD   B,0        Clear B
076104 376 200                  2480         CP   200        If displacement less than
076106 070 010                  2490         JR   C,POS          200Q, negate it
076110 355 104                  2500         NEG                 and load it
076112 117                      2510         LD   C,A            into C
076113 257                      2520         XOR  A          Reset carry flag
076114 355 102                  2530         SBC  HL,BC      Displace the address
076116 030 002                  2540         JR   SWITCH
076120 117                      2550 POS     LD   C,A        Get displacement into C
076121 011                      2560         ADD  HL,BC      Displace the address
076122 345                      2570 SWITCH  PUSH HL         Get new address into
076123 301                      2580         POP  BC             BC; jump into reg.
076124 030 006                  2590         JR   INDR1          indir. routine
076126                          2591 *
076126                          2592 * REGISTER INDIRECT
076126                          2593 *
076126 116                      2600 INDR    LD   C,(HL)     Get contents of register
076127 043                      2610         INC  HL             buffer
076130 106                      2620         LD   B,(HL)         into BC
076131 315 124 077              2630         CALL GET        Ignore next character
076134 315 124 077              2640 INDR1   CALL GET        Ditto (should be ')')
076137 315 124 077              2650         CALL GET        Get next non-blank
076142 376 215                  2660         CP   215        Is it c/r?
076144 050 007                  2670         JR   Z,SHEW     Yes, go to display routine
076146 315 014 077              2680         CALL NEXNUM     Get hex or octal byte &
076151 002                      2690         LD   (BC),A     store at indirect address
076152 303 201 074              2700         JP   PRMPT      Get next command
076155 012                      2710 SHEW    LD   A,(BC)     Display byte at
076156 137                      2720         LD   E,A        indirect address
076157 315 106 002              2730         CALL NUMOUT
076162 006 075                  2740         LD   B,61D      Fill line
076164 303 325 075              2750         JP   ER
076167                          2751 *
076167                          2752 * JUMP AND EXECUTE
076167                          2753 *
```

```
076167 315 136 077
076172 315 014 077
076175 107
076176 315 014 077
076201 117
076202 355 103 304 076
076206 315 124 077
076211 376 254
076213 040 031
076215 315 014 077
076220 107
076221 315 014 077
076224 157
076225 140
076226 016 377
076230 042 154 074
076233 176
076234 062 354 077
076237 161
076240 041 000 041
076243 042 152 074
076246
076246
076246
076246 335 041
076250
076252 375 041
076254
076256 041
076257
076260
076261 345
076262 076
076263
076264 355 107
076266 361
076267 041
076270
076272 001
076273
076275 021
076276
076300 061
076301
076303 303
076304
076306
076307
076307
076307
076307 315 370 000
076312 020 373
076314 311
076315
076315
076315
076315 076 337
076317 323 000
076321 076 001
076323 323 000
076325 315 250 001
076330 311
076331
076331
076331
076331 315 370 000
076334 021 376 004
076337 076 001
076341 315 372 000
076344 035
076345 040 370
076347 025
076350 302 337 076
076353 311
076354
```

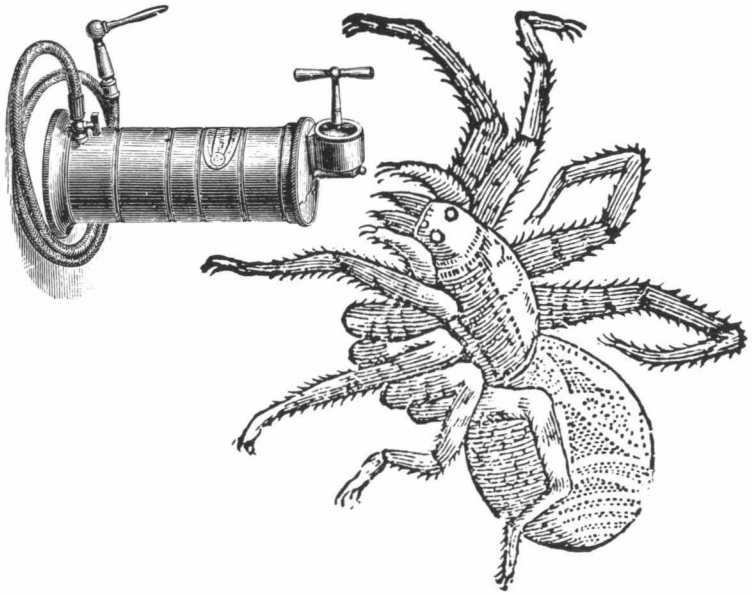
```
2760 XSCAN  CALL INTRPT     Enable interrupts (?)
2770        CALL NEXNUM     Get two hex or octal
2780        LD   B,A           bytes
2790        CALL NEXNUM        into
2800        LD   C,A           BC, then load them
2810        LD   (JUMP),BC     into jump buffer
2820 BKPTO  CALL GET        Get next non-blank char.
2830        CP   ',' Is it ','?
2840        JR   NZ,GO      No, load & go
2850        CALL NEXNUM     Yes, get two hex or
2860        LD   B,A           octal bytes with
2870        CALL NEXNUM        breakpoint address
2880        LD   L,A           into HL
2890        LD   H,B
2900        LD   C,377      377 = RST Ø7Ø
2910 BKPT   LD   (ZBP),HL   Save breakpoint address
2920 LOAD   LD   A,(HL)     Save byte at that addr.
2930        LD   (BPBUF),A     in buffer and replace
2940        LD   (HL),C        it with RST Ø7Ø
2950        LD   HL,041000  See note #3
2960        LD   (FLAG),HL
2961 *
2962 * LOAD REGISTERS FROM BUFFERS
2963 *
2970 GO     DW   041335     Load IX immediate
2980 XBUF   DS   2          with buffer contents
2990        DW   041375     Load IY immediate
3000 YBUF   DS   2
3010        DB   041        Load HL immediate with
3020 FBUF   DS   1             A and F
3030 ABUF   DS   1             buffer contents
3040        PUSH HL            & save on stack
3050        DB   076        Load A immediate
3060 IBUF   DS   1             with I buffer contents
3070        LD   I,A           and get into I
3080        POP  AF         Pop AF
3090        DB   041        Load HL immediate
3100 HBUF   DS   2
3110        DB   001        Load BC immediate
3120 ZBUF   DS   2
3130        DB   021        Load DE immediate
3140 DBUF   DS   2
3150        DB   061        Load SP immediate
3160 SBUF   DS   2
3170        DB   303        Jump to address in the
3180 JUMP   DS   2             jump buffer
3190 RBUF   DS   1
3191 *
3192 * PRINT BLANKS
3193 *
3200 LINE   CALL SPACE
3210        DJNZ LINE
3220        RET
3221 *
3223 * GET A BYTE FROM KEYBD
3224 *
3230 KEYIN  LD   A,337      Load A with cursor char.
3240        OUT  0          Send to TV
3250        LD   A,1
3260        OUT  0
3270        CALL KEYBD      Get byte from keyboard
3280        RET
3281 *
3282 * BACKSPACE
3283 *
3290 BACKS  CALL SPACE      See note #5
3300        LD   DE,4376
3310 BACKUP LD   A,1
3320        CALL TV
3330        DEC  E
3340        JR   NZ,BACKUP
3350        DEC  D
3360        JP   NZ,BACKUP
3370        RET
3371 *
```

```
076354
076354
076354 345
076355 365
076356 076 310
076360 041 247 001
076363 276
076364 341
076365 174
076366 341
076367 311
076370
076370
076370
076370 376 260
076372 332 313 075
076375 315 354 076
077000 050 004
077002 376 270
077004 030 002
077006 376 307
077010 322 313 075
077013 311
077014
077014
077014
077014
077014 315 124 077
077017 315 370 076
077022 305
077023 315 354 076
077026 050 037
077030 017
077031 017
077032 346 300
077034 117
077035 315 124 077
077040 315 370 076
077043 007
077044 007
077045 007
077046 346 070
077050 201
077051 117
077052 315 124 077
077055 315 370 076
077060 346 007
077062 201
077063 376 000
077065 301
077066 311
077067 315 113 077
077072 007
077073 007
077074 007
077075 007
077076 107
077077 315 124 077
077102 315 370 076
077105 315 113 077
077110 200
077111 301
077112 311
077113 376 272
077115 070 002
077117 326 007
077121 326 260
077123 311
077124
077124
077124
077124
077124 375 053
077126 375 176 000
077131 376 240
```

```
3372 * FIND MODE
3373 *
3380 FINDMO PUSH HL         Save HL on stack
3390        PUSH AF         Save AF on stack
3400        LD   A,'H'      If byte at 001247
3410        LD   HL,001247     is 'H', set
3420        CP   (HL)          zero flag
3430        POP  HL         Restore A
3440        LD   A,H           (but not F)
3450        POP  HL         Restore HL
3460        RET
3461 *
3462 * VALIDATE NUMBER
3463 *
3470 VALNUM CP   260        Is it less than 'Ø'?
3480        JP   C,ERR      If so, error
3490        CALL FINDMO     Is mode hex?
3500        JR   Z,VALHEX   Yes, validate hex
3510        CP   '8'        Is octal greater than '7'?
3520        JR   NXT9A         or
3530 VALHEX CP   'G'        is hex greater than 'F'?
3540 NXT9A  JP   NC,ERR     If so, error
3550        RET
3551 *
3552 * GET OCTAL/HEX BYTE
3553 * FROM INPUT BUFFER
3554 *
3560 NEXNUM CALL GET        Get non-blank character
3570 MIDNUM CALL VALNUM     Is it a valid number?
3580        PUSH BC         Yes, save BC
3590        CALL FINDMO     Is mode hex?
3600        JR   Z,NEXHEX
3610        RRCA            No, number is octal
3620        RRCA
3630        AND  300           Standard routine
3640        LD   C,A           converts
3650        CALL GET           this character
3660        CALL VALNUM        and the
3670        RLCA               two that
3680        RLCA               follow it
3690        RLCA               into one
3700        AND  070           binary value
3710        ADD  C
3720        LD   C,A
3730        CALL GET
3740        CALL VALNUM
3750        AND  007
3760        ADD  C
3770        CP   0
3780        POP  BC         Restore BC
3790        RET
3800 NEXHEX CALL HEXA       Number is hex
3810        RLCA
3820        RLCA               Standard routine
3830        RLCA               converts
3840        RLCA               this character
3850        LD   B,A           and the
3860        CALL GET           next character
3870        CALL VALNUM        into one
3880        CALL HEXA          binary value
3890        ADD  B
3900        POP  BC         Restore BC
3910        RET
3920 HEXA   CP   272
3930        JR   C,NUMBER
3940        SUB  007
3950 NUMBER SUB  260
3960        RET
3961 *
3962 * GET A BYTE FROM
3963 * INPUT BUFFER
3964 *
3970 GET    DEC  IY            Bump pointer
3980        LD   A,(IY+0)      Get byte from buffer
3990        CP   240           Is it a blank?
```

```
077133 300                4000         RET  NZ          No, return
077134 030 366            4010         JR   GET         Get next byte
077136                    4011 *
077136                    4012 * ENABLE INTERRUPTS
077136                    4013 *
077136 315 124 077        4020 INTRPT CALL GET         Get next non-blank char.
077141 376 311            4030         CP   'I'        Is it 'I'?
077143 040 002            4040         JR   NZ,NONINT  No, don't enable
077145 373                4050         EI              Enable interrupts
077146 311                4060         RET
077147 375 043            4070 NONINT INC  IY          Back up pointer
077151 311                4080         RET
077152                    4081 *
077152                    4082 * SET OR SHOW MODE
077152                    4083 *
077152 315 124 077        4090 MODEIS CALL GET         Get non-blank character
077155 376 215            4100         CP   215        Is it c/r?
077157 050 021            4110         JR   Z,SHOWMO   Yes, display mode
077161 376 310            4120         CP   'H'        Is it 'H'?
077163 050 007            4130         JR   Z,NXT0Q    Yes, set mode to hex
077165 376 317            4140         CP   'O'        Is it 'O'?
077167 050 003            4150         JR   Z,NXT0Q    Yes, set mode to octal
077171 303 313 075        4160         JP   ERR        No, error
077174 062 247 001        4170 NXT0Q  LD   (001247),A Load 001247 with mode code
077177 303 201 074        4180         JP   PRMPT      Get next command
077202 076 310            4190 SHOWMO LD   A,'H'       Does 001247 contain
077204 041 247 001        4200         LD   HL,001247     the mode
077207 276                4210         CP   (HL)          code 'H'?
077210 050 002            4220         JR   Z,NXT8C    If so, print 'H'
077212 076 317            4230         LD   A,'O'      Else print 'O'
077214 315 372 000        4240 NXT8C  CALL TV
077217 006 077            4250         LD   B,63D      Fill line
077221 303 325 075        4260         JP   ER
077224                    4261 *
077224                    4262 * TWO'S COMPLEMENT
077224                    4263 *
077224 315 014 077        4270 TWOS   CALL NEXNUM      Get hex or octal byte
077227 355 104            4280         NEG             Take its 2's complement
077231 137                4290         LD   E,A        Print the result
077232 315 106 002        4300         CALL NUMOUT
077235 006 075            4310         LD   B,61D      Fill line
077237 303 325 075        4320         JP   ER
077242                    4321 *
077242                    4322 * OCTAL/HEX MATH
077242                    4323 *
077242 365                4330 MATH   PUSH AF          Save sign (+ or -)
077243 315 014 077        4340         CALL NEXNUM     Get two hex or octal
077246 127                4350         LD   D,A           bytes into
077247 315 014 077        4360         CALL NEXNUM        DE register
077252 137                4370         LD   E,A           pair
077253 315 124 077        4380         CALL GET        Skip next char. (',')
077256 315 014 077        4390         CALL NEXNUM     Get two hex or octal
077261 107                4400         LD   B,A           bytes into
077262 315 014 077        4410         CALL NEXNUM        BC register
077265 117                4420         LD   C,A           pair
077266 353                4430         EX   DE,HL      Swap DE and HL
077267 361                4440         POP  AF
077270 376 253            4450         CP   '+'        Was the sign '+'?
077272 050 005            4460         JR   Z,PLUS
077274 257                4470         XOR  A          If not, clear carry
077275 355 102            4480         SBC  HL,BC         and subtract
077277 030 001            4490         JR   OUT
077301 011                4500 PLUS   ADD  HL,BC      If so, add and
077302 134                4510 OUT    LD   E,H           print the result
077303 315 106 002        4520         CALL NUMOUT
077306 135                4530         LD   E,L
077307 315 106 002        4540         CALL NUMOUT
077312 006 072            4550         LD   B,58D      Fill line
077314 303 325 075        4560         JP   ER
077317                    4561 *
077317                    4562 * DISPLAY IN BINARY
077317                    4563 *
077317 315 014 077        4570 BINOUT CALL NEXNUM      Get hex or octal byte
077322 006 010            4580         LD   B,8D       Set counter for 8 bits
077324 007                4590 AGAIN  RLCA            Rotate left with carry
077325 117                4600         LD   C,A        Save A in C
077326 070 004            4610         JR   C,AONE     If carry, print '1'
077330 076 260            4620         LD   A,'0'      If not, print 'Ø'
077332 030 002            4630         JR   OUT2
077334 076 261            4640 AONE   LD   A,'1'
077336 315 372 000        4650 OUT2   CALL TV
077341 315 370 000        4660         CALL SPACE
077344 171                4670         LD   A,C        Restore A
077345 020 355            4680         DJNZ AGAIN
077347 006 060            4690         LD   B,48D      Fill line
077351 303 325 075        4700         JP   ER
077354                    4701 *
077354                    4702 * INPUT BUFFER
077354                    4703 *
077354                    4710 BPBUF  DS   19D
077377                    4720 INBUF  DS   1          This address must
                                                      be ***/377
```

```
SYMB
ERASE  000346  TV      000372  SPACE  000370  NUMOUT 002106
KEYBD  001250  ST      074061  ST2    074070  FLAG   074152
ZBP    074154  UNLOAD  074156  NNO    074170  NO     074176
PRMPT  074201  SET     074206  ENTR   074213  NXT1   074234
STOR   074255  CONTNU  074273  DECODE 074276  ANEXT  074351
BNEXT  074375  N4      075032  HERE   075050  N5     075055
N55    075066  N6      075077  N7     075110  FND    075136
SHO    075154  SEND    075161  XIP    075170  YIP    075175
N9     075202  N10     075213  N11    075224  N12    075235
FIND   075245  LP0     075247  LP1    075263  SHOW   075275
LP2    075300  ERR     075313  ER     075325  REGND  075333
GET3   075337  N14     075373  N15    076007  N16    076020
N17    076031  N18     076042  XX     076057  YY     076064
XY     076067  POS     076120  SWITCH 076122  INDR   076126
INDR1  076134  SHEW    076155  XSCAN  076167  BKPT0  076206
BKPT   076230  LOAD    076233  GO     076246  XBUF   076250
YBUF   076254  FBUF    076257  ABUF   076260  IBUF   076263
HBUF   076270  ZBUF    076273  DBUF   076276  SBUF   076301
JUMP   076304  RBUF    076306  LINE   076307  KEYIN  076315
BACKS  076331  BACKUP  076337  FINDMO 076354  VALNUM 076370
VALHEX 077006  NXT9A   077010  NEXNUM 077014  MIDNUM 077017
NEXHEX 077067  HEXA    077113  NUMBER 077121  GET    077124
INTRPT 077136  NONINT  077147  MODEIS 077152  NXT0Q  077174
SHOWMO 077202  NXT8C   077214  TWOS   077224  MATH   077242
PLUS   077301  OUT     077302  BINOUT 077317  AGAIN  077324
AONE   077334  OUT2    077336  BPBUF  077354  INBUF  077377
```



EXIT THIS REVIEWER

Jef Raskin
Director of Communications,
Apple Computer Co.
Received: 78 Jan 19

I have accepted employment with APPLE Computers, Inc. It is a case of putting their money where my mouth is, and now I will have a chance to apply my criticism from inside a company. Since I will be Director of Publications there you will know who to blame if, in a year's time, their manuals aren't up to snuff. Some other reviewer will have to judge if I've succeeded.

This (except for a few that are in the works and may appear in print after this one) is my last review. I will probably continue to write *opinion* articles, but will no longer cling to the mantle of objectivity. At the end of this article I will have a few words to say about my new employer's products.

But on to a review. A reader called and asked me about Electronic System's kits. He pointed out that the company looked fishy, having a Santa Clara phone number and a P.O. Box in Burlingame (Box 212, Burlingame, CA 94010). No street address is given. Burlingame is in San Mateo County. I told the reader that I would find out what was going on. I called them up, and after evading the answering service to talk to the real wheels, found out that the separation was due to their desire to keep a continuity of address for their customers after they had moved. Fair enough. They use a box rather than a street address to prevent their being ripped off (which apparently has happened once). They went a bit further and agreed to send *Dr. Dobb's* a small kit for review. Any company brave enough to do that can't be all bad.

Their catalog is labeled: "Hobby Computer Kits", but should be subtitled: "mostly for those with a strong electronic bent". These kits are not for beginners, nor are they intended to be. In fact, Electronic Systems prefers to just sell boards for most of their kits. You can scrounge up the parts, follow their schematic, and modify away to your heart's content. The kit I got was their RF Modulator. It's $13.50 *with* built in power supply and regulator (to run off a 12VAC center tapped transformer, 8 volt unregulated supply, or it can run from a regulated 5 volt supply) and with the parts supplied. Almost. The obvious unit for comparison is the Pixieverter. The Pixieverter (which works fine) requires –6 volts which I find sometimes hard to come by in my TTL breadboarding. So I appreciate Electronic System's choice of operating voltage. It also makes their unit a very good buy for the money.

There are no "put square peg 1 into square hole 1. . . " type instructions. Just a schematic and in this case, a slightly off-register but clearly silkscreened parts placement on the printed circuit board. Layout was clean and professional. I discovered that C7, a 3Pf capacitor was missing from their parts list in the catalog, on the package, and was missing from the kit too. Consistent, at least. Unfortunately, the schematic and the circuit do require C7. I didn't have any in my parts box so I just twisted a 3/8 inch pair of solid insulated wires together to make a capacitor (what Hams calls a "gimmick"). 3 Pf isn't much capacitance, and any two conductors in the same room together have some capacitance between them.

You have to wind a coil yourself. There is a note telling you how to do this:

COIL –3 3/4 turns of 22 AWG wire wound on a no. 2 pencil.

I quote this exactly. If you can't recognize a piece of 22 AWG wire you shouldn't be building this kit. It just refers to the gauge of the wire. You also should know that this wire gauge is not too critical. If you get a finer gauge, the coil will be rather unsturdy, but will work. A little heavier wire is OK too. The kicker is the specification of a no. 2 pencil. Since the pencil does not become part of the circuit (it is just the form for the coil–although they doen't say so) it is unclear just what difference the hardness of the lead makes. What they meant was to use a standard 5/16 inch thick pencil as a form for winding the coil. A 5/16 dowel will work. Again, this kit is meant for people who will automatically figure this out for themselves. No hint is given as to how to read the values of the components given with the kit. The components seem to be of acceptable quality. The capacitors that were supposed to be Silver Mica were indeed of that type.

While I do think that Electronic Systems could be a bit more careful in their documentation, I believe that there is a place in this world for kits such as these, where one can substitute knowledge and experience for money. These kits are inexpensive, and in this reviewer's opinion, a good value. Those who are not in a position to figure out sparse instructions will have to pay more elsewhere. Many of their kits seem fascinating, and there are a number that seem downright handy. They have a UART and baud rate generator kit for $35.00 (board and schematic only $12.00), a modem kit for $27.50, a very appealing RS-232 to and from TTL converter kit for $7.00, and some others. Write for their catalog.

Electronic Systems also offers a memory board and their TIDMA–which is a Tape Interface Direct Memory Access board–which only comes assembled and tested. It sounds fascinating , but you will have to write to them for details. Their catalog contains schematics, and is quite educational if you know which squiggle means what.

I promised a few words about my new employer's computer. The APPLE: it is a very neatly packaged 11 pound computer that is unusually complete. It uses a home TV (preferably color) for output. It comes with built-in speaker and joysticks and the associated interfaces, and with a built-in cassette interface. The keyboard is built in as well–and it is a pretty good keyboard (compared to the PET it is an excellent keyboard). BASIC is in ROM and comes up at the press of (three) buttons. The APPLE is fun. But it is not, as it stands, too serious. You can't (from within BASIC) store data, but only programs on the cassette. The screen is upper case only, so you can't do text editing. The BASIC is integer only, so you can't do business or most scientific applications–although APPLE *does* provide a floating point BASIC on cassette. It is called "APPLESOFT" as it was written by Microsoft which has provided so much of the personal computer software. Both APPLE BASICs provide color graphics in an easy to use form, which is one of the chief charms of the machine. It is probably the best computer around to use to teach programming in BASIC.

The hardware is especially good, it is a very reliable machine, as I have learned in some 12,000 miles of traveling with one in its optional leatherette (i.e. plastic) carrying bag. The switching power supply is both light in weight and more than adequate, and I have had no overheating problems at all. You can't even warm your hands with it in a cold hotel room. Unlike the PET which limits you to a measly 8K, the APPLE II has room for 48K of RAM inside the little box, and upping the memory merely requires changing three jumper blocks and slipping the appropriate chips into the sea of sockets provided. An 8K APPLE costs $995 compared to the $795 PET, but the PET does contain its own cassette recorder and video display. My thoughts on the PET were published in the September/October (1977) issue of *Personal Computing*. The APPLE II is a lot classier.

The existing APPLE II documentation is very poor. But this is what I am trying to change, as I said. I think the APPLE has a lot of potential, otherwise I wouldn't have gone to work for the company. It's not on the S-100 bus (it's not 8080 or

8080 QUICKSORT

by Mike Gabrielson
P.O. Box 2692
Stanford, CA 94305

```
1                       ********************************************************
2                       * QUICKSORT - MIKE GABRIELSON - 12/1/77                *
3                       *                                                      *
4                       * ACCEPTS:  DE = ADDRESS OF FIRST RECORD TO SORT       *
5                       *           HL = ADDRESS OF LAST RECORD TO SORT        *
6                       *                                                      *
7                       * QUICK IS A RECURSIVE ROUTINE...                     *
8                       *                                                      *
9                       *       PROCEDURE QUICK (DE, HL)                       *
10                      *       IF DE >= HL THEN RETURN                        *
11                      *       ELSE PARTITION TO FIND K                       *
12                      *       QUICK (K+1, HL)                                *
13                      *       QUICK (DE, K-1)                                *
14                      *       RETURN                                         *
15                      *                                                      *
16                      * PARTITIONING SWAPS DATA AND PRODUCES AN ADDRESS K,   *
17                      * DE <= K <= HL, SUCH THAT ALL DATA TO THE LEFT OF K   *
18                      * IS LESS THAN THE DATA AT K, AND DATA TO THE RIGHT    *
19                      * IS GREATER.  THIS IS DESCRIBED IN VOLUME 3 OF KNUTH. *
20                      * THE ROUTINE IS CODED FOR SORTING ONE BYTE RECORDS,   *
21                      * BUT THE SIX COMMENTS INDICATE WHAT TO CHANGE TO SORT *
22                      * RECORDS OF OTHER LENGTHS.                            *
23                      ********************************************************
24 0000 R CD3200        QUICK   CALL    DE_HL
25 0003   D0                    RM
26 0004   D5                    PUSH    RD
27 0005   E5                    PUSH    RH
28 0006 R CD3800        LOOP    CALL    DE__HL
29 0009 R DA1F00                BL      RIGHT
30 000C R CD3800                CALL    SWAP
31 000F   13            LEFT    INCP    RD                      1 - MAKE DE POINT
32 0010 R CD3200                CALL    DE_HL                   TO NEXT RECORD
33 0013 R CA2600                BE      PARTED
34 0016 R CD3800                CALL    DE__HL
35 0019 R DA0F00                BL      LEFT
36 001C R CD3800                CALL    SWAP
37 001F   2B            RIGHT   DECP    RH                      2 - MAKE HL POINT
38 0020 R CD3200                CALL    DE_HL                   TO PREVIOUS RECORD
39 0023 R C20600                BNE     LOOP
40 0026   E3            PARTED  XTHL
41 0027   13                    INCP    RD                      3 - MAKE DE POINT
42 0028 R CD0000                CALL    QUICK                   TO NEXT RECORD
43 002B   E1                    POP     RH
44 002C   2B                    DECP    RH                      4 - MAKE HL POINT
45 002D   D1                    POP     RD                      TO PREVIOUS RECORD
46 002E R CD0000                CALL    QUICK
47 0031   C9                    RET
48 0032   7A            DE_HL   LR      RA,RD                   COMPARE DE TO HL
50 0033   BC                    CR      RA,RH
51 0034   C0                    RNE
52 0035   7B                    LR      RA,RE
53 0036   BD                    CR      RA,RL
54 0037   C9                    RET
55                      *
56 0038   1A            DE__HL  LBA     RD                      5 - COMPARE RECORDS
57 0039   BE                    C       RA                      POINTED TO BY DE, HL
58 003A   C9                    RET
59                      *
60 003B   1A            SWAP    LBA     RD                      6 - SWAP RECORDS
61 003C   46                    LB      RB                      POINTED TO BY DE, HL
62 003D   77                    STB     RA
63 003E   7B                    LR      RA,RE
64 003F   12                    STA     RD
65 0040   C9                    RET
66                      *
67                              END
```

Z-80 based so it wouldn't be anyway) and there aren't a lot of accessories available for it yet. On the other hand, it comes with more built in than any S-100 computer, e.g. color graphics a la Cromemco's dazzler, A/D to run joysticks (and it comes with two single axis sticks), cassette interface, BASIC in ROM, programmable louspeaker output, keyboard, room for memory expansion without buying boards, etc. So it's a toss-up in that regard. I am not getting rid of my S-100 computer.

I expect APPLE to come out with RS-232 interface, upper and lower case display, floppy disk interface, and all the other goodies that I can put in my Poly-88 (which, dear fans, still runs great).

I would like to thank the hundred or so readers who sent in helpful letters spurring me on to write more reivews, and the three or four who gave me hard, but useful criticism. Thanks to Jim Warren for putting up with me, and to the neat people at People's Computer Company who publish this Journal. And a big thank you, and my admiration to the many manufacturers who sent us equipment for scrutiny, and hope that the new reviewer(s) get as much cooperation as I did.

ELECTRIC PENCIL WORD PROCESSOR FOR NORTH STAR DISK

News Release *Received: 77 Dec 28*

The Electric Pencil character-oriented word processing software allows text to be entered and manipulated at any location as a continuous string of characters. Lines are formatted automatically. Features include cursor controls, global search and replace functions, bi-directional scrolling, right and left justification, wrap around text as well as character, line or block insert and delete. Page formatting is at the control of the user. The Electric Pencil has its own DOS. Storage and retrieval of text is simple and automatic. Six versions are available for North Star including SOL-20, VDM-1 or VTI videos, standard printer (TTY, etc.) or Diablo HyTerm. Minimum system hardware for 8080 or Z-80 micro is 12K of memory, printer plus interface, video monitor plus VDM-1 or VTI interface, North Star Disk and/or Tarbell or CUTS cassette interface and recorder. Available from local dealers or from Michael Shrayer; 3901 Los Feliz Blvd.; Los Angeles, CA 90027.

PERCOM CIS-30+ CASSETTE INTERFACE TO MITS 680B

News Release *Received: 78 Jan 13*

Percom Data Company released technical memos describing how to use their standard CIS-30+ interfacing unit with the MITS 680b computer. Long used with the SWTP 6800, the CIS-30+ interfaces two cassette recorders and a data terminal to 6800-based microcomputers.

The CIS-30+ provides user-selected cassette data rates of 30, 60, or 120 bytes per second, and data terminal rates of 300, 600, or 1200 baud. The two cassette interfacing circuits are independent and operations such as cross filing therefore are possible. Optional program control of recorders is available.

Cassette data recording is KC-Standard/Bi-Phase-M (double frequency). The data terminal communication mode is full duplex.

The MITS 680b PROM monitor may be used for loading and running programs, except for flipping the tape switch to ON for program loading.

The CIS-30+ sells for $79.95 in kit form, and $99.95 assembled and tested. An instruction manual is included. For MITS 680b applications, users should request PERCOM Tech Memo TM-CIS-30-09, "Using the PERCOM CIS-30+ with the MITS 680b Computer."

Optional accessories include an IC socket kit ($4.95), a Remote Control Kit (for program control of recorders) ($14.95), and a test cassette with operating software ($4.95).

Contact PERCOM Data Company, 318 Barnes, Garland, Texas 75042; (214) 276-1968.

SMAL/80 MACRO-ASSEMBLY LANGUAGE FOR 8080 AND 8085

News Release *Received: 78 Feb 6*

CHROMOD Associates has announced the development of SMAL/80, a compiled, structured, macro-assembly language for 8080 and 8085 microprocessors that requires 7K of memory.

SMAL/80 statements are written in a symbolic notation resembling PASCAL and PL/M that simplifies the writing of assembly language programs. SMAL/80 also incorporates the basic structured-programming constructs, the DO-END, IF-THEN-ELSE, and LOOP-REPEAT, which may be combined with and/or nested within each other without limit to form complex statements. The code produced by the compiler is as efficient as that written in a traditional assembly language by a skilled programmer.

The SMAL/80 package includes a 2K macro preprocessor written in SMAL/80 that extends the usefulness of the language. The macro preprocessor permits conditional expansion of statements, unlimited nesting of macros, and has a natural notation.

Also included in the SMAL/80 package is a translator program that allows one to convert any 8080 or 8085 program written in standard Intel mnemonics into SMAL/80, but without the constructs.

SMAL/80 is a revision of SMAL, and 8080 language developed by Dr Charles Popper at Bell Labs. The original SMAL was implemented via a SNOBAL preprocessor running on an IBM 370/168. SMAL/80, also developed by Dr Popper, has been redesigned and can now be compiled directly on any 8080, 8085, or Z-80 machine.

SMAL/80 is being offered initially in CP/M and ISIS I disk formats. Price, including documentation, is $75. In the near future, a Z-80 version of SMAL/80 in the same disk formats will also be offered, as will various relocatable tape cassette formats that incorporate linking loaders.

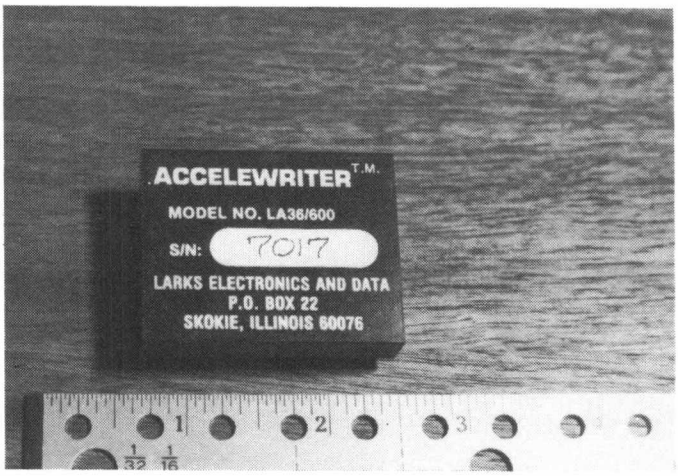
Write CHROMOD Associates, P.O. Box 3169, Grand Central Station, New York, N.Y. 10017.

STANDARD BUBBLESORT

```
SIZE:   EQU     255         ;max size of ARRAY
LOOP1:  MVI     D,Ø         ;reset "switched" flag for this
                            ;iteration
        LDA     N
        MOV     B,A         ;number of list elements
        LXI     H,ARRAY
LOOP2:  DCR     B
        JZ      NEXT1       ;no more pairs left
        MOV     A,M         ;first element of pair
        INX     H
        CMP     M
        JP      LOOP2
        MOV     C,M         ;do an interchange
        MOV     M,A
        DCX     H
        MOV     M,C
        INX     H
        MVI     D,1         ;set the "switched" flag
        JMP     LOOP2
NEXT1:  MOV     A,D
        ORA     A
        JNZ     LOOP1
        HLT

N:      DB      SIZE
ARRAY:  DS      SIZE

END
```

SMAL/80 BUBBLESORT

```
SIZE EQU 255                        /* max size of ARRAY */;
LOOP;
   D = Ø                 /* reset "switched" flag for this loop
                                                  iteration */;
   A = M(N);
   B = A                          /* number of list elements */;
   HL = ARRAY;
      LOOP;
         IF --B ZERO               /* if no more pairs left */
            BREAK;
         A = M(HL)                 /* first element of pair */;
         ++HL;
         IF A : M(HL) NEG
            THEN DO                          /* an interchange */;
               C = M(HL);
               M(HL) = A;
               --HL;
               M(HL) = C;
               ++HL;
               D = 1               /* set the "switched" flag */;
            END;
   REPEAT;
REPEAT WHILE [A = D; A = A OR A] NOT ZERO;

N:      BYTE     SIZE;
ARRAY:  RESERVE  SIZE;

END PROGRAM;
```



ADAPTER DOUBLES DECWRITER SPEED

News Release *Received: 78 Jan 23*

The Larks Accellewriter™ adapts any LA36 DECwriter to operate at 600 baud. The adapter converts the standard 110/150/300 baud DECwriter to 110/300/600 baud. It changes the internal timing of the DECwriter and causes it to print at 60 characters-per-second. Installation typically takes less than one hour and requires the removal of two integrated circuits from the logic board of the DECwriter. These are replaced with low-profile IC sockets. The Accelewriter is then installed in the board in place of the two original IC's, and the logic board is reinstalled in the DECwriter.

The DECwriter can be reconverted to its original electronic configuration by unplugging the adapter and installing IC's of the original types in the sockets. This takes less than one minute and requires no tools. The Accelewriter causes no permanent modifications to the DECwriter and does not interfere with the installation of standard DEC™ option boards. Price is $95 and delivery is from two to four weeks. Installation is available in selected areas. Contact Larks Electronics and Data, Box 22, Skokie, IL 60077; (302) 677-6080.

A Line-Oriented PILOT Editor

BY DAVID J. MORGAN
University of Texas at Dallas
P.O. Box 688
Richardson, TX 75080

Dear DDJ:

*I have written a line oriented editor for Dr. Starkweather's PILOT (*DDJ, *April, May, 1977). My program takes an additional 629 bytes of memory and provides nine editor commands.*

The attached program listing shows the assembled output and assumes that the PILOT source program (as shown in the May, 1977 DDJ*) has been assembled with it. User documentation, in the format of the April, 1977* DDJ *PILOT documentation, describes the operation of the editor. Thus, the routine can be implemented by assembling my code at the end of Starkweather's and filing the documentation in place.*

Interfacing to PILOT is easy due to the modularity and well planned design of the original program. The steps are as follows:

1. *Since we are extending PILOT we must change PBUFB which indicates where the user program area starts. Change the following line (originally the sixth line of the program on p. 18 of* DDJ, *May, 1977):*

```
        PBUFB   EQU   1000H
  to:   PBUFB   EQU   11A7H
```

2. *Change the JMP in the vectors (so that when the user enters edit mode the machine does not jump to the monitor) from:*

```
        0112   C360E0   EDIT:   JMP   MNTR
  to:   0112   C3EA0E   EDIT:   JMP   EDTR
```

3. *Similarly, change the following:*

```
        0309   C360E0            JMP   MNTR
  to:   0309   C3EA0E            JMP   EDTR
```

My routines call Starkweather's I/O (DSPLY, CHI, KEYIN, CHO) and his block transfer routine (BLKTFR). I also pickup the EBUFF address (the entry buffer address) and PBUFE (End of user program area) as set by the programmer (p. 18, DDJ, *May, 1977, 7th line). Any alterations made to any of these routines or addresses should be transparent to my program. That is, it should run anyway.*

I also have implemented the PILOT command LOAD: for loading a PILOT program from an iCOM disk (and constructed with iCOM BUILD or EDIT commands) running with FDOS-II and a TIME: command for use with a Comptek (now Cañada) CL2400 real time clock. I'll be glad to send these routines to anyone who wants a copy. I'll also send the editor routine on a diskette if you send me one to write on (iCOM compatible) but I guess I can't include all of PILOT on disk because of the copyright.

Sincerely,
David J. Morgan
Assistant Professor

P.S. Please mention that Binary Systems, Inc. of Richardson, Texas (The Micro Store) was good enough to let me use their machine and a new ribbon to produce a hardcopy output. They are pretty good people.

After invoking the PILOT EDIT: command the system will respond:

PILOT EDITOR READY

If there is no current user program in memory the system will then display:

PGM SPACE EMPTY – ENTER 'I' OR 'E'

indicating that the user must either insert text or exit from the editor. The system will then prompt the user with an asterisk (*). After the prompt is displayed, any of the following editor commands may be entered.

B Backup the line pointer. This routine backs up the internal pointer to the previous line of text and then displays that line. The previous line of text is then made the current line. If this command is invoked when the current line is the first line of the program (and hence no backup is allowed) the message:

– START OF FILE –

is displayed and the current line is left at the first line of the program.

C Display the "current" line of text. When the editor is entered from PILOT and a program is in memory, the current line is automatically set to the first line of the program.

D Delete the current line of text. The current line is removed from the program and all following lines are moved up. If there are no lines left to delete, the message:

PGM SPACE EMPTY – ENTER 'I' OR 'E'

is displayed indicating that a program line must be entered or the user must exit the editor function.

If the delete is allowed, the next sequential line becomes the current line unless there are no more lines in the file. If no more lines exist in the file the last line (before the deletes started) is made the current line.

E End of editing. Returns to PILOT mainline.

I Insert a new line of text. After this command is entered the editor will respond with an equal sign prompt (=). The user may then enter a new program line. The new line will be terminated by a carriage return.

The insertion is made immediately after the current line and the new line, once entered, becomes the current line. If there is no space available to insert the line the following appears:

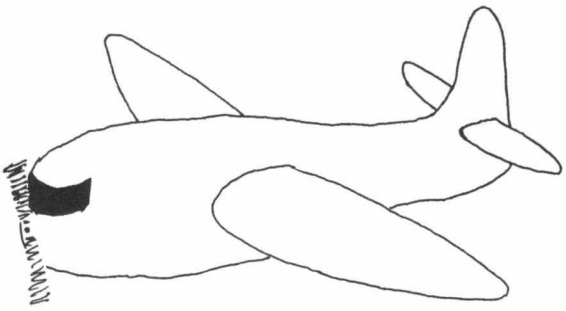
– PROGRAM SPACE FULL –

The insert will not take place but no other action will be taken and the editor will prompt for a new command.

N Display the next line of text. The routine advances the internal pointer to the next line of text and then displays that line. The next line is then the current line. The search for end of line is always to the carriage return character (0D hex). If there are no more lines in the program to display, the message

– END OF PROGRAM –

appears and the current line is left at the last line of the program.

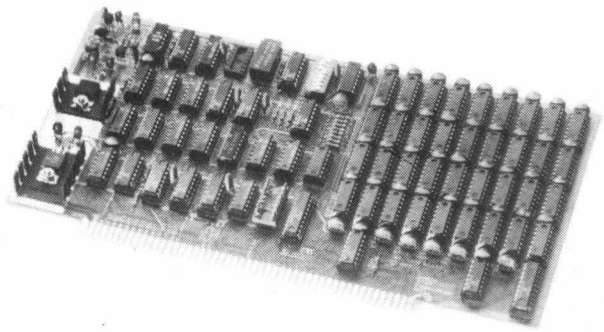


P Prints the entire program on the console device. This routine will print the entire program exactly as a regular PILOT command of DP: would.

R Reset the line pointer to the beginning of the program. The first line of the program in memory becomes the current line.

Z Reset the line pointer to the end (last line) of the program. The last line in the user program is made the current line.

NOTES: It is imperative that all programs end with a carriage return and a ctl/a (01 hex) character for the proper operation of the editor. When entering programs in PILOT this will automatically happen if the terminator character (ctl/z) is not typed *until* the last program line is ended with a carriage return. If you are entering programs in the editor, it will automatically take care of this operation.



16K RAM BY NORTH STAR

News Release *Received: 77 Dec 20*

North Star Computers, Inc, announces a new, 16K RAM board for S-100 bus computer systems.

North Star's 16K RAM is designed for use in *both* 8080 and Z-80 computer systems. It will operate at full speed (zero wait states) even at 4MHZ.

The low-power board uses 200 ns dynamic RAM chips and the on-board memory refresh is invisible to the processor. Bank switching capability is provided and the addressing of the board is switch-selectable in two 8K sections.

Another feature of the board is the availability of a parity check option. The North Star 16K RAM board is offered in kit form at $299 and fully assembled at $459. The parity option costs $38 in kit and $59 assembled. For further information, contact: North Star Computers, inc, 2547 Ninth St, Berkeley, CA 94710, (415) 549-0858.

```
                ;
                ;---PILOT EDITOR ROUTINES---
                ;
                ;     DAVID J. MORGAN
                ;  UNIV. TEXAS AT DALLAS
                ;     JAN. 1, 1978
                ;
0EEA 217911 EDTR:  LXI   H,EDMSG ;POINT TO EDITOR MSG
0EED CD490D        CALL  DSPLY   ;PRINT MSG
0EF0 210100        LXI   H,1     ;SET THE LINE COUNTER
0EF3 225211        SHLD  LINCT   ;  AND SAVE IT
0EF6 21A711        LXI   H,PBUFF ;POINT TO USER PGM START
0EF9 225011        SHLD  LINST   ;SAVE AS POINTER
0EFC 21A711 EDIT1: LXI   H,PBUFF ;POINT TO START USER AREA
0EFF 7E            MOV   A,M     ;GET 1ST BYTE
0F00 FE01          CPI   01H     ;CHECK FOR EMPTY
                                 ;(01H IN 1ST LOC MEANS NO USER PGM IN MEMORY)
0F02 CA0E0F        JZ    MPRT    ;GO OUT AND ACCEPT ONLY INSERT (& END COMMANDS)
0F05 CD530F        CALL  GCOM    ;GET COMMAND
0F08 CD660F        CALL  EDIT2   ;PROCESS COMMAND
0F0B C3FC0E        JMP   EDIT1   ;DO MORE
                ;
                ;  GET COMND WHEN PGM AREA EMPTY
                ;
0F0E 21230F MPRT:  LXI   H,MSGL  ;PRINT THAT ONLY
0F11 CD490D        CALL  DSPLY   ;  2 COMMANDS ARE ALLOWED
0F14 CD530F        CALL  GCOM    ;GET A COMMAND
0F17 CD450F        CALL  PCOM    ;PROCESS IT
0F1A 210100        LXI   H,1     ;MAKE SURE LINE CNTR
0F1D 225211        SHLD  LINCT   ;  IS CORRECT
0F20 C3FC0E        JMP   EDIT1   ;GO ON
0F23 50474D20 MSGL: DB    'PGM SPACE EMPTY--ENTER `I` OR `E`'
0F27 53504143
0F2B 4520454D
0F2F 5054592D
0F33 2D454E54
0F37 45522060
0F3B 4960204F
0F3F 52206045
0F43 60
0F44 0D                DB    0DH
                ;
                ;  PROCESS CMND WHEN PGM AREA EMPTY
                ;
0F45 FE45  PCOM:   CPI   'E'     ;CHECK FOR END CMND
0F47 CA1611        JZ    EDRTR   ;END--EXIT EDITOR
0F4A FE49          CPI   'I'     ;CHECK FOR INSERT CMND
0F4C CA8110        JZ    INSTX   ;SPECIAL FIRST INPUT ROUTINE
0F4F E1            POP   H       ;CLEAR STACK AND RET
0F50 C30E0F        JMP   MPRT    ;GO START OVER IF NOT E OR I
                ;
                ;  GET EDITOR COMMAND
                ;
0F53 218C11 GCOM:  LXI   H,RCOM  ;ASK FOR COMMAND
0F56 CD490D        CALL  DSPLY   ;  FROM THE TERMINAL
0F59 CDCE0D        CALL  CHI     ;GET THE COMMAND
0F5C 324F11        STA   EDCOM   ;SAVE EDITOR COMMAND
0F5F CDC40C        CALL  CRLF    ;DO LINE FEED-CAR RET
0F62 3A4F11        LDA   EDCOM   ;GET EDITOR COMMAND BACK AGAIN
0F65 C9            RET
                ;
                ;  PROCESS EDITOR COMMAND
                ;
0F66 FE45  EDIT2:  CPI   'E'     ;END COMMAND
0F68 CA1611        JZ    EDRTR   ;RETURN
0F6B FE50          CPI   'P'     ;PRINT ENTIRE PROGRAM
0F6D CA7A0D        JZ    DPRG    ;GO TO REGULAR PROGRAM FOR THIS
0F70 FE4E          CPI   'N'     ;EXAMINE NEXT LINE
0F72 CABB10        JZ    NEXT
0F75 FE42          CPI   'B'     ;BACKUP AND EXAMINE
0F77 CAE210        JZ    BACK
0F7A FE43          CPI   'C'     ;PRINT CURRENT LINE
0F7C CA8B0F        JZ    CURPR
0F7F FE44          CPI   'D'     ;DELETE A LINE
0F81 CAAF0F        JZ    KILL
0F84 FE49          CPI   'I'     ;INSERT A LINE
0F86 CAFC0F        JZ    INST
0F89 FE52          CPI   'R'     ;RESET EDITOR TO FIRST LINE
0F8B CA9A0F        JZ    RESET
0F8E FE5A          CPI   'Z'     ;MAKE LAST LINE THE CURRENT LINE
0F90 CA2E11        JZ    LASTL
0F93 218E11        LXI   H,EDMS1 ;PRINT BAD COMMAND MSG
0F96 CD490D        CALL  DSPLY
0F99 C9            RET           ;GO BACK AND TRY AGAIN
```

```
                        ;
                        ;  RESET SYSTEM TO FIRST LINE
                        ;
0F9A 210100     RESET:  LXI     H,1     ;RESET LINE COUNTER
0F9D 225211             SHLD    LINCT   ;   AND RESAVE IT
0FA0 EB                 XCHG            ;SAVE FOR CALLING ROUTINE IF NEEDED
0FA1 21A711             LXI     H,PBUFF ;RESET BUFFER POINTER
0FA4 225011             SHLD    LINST   ;   AND RESAVE IT TOO
0FA7 C9                 RET
                        ;
                        ;  PRINT CURRENT TEXT LINE
                        ;
0FA8 2A5011     CURPR:  LHLD    LINST   ;SET POINTER TO CURRENT LINE
0FAB CD490D             CALL    DSPLY   ;   AND GO PRINT IT
0FAE C9                 RET
                        ;
                        ;  DELETE THE CURRENT LINE
                        ;
0FAF 2A5211     KILL:   LHLD    LINCT   ;GET LINE COUNT
0FB2 3E00               MVI     A,0     ;CLEAR SWITCH
0FB4 4F                 MOV     C,A     ;SAVE IT IN C AND THEN SEE IF
0FB5 BC                 CMP     H       ;   WE ARE AT THE
0FB6 C2BF0F             JNZ     KILL3   ;   FIRST LINE OF
0FB9 3C                 INR     A       ;   THE PGM
0FBA BD                 CMP     L       ;   AREA
0FBB C2BF0F             JNZ     KILL3   ;IF NOT,GO ON
0FBE 4F                 MOV     C,A     ;IF SO, SET FLAG IN C
0FBF 2A5011     KILL3:  LHLD    LINST   ;GET CURRENT LINE
0FC2 EB                 XCHG
0FC3 CDEE0F             CALL    NXTAD   ;GET NEXT LINE ADDR
0FC6 7E                 MOV     A,M     ;GET THAT BYTE
0FC7 FE01               CPI     1       ;SEE IF EOF
0FC9 C2E40F             JNZ     KILL1   ;NO, MOVE EVERYTHING UP
0FCC B9                 CMP     C       ;NOW SEE IF WE ARE ON LINE ONE
0FCD C2D60F             JNZ     KILL4   ;IF NOT, GO ON
0FD0 32A711             STA     PBUFF   ;IF SO, PUT 01H IN PBUFF TO CLEAR
0FD3 C3ED0F             JMP     KIRET   ;AND GET OUT
0FD6 12         KILL4:  STAX    D       ;PUT EOF IN CURRENT LINE
0FD7 2A5211             LHLD    LINCT   ;THEN BACKUP THE LINE
0FDA 2B                 DCX     H       ;   COUNTER AND
0FDB 225211             SHLD    LINCT   ;   RESAVE IT.  ALSO,
0FDE CD1E11             CALL    BKADR   ;   THEN MUST RESET CURRENT
0FE1 C3ED0F             JMP     KIRET   ;   ADDR TO PRIOR ADDR
0FE4 7E         KILL1:  MOV     A,M     ;GET BYTE
0FE5 12                 STAX    D       ;SAVE BYTE
0FE6 23                 INX     H       ;KICK UP THE POINTERS
0FE7 13                 INX     D
0FE8 FE01               CPI     1       ;CHECK FOR END
0FEA C2E40F             JNZ     KILL1   ;IF NOT, MOVE MORE
0FED C9         KIRET:  RET
                        ;
                        ;  FIND ADDR OF LINE FOLLOWING CURRENT LINE
                        ;
0FEE 2A5011     NXTAD:  LHLD    LINST   ;GET START ADDR OF CURRENT LINE
0FF1 7E         NXTA1:  MOV     A,M     ;GET BYTE
0FF2 FE0D               CPI     0DH     ;SEE IF END
0FF4 23                 INX     H       ;KICK UP
0FF5 C2F10F             JNZ     NXTA1   ;GO BACK IF NOT END
0FF8 22B610             SHLD    TEMX    ;SAVE NEXT LINE ADDR
0FFB C9                 RET
                        ;
                        ; INSERT A NEW LINE OF TEXT
                        ;
0FFC CDEE0F     INST:   CALL    NXTAD   ;GET ADDR OF NEXT LINE
0FFF 0E3D               MVI     C,'='   ;GET A NEW LINE (PROMPT IS =)
1001 CDA40D             CALL    CHO     ;   AND PRT IT
1004 CD650C             CALL    KEYIN   ;GO TO INPUT ROUTINE
1007 3A4401             LDA     CHMAX   ;GET MAX COUNT
100A 90                 SUB     B       ;SUB CNT FOR ACTUAL COUNT
100B 3C                 INR     A       ;ADD 1 FOR CR CHAR
100C 32B810             STA     CCNT    ;AND SAVE IT
100F CD7510             CALL    EMEM    ;FIND END OF CURRENT PGM
1012 22B410             SHLD    CEPTR   ;SAVE CURRENT END PTR
                        ;--(H-L NOW POINTS TO END OF PGM)--
1015 3AB810             LDA     CCNT    ;GET CHAR COUNT AGAIN
1018 5F                 MOV     E,A     ;SET UP DE WITH LENGTH
1019 1600               MVI     D,0     ;   OF NEW LINE
101B 19                 DAD     D       ;   AND ADD IT TO CURRENT END PGM
                                        ;   INDICATOR TO MAKE SURE IT FITS
101C 22B910             SHLD    EOPTR   ;SAVE AS END PGM PTR
101F 11FF3F             LXI     D,PBUFE ;   SEE NOT BEYOND
1022 7A                 MOV     A,D     ;   END OF MEMORY
1023 BC                 CMP     H       ;FIRST CHECK D TO H
1024 DA2F10             JC      IERR    ;NOPE-NO ROOM TO ADD
1027 C23810             JNZ     IOKAY   ;ITS OKAY
102A 7B                 MOV     A,E     ;NOW CHECK E-L
102B BD                 CMP     L
102C D23810             JNC     IOKAY   ;ITS ALRIGHT
102F 219D10     IERR:   LXI     H,OSMSG ;OUT OF SPACE MSG
1032 CD490D             CALL    DSPLY
1035 C37410             JMP     IRET    ;GET OUT
1038 2AB910     IOKAY:  LHLD    EOPTR   ;GET END PTR
103B 3E01               MVI     A,1     ;GET END FLAG
103D 77                 MOV     M,A     ;AND SAVE IT
103E EB                 XCHG            ;PUT IN DE
103F 2AB610             LHLD    TEMX    ;GET INSERT ADDR
1042 4E                 MOV     C,M     ;SAVE BYTE THERE
1043 B9                 CMP     C       ;CHECK IF AT END
1044 77                 MOV     M,A     ;SAVE END FLAG
1045 2AB410             LHLD    CEPTR   ;GET CURRENT END
1048 EB                 XCHG            ;SAVE IN DE
1049 CA5910             JZ      INST6   ;IF ADD TO END, SKIP MOVE UP OF DATA
104C 2B         INST3:  DCX     H       ;KICK PTRS
104D 1B                 DCX     D
104E 1A                 LDAX    D       ;GET BYTE
104F FE01               CPI     01      ;SEE IF END YET
1051 CA5810             JZ      INST4   ;QUIT IF SO
1054 77                 MOV     M,A     ;SAVE IT
1055 C34C10             JMP     INST3   ;DO MORE IF NOT DONE
1058 71         INST4:  MOV     M,C     ;SAVE ORIGINAL BYTE
                        ;   (MOVE IN THE NEW LINE NOW)
1059 3AB810     INST6:  LDA     CCNT    ;GET COUNT FOR TRANSFER
105C 4F                 MOV     C,A
105D 2AB610             LHLD    TEMX    ;POINT TO DEPOSIT ADDR
1060 EB                 XCHG            ;SWITCH
1061 219D01             LXI     H,EBUFF ;POINT TO INPT BUF WHERE NEW LINE IS
1064 CD540C             CALL    BLKTFR  ;MOVE IN NEW LINE FROM INPUT BUF
                        ;   (NOW UPDATE THE CURRENT LINE POINTER)
1067 2A5211             LHLD    LINCT   ;GET COUNT
106A 23                 INX     H       ;KICK IT ONE
106B 225211             SHLD    LINCT   ;SAVE COUNT
106E 2AB610             LHLD    TEMX    ;GET INSERT ADDR
1071 225011             SHLD    LINST   ;SAVE AS NEW ADDR
1074 C9         IRET:   RET
                        ;
                        ;   FIND END OF CURRENT PROGRAM
                        ;
1075 21A711     EMEM:   LXI     H,PBUFF ;POINT TO START OF USER PGM
1078 7E         EMEM1:  MOV     A,M     ;GET BYTE
1079 23                 INX     H       ;KICK COUNTER
107A FE01               CPI     01H     ;CHECK FOR END
107C C27810             JNZ     EMEM1   ;RETURN FOR MORE IF NOT
107F 2B                 DCX     H       ;KICK BACK AND
1080 C9                 RET             ;OUT
                        ;
                        ;  INSERT WHEN PGM SPACE EMPTY
                        ;
1081 0E3D       INSTX:  MVI     C,'='   ;PUT OUT PROMPT
1083 CDA40D             CALL    CHO
1086 CD650C             CALL    KEYIN   ;GET THE NEW PROGRAM LINE
1089 3A4401             LDA     CHMAX   ;GET MAX
108C 90                 SUB     B       ;AND FIND ACTUAL LEN
108D 3C                 INR     A       ;PLUS ONE FOR CR CHAR
108E 4F                 MOV     C,A     ;SAVE FOR TFR
108F 21A711             LXI     H,PBUFF ;LOAD DESTINATION ADDR
1092 EB                 XCHG
1093 219D01             LXI     H,EBUFF ;LOAD ORIGIN ADDR
1096 CD540C             CALL    BLKTFR  ;MOVE IT
1099 3E01               MVI     A,01    ;SET END OF PGM FLAG
109B 12                 STAX    D       ;SAVE IT AT END OF DATA
109C C9                 RET
109D 2D2D5052 OSMSG:    DB      '--PROGRAM SPACE FULL--',0DH
10A1 4F475241
10A5 4D205350
10A9 41434520
10AD 46554C4C
10B1 2D2D0D
10B4          CEPTR:    DS      2
10B6          TEMX:     DS      2
10B8          CCNT:     DS      1
10B9          EOPTR:    DS      2
```

```
                        ;
                        ;  PRINT NEXT PGM LINE
                        ;
10BB 2A5011     NEXT:   LHLD    LINST   ;GET START OF CURRENT LINE
                        ;       (FIRST GET START OF NEXT LINE IF ONE EXISTS)
10BE 7E         NEXT1:  MOV     A,M     ;GET 1ST BYTE
10BF FE0D               CPI     0DH     ;CHECK FOR END
10C1 23                 INX     H       ;KICK POINTER
10C2 C2BE10             JNZ     NEXT1   ;MORE IF NOT END
10C5 7E                 MOV     A,M     ;GET NEXT BYTE
10C6 FE01               CPI     1       ;MAKE SURE NOT EOF
10C8 C2D410             JNZ     NEXT2   ;GO ON IF NOT EOF
10CB 215411             LXI     H,EOFMS ;PRINT END MSG
10CE CD490D             CALL    DSPLY
10D1 C3E110             JMP     NEXRT
                        ;       (NOW UPDATE POINTER AND PRINT NEXT LINE)
10D4 225011     NEXT2:  SHLD    LINST   ;SAVE POINTER
10D7 CD490D             CALL    DSPLY   ;PRINT
10DA 2A5211             LHLD    LINCT   ;GET COUNTER
10DD 23                 INX     H       ;KICK IT
10DE 225211             SHLD    LINCT   ;AND SAVE IT
10E1 C9         NEXRT:  RET
                        ;
                        ;  PRINT PREVIOUS TEXT LINE
                        ;
10E2 2A5211     BACK:   LHLD    LINCT   ;FIRST CHECK IF ANY BACKUP ALLOWED
10E5 97                 SUB     A       ;CLEAR ACC
10E6 BC                 CMP     H       ;SEE IF ZERO
10E7 C20B11             JNZ     BACK1   ;GO ON IF NOT
10EA 7D                 MOV     A,L     ;GET LOW ORDER AND SEE
10EB FE01               CPI     1       ;IF ITS ZIPO
10ED C20B11             JNZ     BACK1   ;GO ON IF NOT
10F0 21F910             LXI     H,SMSG  ;PRT MSG THAT
10F3 CD490D             CALL    DSPLY   ;CAN'T BACK UP
10F6 C31511             JMP     BCKRT   ;AND RET
10F9 2D2D5354   SMSG:   DB      '--START OF FILE--',0DH
10FD 41525420
1101 4F462046
1105 494C452D
1109 2D0D
110B 2B         BACK1:  DCX     H       ;KICK COUNTER
110C 225211             SHLD    LINCT   ;  AND SAVE IT AS ONE LESS THAN
110F CD1E11             CALL    BKADR   ;GET ADDR OF START OF PREVIOUS LINE
1112 CD490D             CALL    DSPLY   ;PRINT
1115 C9         BCKRT:  RET
                        ;
                        ;  EXIT FROM THE EDITOR
                        ;
1116 216711     EDRTR:  LXI     H,RMSG  ;PRINT RETURN MSG
1119 CD490D             CALL    DSPLY
111C E1                 POP     H       ;CLR STACK FOR DIR RET
111D C9                 RET
                        ;
                        ;  GET PREVIOUS LINE ADDR
                        ;
111E 2A5011     BKADR:  LHLD    LINST   ;GET CURRENT ADDR
1121 2B                 DCX     H       ;SKIP CR CHAR
1122 2B                 DCX     H       ;KICK
1123 7E         BKAD1:  MOV     A,M     ;GET CHAR
1124 FE0D               CPI     0DH     ;SEE IF CR OF PREV FIELD
1126 C22211             JNZ     BKAD1   ;GO ON IF NOT
1129 23                 INX     H       ;BACK UP TO NEXT LINE
112A 225011             SHLD    LINST   ;SAVE AGAIN
112D C9                 RET
                        ;
                        ;  MOVE POINTER TO LAST LINE IN USER PGM
                        ;
112E 2A5211     LASTL:  LHLD    LINCT   ;GET LINE CTR
1131 EB                 XCHG            ;SAVE IN DE
1132 2A5011             LHLD    LINST   ;GET CURRENT LINE ADDR
1135 7E         LAST2:  MOV     A,M     ;GET CHAR
1136 23                 INX     H       ;BUMP PTR
1137 FE0D               CPI     0DH     ;CHECK FOR CR
1139 C23D11             JNZ     LAST1   ;GO ON IF NOT
113C 13                 INX     D       ;KICK THE COUNTER
113D FE01       LAST1:  CPI     01H     ;CHECK FOR END
113F C23511             JNZ     LAST2   ;DO SOME MORE
1142 2B                 DCX     H       ;KICK BACK TO PROPER ADDR
1143 225011             SHLD    LINST   ;SAVE AS ADDR
1146 CD1E11             CALL    BKADR   ;BACKUP TO LAST START ADDR
1149 1B                 DCX     D       ;KICK DOWN LINE COUNTER TO
114A EB                 XCHG            ;PROPER VALUE AND SAVE
114B 225211             SHLD    LINCT
114E C9                 RET
114F            EDCOM:  DS      1       ;EDITOR COMMAND
1150            LINST:  DS      2       ;LINE TO PRINT
1152            LINCT:  DS      2       ;LINE COUNTER
1154 2D2D454E   EOFMS:  DB      '--END OF PROGRAM--',0DH
1158 44204F46
115C 2050524F
1160 4752414D
1164 2D2D0D
1167 2E2E2E4C   RMSG:   DB      '...LEAVING EDITOR',0DH
116B 45415649
116F 4E472045
1173 4449544F
1177 520D
1179 50494C4F   EUMSG:  DB      'PILOT EDITOR READY',0DH
117D 54204544
1181 49544F52
1185 20524541
1189 44590D
118C 2A01       RCOM:   DB      '*',01H
118E 2A2A2A49   EOMS1:  DB      '***INVALID EDIT COMMAND',0DH
1192 4E56414C
1196 49442045
119A 44495420
119E 434F4D4D
11A2 414E440D
11A6 0D                 DB      0DH     ;END OF PBUFF AREA (EDITOR CHECKPOINT)
11A7            PBUFF:  ORG     PBUFB
11A7 01                 DB      1
0000                    END
```

TSC 6800 TEXT PROCESSOR

News Release *Received: 77 Dec 15*

Technical Systems Consultants, Inc. is pleased to announce the availability of the TSC Text Processing System. This program allows the use of over 50 special text formatting commands. The commands included will support multiple spacing, left margin control, indenting, the ability to save contiguous space, save contiguous text, paging, right hand justification, left hand justification, left and right justification, centering, no-fill modes, page numbering, the printing of left hand, right hand, or centered titles, and line length control.

Also included are capabilities for macro definition to define and build special formatting commands, number registers which can be used like variables in a program, conditional command execution, setable macro execution points (to execute a macro at a pre-defined line number), text diversion for later use (such as footnote processing), the ability to prompt a terminal for text during the formatting process, and a feature which allows sending informative strings to the terminal.

The TSC Text Processor will also output numbers in either Arabic, capital Roman numerals, or small Romal numerals. Tab columns may be defined as well as the tab character and the tab fill character. Environment switching is permitted for easy parameter changing and a loop command is available for repeated formatting jobs such as form letters.

TSC recommends the TSC Text Editing System to be used in conjunction with the processor. These two packages will give the user all the powers of the most complete word processing system. As with all TSC software, the complete commented assembler language source listing is provided as well as a very extensive user's manual which includes an "Introduction to Text Processing". Price of the SL68-29 Manual and Source Listing is $32, with cassette $38, and $40 with paper tape. Contact Technical Systems Consultants, Box 2574, W Lafayette, IN 47906.

NEW GRAPHICS BOARD FOR SOL

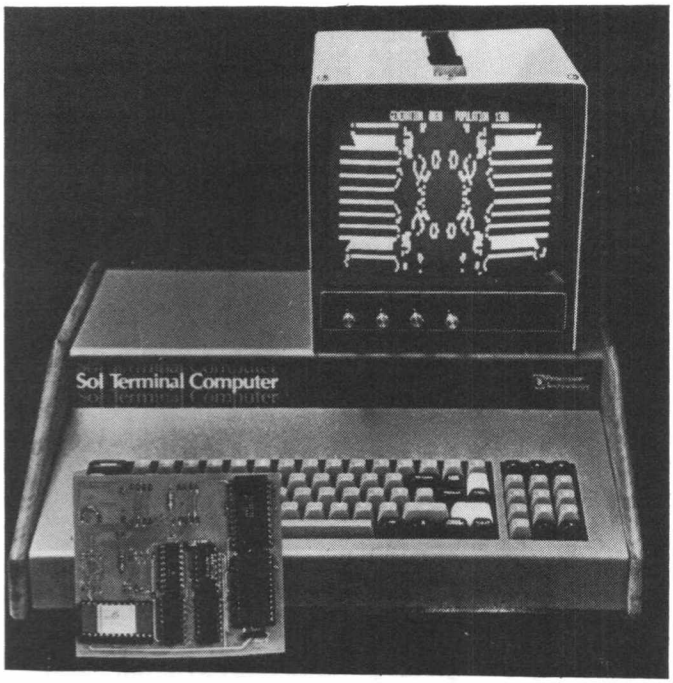
News Release *Received: 77 Nov 15*

KEA Micro Design of Toronto has annouced the availability of GraphicAdd, a piggyback board for use in SOL computers and VDM-1 Video Display Modules. GraphicAdd gives graphics capability to these units (128H by 48V) and includes a SOL ready-to-load software package.

The GraphicAdd board mounts directly on the VDM and SOL main boards. It works by replacing half of the inverse video character set by bit-mapped graphic cells. Thus alpha-numerics and graphics can be mixed on the same line. Mode control options include fixed graphics, switch-selectable graphics, or programmable graphics mode.

The software package contains a Graphics Driver, BASIC Links, LIFE, and demonstration programs. The Graphics Driver maps screen memory locations to H and V coordinates with subroutines to designate black or white cells. BASIC Links provide parameter passing to permit X, Y plotting directly from BASIC.

GraphicAdd comes in kit form with all parts including sockets and IC's. Fully documented hardware and software manuals (with all source listings) are included. Introductory price for the GraphicAdd kit is $50. It is available from SOL dealers or directly from Micro-Ware Limited, 27 Firstbrooke Rd., Toronto, Ont., Canada, M4E 2L2..



DALLAS MICROCOMPUTER EXPOSITION

News Release *Received: 78 Feb 22*

The International Microcomputer Exposition will be held in the Dallas, Texas Convention Center Sept. 29 through Oct. 1. Co-sponsored by several groups, including the American Association of Microprocessors, the Exposition will be directed toward all levels of technology from the professional engineer to the beginning computer hobbyist. In addition to the seminars, a panel of experts will be available to answer questions. Advance registrations indicate attendance in excess of 15,000 to view some 250 exhibits. Further information is available from Beverly Tanner; (214) 271-9311.

EPA's LOW COST MINIPRINTER

news release *Received: 78 Jan 3*

San Diego—Electronic Product Associates, Inc. announces the availability of the MP-44 MINIPRINTER. It is a simple and inexpensive 5 x 7 dot matrix printer for microcomputer systems. Electrosensitive paper is used to make copies at speeds up to 88 characters per second with 44 characters per line. Software control allows expansion of character size for emphasis. Black characters are printed on aluminized paper 6 centimeters (2 3/8 inches) wide.

An enclosure 4½ by 8 3/4 inches houses the printer and paper supply mounted on an interface board with all necessary components for connection to any microcomputer with TTL logic levels. The microcomputer controls the motor and print electrodes by an eight bit parallel output word, and senses timing signals on two input lines. A power supply of 40 volts dc at one amp must be provided to the interface board.

Low cost is achieved by using the microcomputer for software control of all writing and timing functions. The software is explained in a structured form which can be easily implemented in any computer language, and examples of complete assembly language coding are provided for microcomputers using the 6800 and the 8080 microprocessors. Coding for the 6800 allows the printer to be operated immediately from the EPA Micro-68 I/O bus. Cost of this MP-44 MINIPRINTER is $257.00 and is available from stock. For additional information please contact: EPA, 1157 Vega Street, San Diego, CA 92110 (714) 276-8911.

MUG MEETING IN SAN FRANCISCO

News Release *Received: 78 Feb 23*

MUMPS Users' Group (MUG), San Francisco, California, June 7-9, 1978 Jack Tar Hotel. Conference includes all-day tutorials on programming in MUMPS, ambulatory care information systems, and technical aspects of ANSI-Stanford MUMPS. Formal presentations of medical and non-medical application; panel and discussions on applications on computers in medicine. Conference is the same week as NCC in Anaheim, California. Information: Ms. Pat Zimmerman, Department of Biometry, Wearn Research Building, University Hospitals, Cleveland, Ohio 44106.

ACT

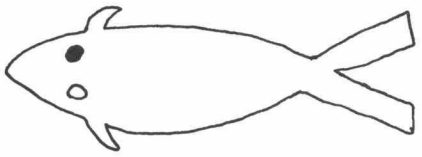
In the last issue of Dr. Dobb's *we published the documentation and PL/M source listing for the ACT 8080 Macroprocessor by Alex Cecil of Lawrence Livermore Laboratory. Here now is the symbol table and the assembler listing. The table preceding the assembler listing references selected line numbers in the PL/M version to the assembler listing. –TRW*

BY ALEX CECIL
Lawrence Livermore Laboratory

An 8080 Macroprocessor

```
S00986 18
S00984 5E
S00983 17
S00981 25
S00979 PRESETDRIVES
S00974 PRESETATIN
S00973 CH
S00903 7473
S00902 7366
S00901 7A7A
S00900 7372
S00899 6C63
S00898 6E76
S00897 7766
S00896 6674
S00895 6E74
S00894 626E
S00893 7865
S00892 6F69
S00891 6F72
S00890 6F73
S00889 7370
S00888 6B75
S00887 646D
S00886 6472
S00885 6664
S00884 666D
S00883 6E72
S00882 6663
S00881 666F
S00880 6677
S00879 6672
S00878 7664
S00877 6C6D
S00876 7573
S00875 6464
S00874 7361
S00873 6461
S00872 6B70
S00871 7373
S00870 6367
S00869 7263
S00868 6567
S00867 746E
S00866 6E69
S00865 7165
S00864 736E
S00863 PFLIST
S00862 I
S00860 CHCH
S00859 PFINDEX
S00858 SECONDCH
S00857 FIRSTCH
S00856 CH
S00855 BLOCK
S00854 BLOCKPTR
S00853 NAMECH
S00852 NAME
S00851 NAMEPTR
S00849 PROCESSENDFUNCTION
S00847 PFRD
S00840 RC
S00838 PFRF
S00830 21
S00826 STUFFCH
S00825 WRITESTATUS
S00824 I
S00823 CH
S00821 PFWF
S00820 19
S00818 PFDF
S00815 PFCF
S00813 22
S00811 PFMF
S00808 PFOF
S00806 PFRN
S00805 12
S00803 RECORDNO
S00802 SETFILERECORD
S00800 8
S00796 ':'
S00791 DRIVE
S00790 CH
S00789 I
S00787 FILENAMEPTR
S00786 SETFILENAME
S00785 '1'
S00784 '3'
S00782 PFVN
S00780 PFFS
S00777 N
S00776 BLOCK
S00774 PFNB
S00772 PFDV
S00770 64
S00759 CH
S00757 PFIO
S00756 N
S00754 PFZZ
S00752 PFMD
S00749 PRODUCT
S00747 PFML
S00745 PFSU
S00742 SUM
S00740 PFAD
S00735 PFDD
S00730 ENDPTR
S00728 PFRO
S00726 PFSO
S00722 CH
S00720 PFUK
S00716 CHB
S00715 CHA
S00713 PFPK
S00711 ' '
S00708 60
S00698 1A
S00696 'DISK EOF  $'
S00694 20
S00691 NEXTCHAR
S00690 NOTCOMMENT
S00689 CH
S00688 K
S00686 PFRS
S00684 PFFW
S00680 NOTFIRST
S00678 PFPS
S00676 PFEX
S00665 SEGNUMBER
S00664 CHB
S00663 CH
S00662 LASTARGNUMBER
S00660 STACKOFFSET
S00659 PFCL
S00655 PFNT
S00653 8000
S00651 PFGE
S00644 ARGC
S00643 ARGB
S00642 ARGA
S00640 PFEQ
S00637 PFCR
S00625 CH
S00624 INPTR
S00622 PFIN
S00610 M
S00608 GCLENGTH
S00604 ANOTHERCH
S00603 NEGCOUNTFAILED
S00602 CH
S00601 BACKUPTO
S00600 CURRENTCHPTR
S00599 FORMPTR
S00598 MARKPTR
S00597 N
S00596 NPTR
S00595 GCPTR
S00593 PFGC
S00590 11000000
S00583 CURRENTBLOCK
S00582 CURRENTBLOCKPTR
S00581 CH
S00579 CHA
S00578 PUTCHASS
S00577 MOREFILL
S00576 ARGNUMBER
S00574 PFSS
S00566 CH
S00564 PFAS
S00557 PFNS
S00544 NEXTSTART
S00533 MATCHHERE
S00532 RESTARTPTR
S00531 CHB
S00530 CHA
S00528 SSMATCH
S00525 N
S00523 NUMBEROFARGS
S00520 HALFBYTE
S00519 HEXTOCH
S00518 57
S00517 37
S00515 47
S00513 '9'
S00511 CH
S00510 CHTOHEX
S00503 NEGATIVE
S00502 CH
S00501 N
S00499 BLOCKPTR
S00498 STRINGTOBINARY
S00495 '-'
S00491 '0'
S00487 NEGATIVE
S00486 K
S00485 I
S00484 D
S00482 N
S00481 BINARYTOSTRING
S00474 CH
S00473 BLOCK
S00472 BLOCKPTR
S00470 PROCESSBEGINFUNCTION
S00460 CH
S00459 LASTDELIMWASLEFT
S00458 DELIMLEVEL
S00456 PROCESSQUOTEDSTRING
S00452 BLOCK
S00451 BLOCKPTR
S00449 CH
S00448 NSADDCH
S00446 'NS STACK OVERFLOW  $'
S00443 FLAGS
S00442 STARTNEXTARGUMENT
S00435 NEXT
S00434 OLDBLOCKPTR
S00433 CH
S00431 MOVEPTRFLAG
S00430 ASNEXTCH
S00428 ASFREEBLOCK
S00421 I
S00420 CLIM
S00417 ASNSINIT
S00411 74
S00409 ' T N I:  $'
```

```
S00407 1B
S00405 CH
S00403 PROCESSCONBREAK
S00401 TURNTRACEOFF
S00399 TURNTRACEON
S00394 TRACEPRINT
S00392 NEXTNS
S00391 THIS
S00389 THISNS
S00386 6D
S00385 7D
S00384 ','
S00383 7C
S00382 73
S00381 70
S00380 7B
S00379 IM
S00376 STREINIT
S00374 STVALUEPTR
S00373 STADDVALUE
S00371 NAMEPTR
S00370 STCREATE
S00368 STDELETE
S00363 11111
S00360 CH
S00359 HASH
S00357 NAMEPTR
S00356 STFIND
S00352 STINIT
S00351 STPREVIOUS
S00350 STPREVIOUSPTR
S00349 STENTRY
S00348 STENTRYPTR
S00347 HASHINDEX
S00346 HASHTABLE
S00342 5000
S00330 MATCHEND
S00329 CHA
S00328 I
S00327 COUNTMAX
S00326 COUNT
S00325 RESTARTPTR
S00323 ENDMESSAGEPTR
S00322 RECEIVESTRING
S00318 CH
S00316 BLOCKPTR
S00315 SENDSTRING
S00311 CLEAROCTOUTPUT
S00305 CH
S00303 BLOCKPTR
S00302 PRINTSTRING
S00297 CH
S00295 BLOCKPTR
S00294 NULLTEST
S00287 NEXTB
S00283 NEXTA
S00282 CHB
S00281 CHA
S00279 PTB
S00278 PTA
S00277 STRINGEQUALITY
S00273 CH
S00271 SINKPTR
S00270 SOURCEPTR
S00269 COPYVALUE
S00266 NEWCHAIN
S00265 CH
S00264 CURRENTBLOCK
S00262 CHA
S00261 INITIALBLOCKPTR
S00260 PUTCHA
S00257 NEWCHAIN
S00256 CH
S00255 CURRENTBLOCK
S00254 CHPTR
S00253 CURRENTBLOCKPTR
S00251 INITIALBLOCKPTR
S00250 GETCHB
S00247 NEWCHAIN
S00246 CH
S00245 CURRENTBLOCK
S00244 CURRENTBLOCKPTR
S00242 INITIALBLOCKPTR
S00241 GETCHA
S00239 OF
S00233 BLOCKC
S00232 CHPTR
S00231 BLOCK
S00229 BLOCKPTR
S00228 CH
S00227 APPENDCH
S00222 OFFFO
S00221 NEXTPTR
S00220 BLOCK
S00218 BLOCKPTR
S00217 FREECHAIN
S00213 16
S00211 BLKPTR
S00209 MAXADDRESS
S00208 MINADDRESS
S00207 FSINIT
S00206 8181
S00201 'FREE STORAGE IS EMPTY
S00199 FIRSTEMPTY
S00198 GOTC
S00197 GOT
S00196 GOTPTR
S00195 FSFIRST
S00193 FSGET
S00192 FREETHIS
S00190 FREETHISPTR
S00189 FSFREE
S00188 NOEXVALUE
S00187 DEADSTART
S00186 NULLSTRINGPTR
S00185 FPTR
S00184 SPTR
S00183 VALUECH
S00182 VALUEPTR
S00181 ARGPTR
S00180 NAMEPTR
S00179 TRACEFLAG
S00178 SSCHA
S00177 SSCHCPTR
S00176 SSCHBPTR
S00175 SSCHAPTR
S00174 ASCH
S00173 ASCHPTR
S00172 ASBLOCK
S00171 ASBLOCKPTR
S00170 NSACTIVEFLAG
S00169 NSARGBREAK
S00168 NSCH
S00167 NSCHPTR
S00166 NSSAVEINDEX
S00165 NSSTACKINDEX
S00164 11
S00163 NSSTACK
S00162 PUTCHACURRENTBLOCKPTR
S00161 PUTCHACHPTR
S00160 GETCHACHPTR
S00159 FIRSTPUTAFLAG
S00158 FIRSTGETBFLAG
S00157 FIRSTGETAFLAG
S00156 FSFIRSTPTR
S00155 7F
S00151 65000
S00149 I
S00147 RECEIVECH
S00141 CH
S00140 SENDCH
S00137 OCTOSTATUS
S00136 OCTOCHANNELA
S00135 135
S00131 I
S00129 N
S00128 WAITTENTH
S00125 I
S00123 N
S00122 WAITMS
S00121 20
S00119 7A
S00118 61
S00116 CH
S00115 UPPERCASE
S00114 14
S00112 7
S00108 SELECTDRIVE
S00107 9
S00105 MESSPTR
S00104 PRINTMESS
S00103 11
S00101 CONBREAK
S00099 READCH
S00095 CH
S00094 DISPLAYCH
S00090 CH
S00089 PRINTCH
S00088 0A
S00087 0D
```



```
S00086 2
S00084 CRLF
S00082 A
S00081 F
S00080 MON2
S00079 5
S00077 A
S00076 F
S00075 MON1
S00074 FBASE
S00073 6
S00072 FBASEA
S00071 ITEMP
S00070 DUMMY
S00069 TEMPPTR
S00068 CURRENTDRIVE
S00067 PREVIOUSDRIVE
S00066 DEFAULTDRIVE
S00065 PRELOADED
S00064 7E
S00063 METACHAR
S00062 04
S00061 OCTOEOM
S00060 TBUFFINDEX
S00059 TBUFF
S00058 80
S00057 TBUFFPTR
S00056 TFCB
S00055 5C
S00054 TFCBPTR
S00053 120
S00052 OCTOTIMEOUT
S00051 1
S00050 CONBUF
S00049 CONIN
S00048 80
S00047 CONCOLUMN
S00046 CONWIDTH
S00045 OFF
S00044 CPMPRINTFLAG
S00043 62
S00042 ATCB
S00040 6F
S00039 72
S00038 ATRO
S00036 6E
S00035 69
S00034 ATIN
S00032 81
S00031 63
S00030 67
S00029 40
S00028 0
S00027 ATOC
S00025 START
S00024 100
S00022 DEC
S00021 DOUBLE
S00020 MOVE
S00019 LAST
S00018 LENGTH
S00017 OUTPUT
S00016 INPUT
S00015 LOW
S00014 HIGH
S00013 TIME
S00012 SCR
S00011 SCL
S00010 SHR
S00009 SHL
S00008 ROR
S00007 ROL
S00006 STACKPTR
S00005 MEMORY
S00004 PARITY
S00003 SIGN
S00002 ZERO
S00001 CARRY
8080 PLM2 VERS 3.0
```

1=000003Q	18=000403Q	96=000406Q	97=000420Q	98=000426Q	99=000440Q
100=000446Q	144=000456Q	145=000461Q	146=000462Q	149=000472Q	150=000475Q
151=000476Q	154=000507Q	155=000520Q	157=000525Q	158=000526Q	160=000532Q
161=000542Q	162=000550Q	163=000560Q	164=000565Q	165=000572Q	166=000575Q
167=000572Q	168=000573Q	171=000577Q	172=000601Q	173=000607Q	174=000614Q
175=000617Q	177=000630Q	178=000631Q	181=000642Q	182=000643Q	185=000651Q
186=000662Q	187=000665Q	188=000666Q	191=000672Q	192=000677Q	193=000706Q
194=000711Q	195=000720Q	196=000724Q	197=000731Q	198=000742Q	199=000745Q
200=000756Q	201=000757Q	205=000763Q	206=000766Q	207=001003Q	208=001007Q
212=001015Q	214=001023Q	215=001072Q	216=001073Q	219=001101Q	220=001107Q
221=001126Q	222=001136Q	223=001150Q	224=001165Q	225=001202Q	226=001222Q
227=001223Q	244=001233Q	245=001237Q	246=001243Q	250=001247Q	252=001252Q
253=001261Q	254=001266Q	255=001271Q	256=001276Q	257=001301Q	258=001303Q
259=001304Q	263=001314Q	264=001333Q	265=001342Q	266=001355Q	267=001363Q
268=001375Q	269=001402Q	270=001405Q	333=001413Q	338=001427Q	339=001443Q
340=001444Q	353=001453Q	354=001461Q	355=001522Q	356=001532Q	357=001534Q
358=001542Q	359=001553Q	361=001556Q	363=001570Q	364=001576Q	365=001604Q
366=001620Q	367=001634Q	368=001650Q	369=001664Q	370=001700Q	371=001714Q
372=001730Q	373=001743Q	374=001744Q	379=001756Q	381=001770Q	382=002037Q
383=002046Q	384=002051Q	385=002052Q	388=002060Q	390=002072Q	391=002114Q
392=002111Q	393=002127Q	394=002132Q	395=002150Q	396=002163Q	397=002166Q
398=002167Q	402=002177Q	407=002204Q	408=002216Q	409=002224Q	410=002232Q
411=002236Q	412=002247Q	413=002257Q	414=002261Q	415=002272Q	416=002274Q
417=002301Q	419=002311Q	420=002326Q	421=002331Q	422=002355Q	423=002357Q
424=002366Q	431=002374Q	436=002404Q	438=002406Q	439=002420Q	440=002425Q
441=002427Q	442=002447Q	443=002463Q	444=002477Q	445=002515Q	446=002517Q
447=002524Q	453=002532Q	458=002542Q	460=002544Q	461=002556Q	462=002563Q
463=002565Q	464=002605Q	465=002613Q	466=002631Q	467=002633Q	468=002640Q
476=002650Q	481=002660Q	483=002662Q	484=002676Q	485=002706Q	486=002710Q
487=002720Q	488=002735Q	489=002740Q	490=002764Q	491=002766Q	492=002767Q
498=003001Q	503=003006Q	504=003034Q	505=003043Q	506=003056Q	507=003061Q
508=003062Q	512=003074Q	517=003077Q	518=003131Q	519=003133Q	523=003144Q
524=003165Q	525=003167Q	529=003174Q	530=003220Q	531=003234Q	532=003236Q
537=003244Q	541=003251Q	542=003262Q	543=003267Q	544=003320Q	545=003331Q
546=003334Q	551=003342Q	556=003347Q	557=003422Q	558=003434Q	559=003444Q
560=003453Q	561=003456Q	562=003465Q	564=003470Q	565=003471Q	571=003500Q
572=003503Q	573=003527Q	574=003530Q	578=003536Q	579=003541Q	581=003545Q
582=003567Q	583=003602Q	584=003605Q	585=003606Q	591=003614Q	597=003617Q
598=003624Q	599=003640Q	600=003672Q	601=003706Q	602=003715Q	603=003730Q
604=003743Q	605=003746Q	606=003754Q	607=003757Q	608=003772Q	609=004001Q
610=004005Q	611=004020Q	612=004022Q	614=004037Q	615=004042Q	616=004052Q
617=004055Q	618=004062Q	619=004067Q	620=004075Q	622=004100Q	623=004113Q
624=004115Q	625=004122Q	626=004200Q	627=004227Q	628=004236Q	629=004251Q
630=004266Q	631=004300Q	632=004313Q	633=004320Q	634=004322Q	661=004327Q
662=004364Q	663=004365Q	667=004373Q	672=004376Q	673=004403Q	674=004425Q
675=004430Q	676=004434Q	677=004440Q	678=004473Q	679=004475Q	680=004510Q
681=004512Q	682=004540Q	683=004552Q	684=004577Q	685=004602Q	691=004612Q
692=004620Q	693=004636Q	694=004644Q	695=004663Q	696=004664Q	700=004672Q
702=004675Q	703=004703Q	704=004720Q	705=004742Q	706=004765Q	707=005012Q
708=005022Q	709=005027Q	713=005035Q	715=005045Q	716=005053Q	717=005063Q
718=005111Q	719=005116Q	724=005121Q	725=005136Q	728=005143Q	729=005162Q
730=005173Q	731=005204Q	732=005215Q	733=005231Q	734=005242Q	736=005247Q
737=005252Q	738=005275Q	739=005317Q	740=005326Q	741=005336Q	742=005337Q
748=005353Q	749=005355Q	750=005370Q	751=005376Q	754=005403Q	755=005416Q
762=005421Q	763=005433Q	764=005436Q	765=005444Q	766=005445Q	767=005454Q
768=005461Q	769=005464Q	770=005465Q	772=005472Q	773=005474Q	774=005475Q
775=005503Q	777=005511Q	778=005512Q	779=005513Q	783=005521Q	784=005534Q
785=005541Q	786=005601Q	787=005612Q	788=005620Q	789=005624Q	790=005644Q
791=005660Q	792=005663Q	793=005674Q	794=005677Q	795=005710Q	796=005716Q
797=005721Q	798=005722Q	807=005725Q	808=005733Q	811=005744Q	812=005747Q
813=005755Q	814=005764Q	815=005771Q	816=006007Q	817=006031Q	818=006040Q
819=006043Q	820=006060Q	821=006064Q	822=006074Q	825=006076Q	826=006103Q
827=006121Q	828=006130Q	829=006133Q	830=006137Q	831=006140Q	834=006154Q
835=006162Q	836=006201Q	837=006211Q	838=006214Q	844=006220Q	849=006223Q
850=006236Q	851=006241Q	852=006256Q	853=006264Q	855=006270Q	856=006271Q
859=006274Q	860=006317Q	861=006325Q	862=006332Q	863=006334Q	864=006336Q
865=006343Q	870=006347Q	872=006354Q	873=006405Q	874=006443Q	875=006445Q
876=006464Q	877=006523Q	878=006526Q	879=006527Q	884=006533Q	888=006543Q
889=006556Q	890=006577Q	891=006604Q	892=006612Q	893=006622Q	894=006637Q
895=006653Q	896=006664Q	897=006710Q	898=006712Q	899=006713Q	905=006720Q
906=006723Q	907=006730Q	908=006741Q	909=006751Q	910=006755Q	911=006766Q
912=006774Q	913=007004Q	914=007003Q	915=007014Q	916=007011Q	917=007017Q
918=007022Q	919=007023Q	925=007030Q	926=007041Q	927=007046Q	928=007070Q
929=007075Q	930=007106Q	931=007111Q	932=007116Q	933=007123Q	934=007126Q
935=007134Q	936=007141Q	937=007155Q	938=007160Q	939=007165Q	940=007166Q
947=007174Q	953=007200Q	954=007203Q	955=007222Q	956=007226Q	957=007237Q
958=007264Q	959=007270Q	960=007274Q	962=007306Q	963=007311Q	964=007315Q
965=007434Q	966=007455Q	967=007466Q	968=007510Q	969=007513Q	970=007521Q
971=007526Q	972=007544Q	973=007571Q	974=007600Q	975=007601Q	977=007607Q
982=007615Q	983=007621Q	984=007657Q	985=007665Q	986=007670Q	987=007704Q
988=007750Q	989=007752Q	990=007755Q	991=007777Q	992=010005Q	995=010011Q
996=010014Q	997=010023Q	998=010035Q	999=010041Q	1002=010045Q	1003=010050Q
1004=010057Q	1005=010063Q	1011=010070Q	1012=010115Q	1013=010122Q	1014=010127Q
1027=010132Q	1028=010137Q	1030=010150Q	1031=010161Q	1032=010172Q	1034=010201Q
1035=010212Q	1036=010222Q	1037=010225Q	1038=010232Q	1039=010233Q	1043=010247Q
1044=010256Q	1045=010262Q	1046=010276Q	1047=010316Q	1048=010326Q	1049=010331Q
1052=010332Q	1055=010346Q	1056=010365Q	1058=010401Q	1059=010410Q	1060=010424Q
1061=010440Q	1062=010443Q	1063=010451Q	1064=010456Q	1065=010461Q	1066=010470Q
1067=010471Q	1079=010474Q	1080=010507Q	1081=010534Q	1082=010551Q	1083=010560Q
1084=010561Q	1085=010570Q	1086=010604Q	1087=010617Q	1088=010622Q	1089=010623Q
1095=010626Q	1096=010641Q	1097=010655Q	1098=010666Q	1099=010675Q	1101=010700Q
1102=010713Q	1103=010710Q	1104=010715Q	1105=010737Q	1106=010762Q	1107=011002Q
1108=011006Q	1109=011032Q	1110=011042Q	1111=011045Q	1112=011050Q	1113=011051Q

1123=0110540 1124=0110600 1129=0110640 1130=0110740 1131=0111010 1132=0111150
1133=0111410 1134=0111430 1135=0111440 1136=0111470 1137=0111610 1138=0111650
1139=0112030 1140=0112100 1141=0112260 1142=0112370 1143=0112530 1144=0112620
1145=0112660 1146=0113140 1147=0113330 1148=0113430 1149=0113620 1150=0113760
1151=0114010 1152=0114140 1153=0114170 1154=0114200 1166=0114230 1167=0114450
1168=0114700 1169=0114720 1175=0115020 1176=0115070 1177=0115210 1178=0115460
1179=0116000 1180=0116200 1181=0116230 1182=0116310 1184=0116370 1185=0116520
1186=0116670 1187=0116720 1188=0117000 1189=0117030 1190=0117140 1191=0117250
1192=0117330 1193=0117510 1194=0117560 1195=0117650 1196=0117740 1197=0117750
1199=0120030 1200=0120110 1201=0120150 1202=0120250 1203=0120330 1204=0120410
1205=0120550 1206=0120660 1207=0120730 1208=0120750 1209=0121240 1210=0121270
1212=0121350 1213=0121500 1214=0121630 1215=0122250 1216=0122350 1217=0122610
1218=0122620 1225=0122650 1226=0122730 1227=0122760 1228=0123070 1229=0123200
1230=0123260 1231=0123540 1232=0123710 1233=0124020 1234=0124100 1235=0124150
1236=0124230 1237=0124300 1238=0124340 1239=0124430 1240=0124570 1241=0125010
1242=0125350 1243=0125400 1244=0125520 1245=0125550 1246=0125710 1247=0125770
1248=0126240 1249=0126330 1250=0126470 1251=0126630 1252=0126660 1254=0126740
1255=0127070 1256=0127160 1257=0127420 1258=0127430 1262=0127460 1263=0127600
1264=0130020 1265=0130070 1272=0130120 1273=0130200 1274=0130230 1275=0130360
1276=0130500 1277=0130630 1278=0130640 1279=0131070 1280=0131230 1281=0131240
1282=0131270 1283=0131300 1288=0131330 1289=0131620 1290=0132170 1291=0132220
1292=0132330 1293=0132420 1296=0132450 1297=0132530 1298=0132640 1299=0133020
1300=0133100 1301=0133240 1302=0133470 1303=0133620 1305=0133630 1309=0133670
1314=0133720 1315=0133770 1316=0134070 1317=0134330 1318=0134410 1319=0134460
1320=0134640 1321=0135140 1322=0135230 1323=0135400 1324=0135520 1325=0135710
1326=0136130 1327=0136210 1328=0136470 1329=0136730 1330=0136760 1332=0137010
1333=0137020 1337=0137050 1338=0137130 1339=0137220 1340=0137340 1346=0137410
1347=0137570 1348=0137720 1349=0137670 1350=0140050 1351=0140100 1352=0140110
1355=0140140 1356=0140200 1357=0140250 1366=0140300 1367=0140410 1368=0140430
1369=0140640 1370=0141100 1371=0141150 1372=0141160 1374=0141310 1375=0141350
1376=0141420 1378=0141450 1379=0141530 1380=0141600 1381=0141700 1382=0142060
1383=0142150 1384=0142200 1385=0142270 1386=0142320 1387=0142370 1388=0142530
1389=0142760 1390=0143010 1392=0143060 1393=0143110 1394=0143210 1395=0143250
1396=0143410 1397=0143470 1398=0143520 1399=0143710 1400=0144060 1401=0144110
1403=0144120 1410=0144150 1411=0144230 1412=0144260 1413=0144340 1414=0144440
1415=0144550 1416=0144610 1417=0144700 1418=0145210 1419=0145540 1420=0145570
1421=0145600 1426=0145630 1427=0145760 1428=0145730 1429=0146070 1430=0146370
1431=0146640 1432=0147060 1433=0147110 1434=0147120 1437=0147150 1438=0147270
1439=0147300 1445=0147360 1446=0147510 1447=0147540 1448=0147650 1449=0147760
1450=0150040 1451=0150070 1452=0150140 1453=0150360 1454=0150420 1455=0150530
1457=0150640 1458=0150700 1459=0151010 1460=0151140 1461=0151170 1462=0151300
1463=0151430 1464=0151440 1468=0151470 1469=0151620 1470=0152000 1471=0152010
1479=0152110 1480=0152270 1481=0152510 1482=0152560 1483=0152670 1484=0152700
1490=0152730 1491=0152770 1492=0153340 1493=0153350 1500=0153450 1501=0153630
1502=0154660 1503=0154770 1504=0155100 1505=0155110 1511=0155140 1512=0155200
1513=0155660 1514=0155670 1518=0155720 1519=0156020 1520=0156030 1525=0156060
1526=0156210 1527=0156260 1528=0156500 1529=0156610 1530=0157010 1531=0157120
1532=0157310 1533=0157400 1534=0157550 1535=0157620 1536=0157730 1537=0160000
1538=0160030 1539=0160040 1546=0160070 1547=0160130 1548=0160570 1549=0160600
1556=0160730 1557=0161030 1558=0161170 1559=0161240 1560=0161340 1561=0161450
1562=0161560 1563=0161570 1566=0161620 1567=0162160 1568=0162170 1571=0162220
1572=0162300 1573=0162350 1574=0162500 1575=0162610 1576=0162720 1577=0162730
1586=0163010 1589=0163060 1590=0163440 1591=0163520 1592=0164170 1593=0164250
1594=0164330 1595=0164400 1596=0164510 1597=0164540 1598=0164570 1599=0164620
1600=0165000 1601=0165020 1602=0165100 1603=0165130 1604=0165160 1605=0165170
1608=0165250 1610=0165350 1611=0165610 1612=0166070 1613=0166130 1616=0166230
1617=0166520 1618=0166530 1621=0166560 1622=0166620 1623=0166750 1624=0167000
1625=0167300 1626=0167330 1627=0167410 1628=0167420 1631=0167450 1632=0167510
1633=0167600 1634=0167630 1635=0170130 1636=0170160 1637=0170240 1638=0170250
1641=0170300 1642=0170360 1643=0170600 1644=0170630 1645=0170710 1646=0170720
1649=0171000 1650=0171070 1651=0171100 1657=0171130 1658=0171250 1659=0171350
1660=0171550 1661=0171710 1662=0171750 1663=0172100 1664=0172230 1665=0172240
1667=0172310 1668=0172530 1669=0172670 1670=0173110 1671=0173140 1672=0173170
1674=0173240 1675=0173520 1676=0174000 1677=0174030 1678=0174110 1679=0174330
1680=0174340 1685=0174370 1686=0174560 1687=0174670 1688=0174760 1690=0175010
1691=0175050 1692=0175160 1693=0175410 1694=0175460 1695=0175720 1696=0175750
1697=0176040 1698=0176100 1699=0176270 1700=0176470 1701=0176530 1702=0176740
1703=0177140 1704=0177230 1705=0177260 1706=0177270 1712=0177340 1713=0177410
1714=0177440 1715=0177450 1730=0177500 1731=0177560 1775=0177620 1776=0177730
1777=0200240 1778=0200320 1779=0200350 1781=0200420 1782=0200660 1783=0201200
1784=0201250 1785=0201270 1787=0201430 1788=0201510 1789=0201570 1791=0201650
1792=0202070 1793=0202250 1794=0202350 1796=0202320 1797=0202560 1798=0202640
1799=0202770 1800=0203230 1801=0203530 1802=0203760 1803=0204040 1804=0204070
1805=0204230 1806=0204340 1807=0204420 1808=0204500 1809=0204560 1810=0204640
1811=0204720 1812=0205000 1813=0205060 1814=0205140 1815=0205220 1816=0205300
1817=0205360 1818=0205440 1819=0205520 1820=0205600 1821=0205660 1822=0205740
1823=0206020 1824=0206100 1825=0206160 1826=0206240 1827=0206320 1828=0206400
1829=0206460 1830=0206540 1831=0206620 1832=0206700 1833=0206760 1834=0207040
1835=0207120 1836=0207200 1837=0207260 1838=0207340 1839=0207420 1840=0207500
1841=0207600 1842=0207660 1843=0207740 1844=0210020 1845=0210230 1846=0210340
1847=0211730 1848=0212070 1853=0212130 1854=0212200 1855=0212430 1856=0213010
1857=0213110 1858=0213250 1859=0213410 1860=0213530 1861=0213550 1863=0213630
1864=0213700 1865=0213730 1866=0214310 1867=0214340 1870=0214310 1871=0214360
1872=0214400 1873=0214610 1874=0214720 1875=0214750 1876=0215050 1877=0215200
1878=0215340 1879=0215400 1880=0215510 1881=0215650 1882=0216000 1883=0216100
1884=0216150 1885=0216220 1886=0216440 1887=0216500 1888=0216530 1889=0216640
1892=0216650 1900=0216730 1901=0217060 1902=0217230 1903=0217300 1904=0217340
1905=0217470 1906=0217730 1907=0217760 1908=0217770 1910=0220100 1911=0220140
1912=0220210 1913=0220310 1915=0220330 1916=0220370 1917=0220460 1918=0220560
1919=0220640 1920=0221070 1921=0221740 1922=0222020 1923=0222160 1924=0222210
1925=0222240 1926=0222270 1927=0222520 1928=0222770 1929=0223130 1930=0223160
1931=0223300 1932=0223330 1933=0223540 1934=0223570 1935=0223640 1936=0223740
1937=0223750 1938=0224000 1939=0224100 1940=0224120 1941=0224150 1942=0224200
1943=0224250 1945=0224300 1948=0224350 1949=0224460 1950=0224510 1951=0224620
1952=0224670 1953=0225000 1954=0225030 1955=0225140 1956=0225170 1957=0225220
1958=0225260 1959=0225310

```
S00001 22744Q  S00002 22744Q  S00003 22744Q  S00004 22744Q
S00005 25000Q  S00006 22744Q  S00023 22744Q  S00044 22744Q
S00046 22745Q  S00047 22746Q  S00049 22747Q  S00050 22750Q
S00052 23072Q  S00054 23074Q  S00057 23076Q  S00060 23101Q
S00061 23102Q  S00063 23103Q  S00065 23104Q  S00066 23105Q
S00067 23106Q  S00068 23107Q  S00069 23110Q  S00070 23112Q
S00071 23113Q  S00072 23114Q  S00076 23117Q  S00077 23120Q
S00081 23123Q  S00082 23124Q  S00090 23126Q  S00095 23127Q
S00105 23130Q  S00116 23133Q  S00123 23134Q  S00125 23136Q
S00129 23140Q  S00131 23142Q  S00136 23144Q  S00141 23145Q
S00149 23146Q  S00156 23150Q  S00157 23153Q  S00158 23154Q
S00159 23155Q  S00160 23156Q  S00161 23160Q  S00162 23162Q
S00163 23164Q  S00165 24162Q  S00166 24163Q  S00167 24164Q
S00169 24166Q  S00170 24167Q  S00171 24170Q  S00173 24172Q
S00175 24174Q  S00176 24176Q  S00177 24200Q  S00179 24203Q
S00180 24204Q  S00181 24206Q  S00182 24210Q  S00184 24212Q
S00185 24214Q  S00186 24216Q  S00187 24220Q  S00188 24221Q
S00190 24222Q  S00196 24224Q  S00199 24227Q  S00208 24230Q
S00209 24232Q  S00211 24234Q  S00218 24236Q  S00221 24240Q
S00228 24243Q  S00229 24244Q  S00232 24246Q  S00242 24250Q
S00244 24252Q  S00247 24255Q  S00251 24256Q  S00253 24260Q
S00254 24262Q  S00257 24265Q  S00261 24266Q  S00262 24270Q
S00266 24271Q  S00270 24272Q  S00271 24274Q  S00273 24277Q
S00278 24300Q  S00279 24302Q  S00281 24304Q  S00282 24305Q
S00295 24306Q  S00297 24311Q  S00303 24312Q  S00305 24315Q
S00316 24316Q  S00318 24321Q  S00323 24322Q  S00325 24324Q
S00326 24326Q  S00327 24330Q  S00328 24332Q  S00329 24335Q
S00346 24336Q  S00347 24437Q  S00348 24440Q  S00350 24442Q
S00357 24444Q  S00359 24446Q  S00360 24447Q  S00371 24450Q
S00374 24452Q  S00391 24454Q  S00405 24456Q  S00421 24457Q
S00431 24460Q  S00433 24461Q  S00434 24462Q  S00443 24464Q
S00449 24465Q  S00451 24466Q  S00458 24471Q  S00459 24472Q
S00460 24473Q  S00472 24474Q  S00474 24477Q  S00482 24500Q
S00484 24503Q  S00485 24511Q  S00486 24512Q  S00487 24513Q
S00499 24514Q  S00501 24516Q  S00502 24521Q  S00503 24522Q
S00511 24523Q  S00520 24524Q  S00525 24525Q  S00530 24526Q
S00531 24527Q  S00532 24530Q  S00566 24532Q  S00576 24533Q
S00577 24534Q  S00579 24535Q  S00582 24536Q  S00595 24540Q
S00596 24542Q  S00597 24544Q  S00598 24546Q  S00599 24550Q
S00600 24552Q  S00601 24554Q  S00602 24556Q  S00603 24557Q
S00610 24560Q  S00624 24562Q  S00625 24565Q  S00642 24566Q
S00643 24570Q  S00644 24572Q  S00660 24574Q  S00662 24575Q
S00663 24576Q  S00664 24577Q  S00665 24600Q  S00680 24601Q
S00688 24602Q  S00689 24603Q  S00690 24604Q  S00715 24605Q
S00716 24606Q  S00722 24607Q  S00730 24610Q  S00742 24612Q
S00749 24614Q  S00756 24616Q  S00759 24621Q  S00777 24622Q
S00787 24624Q  S00789 24627Q  S00790 24630Q  S00791 24631Q
S00803 24632Q  S00823 24635Q  S00824 24636Q  S00825 24637Q
S00840 24640Q  S00851 24642Q  S00854 24644Q  S00856 24647Q
S00857 24650Q  S00858 24651Q  S00860 24652Q  S00862 24655Q
S00863 24656Q  S00973 24777Q  S01004 22743Q  S01005 22742Q
S01006 22740Q  S01008 22736Q  S01010 22734Q  S01011 22732Q
S01013 22730Q  S01014 22726Q  S01015 22724Q  S01016 22722Q
S01017 22720Q  S01018 22716Q  S01019 22714Q  S01021 22712Q
STACK SIZE = 18 BYTES
000400Q   LXI SP  22712Q    JMP      22021Q
000406Q    000Q 000Q 100Q 147Q 143Q 201Q 000Q 000Q 000Q 000Q
000420Q    000Q 000Q 100Q 151Q 156Q 201Q
000426Q    000Q 000Q 100Q 162Q 157Q 201Q 000Q 000Q 000Q 000Q
000440Q    000Q 000Q 100Q 143Q 142Q 201Q
000446Q   LXI H   23117Q   MOV MC           INR L            MOV ME
000454Q   INX H            MOV MD           JMP       5Q     RET
000462Q   LXI H   23123Q   MOV MC           INR L            MOV ME
000470Q   INX H            MOV MD           JMP       5Q     RET
000476Q   MOV CI      2Q   MOV EI     15Q   MOV DI      0Q   CALL     446Q
000507Q   MOV CI      2Q   MOV EI     12Q   MOV DI      0Q   CALL     446Q
000520Q   LXI H   22746Q   MOV MI      0Q   RET              LXI H  23126Q
000531Q   MOV MC           MOV CI      2Q   MOV EM           MOV DI      0Q
000537Q   CALL     446Q    LXI H  23126Q    MOV AM           SUB I      15Q
000550Q   JNZ      560Q    DCR H            MOV LI     346Q  MOV MI      0Q
000560Q   LXI H   22746Q   INR M            MOV AM           DCR L
000566Q   SUB M            CZ       476Q    RET              LXI H  23127Q
000576Q   MOV MC           IN         1Q    RRC              RRC
000603Q   RRC              JC       577Q    LXI H  23127Q    MOV AM
000613Q   CMA              OUT         0Q   RET              MOV CI      1Q
000621Q   MOV EI      0Q   MOV DI      0Q   CALL     462Q    RET
000631Q   MOV CI     13Q   MOV EI      0Q   MOV DI      0Q   CALL     462Q
000642Q   RET              LXI H  23130Q    MOV MC           INX H
000650Q   MOV MB           MOV CI     11Q   DCR L            MOV EM
000655Q   INR L            MOV DM           CALL     446Q    CALL     476Q
000665Q   RET              LHLD    23074Q   MOV AM           SUB I       0Q
000674Q   JNZ      711Q    LXI H  23105Q    MOV CM           MOV LI     107Q
000705Q   MOV MC           JMP      731Q    LHLD   23074Q    MOV AM
000715Q   ANA I        7Q  DCR A            LXI H  23107Q    MOV MA
000724Q   LHLD   23074Q    MOV MI      0Q   LXI H  23106Q    MOV AM
000735Q   INR L            SUB M            JZ       756Q    MOV CM
000743Q   DCR L            MOV MC           MOV CI      16Q  INR L
000750Q   MOV EM           MOV DI      0Q   CALL     446Q    RET
000757Q   LXI H   23133Q   MOV MC           MOV AM           SUB I      141Q
000766Q   SBC A            MOV CA           MOV AI     172Q  SUB M
000773Q   SBC A            ORA C            RRC              JNC       1003Q
001001Q   MOV AM           RET              MOV AM           SUB I       40Q
001006Q   RET              LXI H  23134Q    MOV MC           INX H
```

```
001014Q  MOV MB          INR L          MOV MI      0Q  INX H
001021Q  MOV MI      0Q  LXI H  23136Q  MOV AM          INR L
001030Q  MOV BM          MOV LI   134Q  SUB M           INR L
001035Q  MOV CA          MOV AB         SBC M           JNC     1072Q
001043Q  MOV AI      7Q  MOV BI    14Q  MOV CB          DCR C
001051Q  JNZ      1050Q  DCR A          JNZ     1047Q  LHLD   23136Q
001063Q  INX H           SHLD   23136Q  JMP     1023Q  RET
001073Q  LXI H   23140Q  MOV MC         INX H          MOV MB
001101Q  INR L           MOV MI     1Q  INX H          MOV MI      0Q
001107Q  LXI H   23140Q  MOV AM         INR L          MOV BM
001115Q  INR L           SUB M          INR L          MOV CA
001121Q  MOV AB          SBC M          JC      1222Q  CALL    631Q
001131Q  RRC             RC             MOV AI   377Q  MOV BI    14Q
001137Q  MOV CB          DCR C          JNZ     1140Q  DCR A
001145Q  JNZ      1137Q  MOV AI   377Q  MOV BI    14Q  MOV CB
001155Q  DCR C           JNZ    1155Q  DCR A          JNZ     1154Q
001165Q  MOV AI    207Q  MOV BI    14Q  MOV CB          DCR C
001173Q  JNZ      1172Q  DCR A          JNZ     1171Q  LXI H   23142Q
001205Q  MOV CM          INR L          MOV BM          LXI H     1Q
001213Q  DAD B           SHLD   23142Q  JMP     1107Q  RET
001223Q  LXI H   23144Q  MOV AM         RRC            JNC     1237Q
001233Q  IN          5Q  CMA            RET            IN        7Q
001241Q  CMA             RET            LXI H   23145Q  MOV MC
001247Q  CALL     1223Q  RRC            RRC            CMA
001255Q  RRC             JC     1247Q  MOV AM          RRC
001263Q  JNC      1276Q  INR L          MOV AM          CMA
001271Q  OUT         4Q  JMP    1303Q  INR L          MOV AM
001300Q  CMA             OUT       6Q  RET            LXI H   23146Q
001307Q  MOV MI      0Q  INX H          MOV MI      0Q  MOV AI    350Q
001316Q  MOV BI    375Q  LXI H   23146Q  SUB M          INR L
001325Q  MOV CA          MOV AB         SBC M          JC      1402Q
001333Q  CALL     1223Q  RRC            JNC     1363Q  MOV AM
001343Q  RRC             JNC    1355Q  IN          4Q  CMA
001352Q  ANA I     177Q  RET            IN          6Q  CMA
001360Q  ANA I     177Q  RET            MOV LI   146Q  MOV CM
001366Q  INR L           MOV BM         LXI H     1Q  DAD B
001374Q  SHLD    23146Q  JMP    1314Q  MOV AI   377Q  RET
001405Q  LXI H   24222Q  MOV MC         INX H          MOV MB
001413Q  LXI H   23150Q  MOV CM         INR L          MOV BM
001421Q  LHLD    24222Q  MOV MC         INX H          MOV MB
001427Q  LXI H   24222Q  MOV CM         INR L          MOV BM
001435Q  LXI H   23150Q  MOV MC         INX H          MOV MB
001443Q  RET             LXI H   23150Q  MOV CM         INR L
001451Q  MOV BM          MOV CB         MOV AC         SUB I       0Q
001456Q  JNZ      1556Q  JMP    1514Q
001464Q   106Q 122Q 105Q 105Q 040Q 123Q 124Q 117Q 122Q 101Q 107Q 105Q 040Q 111Q
001502Q   123Q 040Q 105Q 115Q 120Q 124Q 131Q 040Q 040Q 044Q
001514Q  LXI B    1464Q  CALL     643Q  LXI H   24227Q  MOV AM
001526Q  RRC             JNC    1542Q  MOV MI      0Q  LXI SP  22712Q
001537Q  JMP       403Q  MOV CI      0Q  MOV EI      0Q  MOV DI      0Q
001550Q  CALL      446Q  JMP    1743Q  DCR L          MOV CM
001560Q  INR L           MOV BM         LXI H   24224Q  MOV MC
001566Q  INX H           MOV MB         LHLD    23150Q  MOV AM
001574Q  INX H           MOV BM         LXI H   23150Q  MOV MA
001602Q  INX H           MOV MB         LHLD    24224Q  MOV MI      0Q
001611Q  INX H           MOV MI      0Q  DCX H          INX H
001616Q  INX H           MOV MI    201Q  INX H          MOV MI    201Q
001624Q  LXI B       4Q  LHLD   24224Q  DAD B          MOV MI    201Q
001635Q  INX H           MOV MI    201Q  LXI B       6Q  LHLD   24224Q
001646Q  DAD B           MOV MI    201Q  INX H          MOV MI    201Q
001654Q  LXI B      10Q  LHLD   24224Q  DAD B          MOV MI    201Q
001665Q  INX H           MOV MI    201Q  LXI B      12Q  LHLD   24224Q
001676Q  DAD B           MOV MI    201Q  INX H          MOV MI    201Q
001704Q  LXI B      14Q  LHLD   24224Q  DAD B          MOV MI    201Q
001715Q  INX H           MOV MI    201Q  LXI B      16Q  LHLD   24224Q
001726Q  DAD B           MOV MI    201Q  INX H          MOV MI    201Q
001734Q  LXI H   24224Q  MOV AM         INR L          MOV BM
001742Q  RET             RET            LXI H   24230Q  MOV MC
001750Q  INX H           MOV MB         INR L          MOV ME
001754Q  INX H           MOV MD         MOV LI    230Q  MOV CM
001761Q  INR L           MOV BM         MOV LI    234Q  MOV MC
001766Q  INX H           MOV MB         LXI H   24232Q  MOV AM
001774Q  INR L           MOV BM         SUB I      20Q  MOV CA
002001Q  MOV AB          SBC I       0Q  MOV BA         MOV AC
002006Q  INR L           SUB M          INR L          MOV CA
002012Q  MOV AB          SBC M          JC      2051Q  JMP     2037Q
002022Q  LXI B       20Q  LHLD   24234Q  DAD B          SHLD    24234Q
002034Q  JMP       1770Q  DCR L          MOV CM         INR L
002042Q  MOV BM          CALL     1405Q  JMP     2022Q  RET
002052Q  LXI H   24236Q  MOV MC         INX H          MOV MB
002060Q  DCR L           MOV AM         INR L          MOV BM
002064Q  ANA I     360Q  MOV CA         MOV AB         ANA I     377Q
002072Q  DCR L           MOV MC         INX H          MOV MA
002076Q  MOV BA          MOV AC         MOV LI    216Q  SUB M
002103Q  INR L           MOV CA         MOV AB         SBC M
002107Q  ORA C           RZ             LXI H   24236Q  MOV CM
002115Q  INR L           MOV BM         MOV CB         MOV AC
002121Q  SUB I       0Q  RZ             LHLD    24236Q  MOV AM
002130Q  INX H           MOV BM         LXI H   24240Q  MOV MA
002136Q  INX H           MOV MB         MOV LI    236Q  MOV CM
002143Q  INR L           MOV BM         CALL     1405Q  LXI H   24240Q
002153Q  MOV CM          INR L          MOV BM         MOV LI    236Q
002160Q  MOV MC          INX H          MOV MB         JMP     2111Q
002166Q  RET             LXI H   24243Q  MOV MC         INR L
```

```
002174Q  MOV ME
002200Q  MOV CM
002204Q  MOV AC
002213Q  MOV BI      0Q
002222Q  INX H
002226Q  SUB C
002234Q  INX H
002242Q  INX H
002252Q  LHLD    24244Q
002264Q  MOV AM
002273Q  SHLD    24246Q
002305Q  LHLD    24246Q
002315Q  XCHG
002323Q  JNZ      2357Q
002335Q  INX H
002343Q  INX H
002353Q  DAD B
002363Q  INR L
002371Q  MOV MC
002377Q  MOV AM
002406Q  LXI H   24250Q
002414Q  INR L
002420Q  LHLD    24252Q
002432Q  INX H
002440Q  ANA I      17Q
002452Q  MOV AM
002457Q  MOV CA
002466Q  MOV MC
002474Q  MOV AM
002502Q  MOV MA
002510Q  LHLD    24252Q
002522Q  MOV AM
002530Q  INX H
002536Q  RRC
002547Q  MOV CM
002553Q  MOV MC
002561Q  INX H
002571Q  SHLD    24262Q
002600Q  SUB I       0Q
002611Q  INX H
002617Q  INX H
002627Q  DAD B
002637Q  RET
002645Q  MOV MB
002653Q  MOV AM
002662Q  LXI H   24266Q
002670Q  LXI H   23162Q
002676Q  LXI B       2Q
002710Q  LXI H   24270Q
002720Q  INX H
002726Q  ANA I      17Q
002740Q  LHLD    23162Q
002746Q  LXI H   23162Q
002754Q  LXI B       2Q
002766Q  RET
002774Q  MOV MB
003000Q  MOV MD
003010Q  MOV MI      377Q
003017Q  MOV BM
003027Q  SUB I       201Q
003036Q  CMA
003045Q  MOV CM
003052Q  MOV EM
003062Q  LXI H   24300Q
003070Q  INR L
003074Q  JMP      3167Q
003104Q  MOV BM
003114Q  SUB I       201Q
003122Q  MOV AM
003126Q  JC       3077Q
003136Q  MOV CM
003144Q  LXI H   24305Q
003154Q  SBC A
003160Q  ANA C
003166Q  RET
003175Q  MOV MI      377Q
003206Q  CALL     3133Q
003220Q  LXI H   24304Q
003231Q  MOV AI      377Q
003236Q  LXI H   24306Q
003244Q  DCR L
003250Q  MOV CB
003257Q  MOV AI      377Q
003267Q  LXI H   24306Q
003275Q  ANA I       360Q
003303Q  MOV BA
003313Q  SUB I       201Q
003322Q  CMA
003330Q  RET
003337Q  MOV MC
003345Q  MOV MI      377Q
003354Q  MOV BM
003361Q  ANA I       377Q
003372Q  MOV MA

    INX H
    INR L
    SUB I        0Q
    RET
    MOV BM
    JNC      2247Q
    MOV BM
    MOV MB
    DAD B
    SUB I       201Q
    JMP      2261Q
    MOV MC
    MOV AE
    CALL     1444Q
    MOV MB
    MOV MB
    SHLD    24246Q
    MOV BM
    INX H
    RRC
    MOV CM
    MOV MC
    INX H
    SHLD    23156Q
    SUB I        0Q
    INR L
    MOV AB
    INX H
    INX H
    INX H
    DAD B
    RET
    MOV MB
    JNC      2565Q
    INR L
    INX H
    SHLD    24262Q
    XCHG
    JNZ      2633Q
    MOV BM
    MOV MB
    SHLD    24262Q
    LXI H   24266Q
    INR L
    RRC
    MOV CM
    MOV MC
    LHLD    23162Q
    MOV CM
    SHLD    23160Q
    SUB I        0Q
    MOV MA
    MOV MA
    LHLD    23162Q
    LXI H   24272Q
    INR L
    LXI H   23153Q
    LXI H   24272Q
    CALL     2366Q
    JZ       3061Q
    RRC
    INR L
    CALL     2640Q
    MOV MC
    MOV ME
    LXI H   24300Q
    CALL     2366Q
    ADD I       377Q
    RLC
    MOV AM
    INR L
    MOV MA
    MOV CA
    RRC
    LXI H   23153Q
    CALL     3077Q
    LXI H   22743Q
    MOV AM
    RET
    MOV MC
    MOV CM
    MOV AC
    RET
    MOV AM
    MOV CA
    CALL     2366Q
    JZ       3331Q
    RRC
    MOV AI      377Q
    INX H
    LXI H   24312Q
    ANA I       360Q
    MOV BA
    SUB I       201Q

    MOV MD
    MOV BM
    JNZ      2216Q
    LHLD    24244Q
    MOV CB
    DCX H
    LXI H   24244Q
    JMP      2216Q
    SHLD    24246Q
    JZ       2301Q
    LXI H   24243Q
    INX H
    ANA I       17Q
    LHLD    24244Q
    LXI H   24246Q
    LXI B        2Q
    LXI H   24246Q
    RET
    MOV MB
    JNC      2427Q
    INR L
    INX H
    SHLD    23156Q
    XCHG
    JNZ      2517Q
    MOV BM
    SBC I        0Q
    MOV MA
    MOV BM
    MOV MB
    SHLD    23156Q
    LXI H   24256Q
    LXI H   23154Q
    MOV MI       0Q
    MOV BM
    MOV MB
    LHLD    24262Q
    MOV AE
    LHLD    24260Q
    LXI H   24260Q
    LXI B        2Q
    LHLD    24262Q
    MOV MC
    MOV ME
    JNC      2710Q
    INR L
    INX H
    DAD B
    LHLD    23160Q
    XCHG
    JNZ      2766Q
    INX H
    INX H
    DAD B
    MOV MC
    MOV ME
    MOV MI      377Q
    MOV CM
    LXI H   24277Q
    MOV AM
    JNC      3012Q
    MOV BM
    JMP      3012Q
    INX H
    INX H
    MOV CM
    LXI H   24304Q
    SBC A
    ANA C
    RET
    MOV BM
    SUB I       201Q
    MOV AM
    JC       3133Q
    MOV MI      377Q
    LXI H   22743Q
    SUB M
    SUB I       201Q
    XRA A
    INX H
    INR L
    SUB I        0Q
    LXI H   23153Q
    INR L
    MOV AB
    LXI H   24311Q
    MOV AM
    JNC      3267Q
    RET
    MOV MB
    MOV AM
    MOV CA
    CALL     2366Q
    ADD I       377Q

    DCR L
    MOV CB
    XRA A
    MOV AM
    MOV MA
    LXI B        2Q
    LHLD    24246Q
    INX H
    MOV CM
    SHLD    24246Q
    SUB I        0Q
    MOV MA
    MOV MA
    LHLD    24246Q
    MOV AM
    LXI H   24250Q
    LXI H   23153Q
    MOV MI       0Q
    MOV BM
    MOV MB
    LHLD    23156Q
    MOV AE
    LXI H   23156Q
    SUB I       20Q
    LXI H   24252Q
    LHLD    24252Q
    LXI H   24252Q
    LXI B        2Q
    LHLD    23156Q
    MOV MC
    MOV AM
    LXI H   24256Q
    INR L
    LHLD    24260Q
    INX H
    ANA I       17Q
    MOV AM
    MOV MA
    LHLD    24260Q
    MOV AM
    INX H
    LXI H   23155Q
    MOV MI       0Q
    MOV BM
    MOV MB
    SHLD    23160Q
    MOV MC
    MOV AE
    CALL     1444Q
    MOV MB
    MOV MB
    SHLD    23160Q
    INX H
    INX H
    MOV LI      155Q
    INR L
    MOV MA
    RLC
    MOV LI      274Q
    MOV LI      277Q
    RET
    MOV MB
    MOV MD
    INR L
    MOV MA
    MOV CA
    RRC
    LXI H   24302Q
    CALL     2524Q
    ADD I       377Q
    RLC
    MOV AM
    INR L
    MOV MA
    JNZ      3234Q
    JNZ      3177Q
    RET
    MOV MB
    MOV BM
    JNZ      3262Q
    MOV MI      377Q
    MOV BM
    ANA I       377Q
    MOV MA
    RLC
    XRA A
    LXI H   24312Q
    LXI H   23153Q
    INR L
    MOV AB
    CALL     2366Q
    LXI H   24315Q
    SBC A
```

```
003400Q  MOV CA        PUSH B       MOV CI   13Q   MOV EI    0Q
003406Q  MOV DI    0Q  CALL  462Q    CMA            POP D
003415Q  ANA E         RRC          JNC   3470Q    LXI H  24315Q
003425Q  MOV AM        RLC          CMA            RRC
003431Q  JNC   3347Q   LXI H 22744Q MOV AM         RRC
003441Q  JNC   3456Q   LXI H 24315Q MOV CM         CALL    526Q
003453Q  JMP   3465Q   LXI H 24315Q MOV CM         CALL    573Q
003465Q  JMP   3347Q   RET          MOV CI    2Q   MOV BI    0Q
003475Q  CALL  1073Q   CALL  1223Q  RRC            JNC   3527Q
003507Q  CALL  1304Q   MOV LI  112Q  MOV MA         MOV CI    2Q
003517Q  MOV BI    0Q  CALL  1073Q  JMP   3500Q    RET
003530Q  LXI H 24316Q  MOV MC       INX H          MOV MB
003536Q  CALL  3471Q   MOV LI  153Q  MOV MI   377Q  LXI H  24316Q
003550Q  MOV CM        INR L        MOV BM         CALL   2366Q
003556Q  LXI H 24321Q  MOV MA       SUB I   201Q   JZ     3605Q
003567Q  MOV AM        RLC          CMA            RRC
003573Q  JNC   3545Q   MOV CM       CALL  1243Q    JMP    3545Q
003605Q  RET           LXI H 24322Q MOV MC         INX H
003613Q  MOV MB        JMP   4022Q  LXI H 23154Q   MOV MI   377Q
003624Q  LXI H 24324Q  MOV CM       INR L          MOV BM
003632Q  LXI H 23156Q  MOV MC       INX H          MOV MB
003640Q  MOV CI    0Q  MOV BI    0Q CALL  2366Q    LXI H  24335Q
003652Q  MOV MA        MOV CA       PUSH B         MOV LI   322Q
003657Q  MOV CM        INR L        MOV BM         CALL   2524Q
003665Q  POP D         SUB E        JNZ   3746Q    LXI H  24335Q
003675Q  MOV AM        SUB I   201Q JNZ   3706Q    MOV AI   377Q
003705Q  RET           CALL  1223Q  RRC            JNC    3640Q
003715Q  LXI H 24330Q  MOV CM       INR L          MOV BM
003723Q  MOV LI  326Q  MOV MC       INX H          MOV MB
003730Q  CALL  1304Q   MOV EA       MOV CI    0Q   MOV BI    0Q
003740Q  CALL  2640Q   JMP   3640Q  LXI H 24335Q   MOV AM
003752Q  SUB I   201Q  JZ    4020Q  MOV LI   324Q  MOV CM
003762Q  INR L         MOV BM       LXI H 23156Q   MOV MC
003770Q  INX H         MOV MB       MOV CI    0Q   MOV BI    0Q
003776Q  CALL  2366Q   LXI H 23112Q MOV MA         MOV LI   156Q
004007Q  MOV CM        INR L        MOV BM         LXI H  24324Q
004015Q  MOV MC        INX H        MOV MB         XRA A
004021Q  RET           LXI H 23072Q MOV AM         INR L
004027Q  MOV BM        SUB I    0Q  MOV CA         MOV AB
004034Q  SBC I    0Q   ORA C        JNZ   4055Q    LXI H  24330Q
004045Q  MOV MI   2Q   INX H        MOV MI    0Q   JMP    4075Q
004055Q  LHLD 23072Q   INX H        SHLD  24330Q   CALL   3471Q
004067Q  MOV LI  102Q  MOV CM       CALL  1243Q    CALL   1444Q
004100Q  LXI H 24210Q  MOV MA       INX H          MOV MB
004106Q  MOV CA        MOV LC       MOV HB         INX H
004112Q  SHLD  24324Q  LXI H 23155Q MOV MI   377Q  LXI H  24330Q
004125Q  MOV CM        INR L        MOV BM         MOV LI   326Q
004132Q  MOV MC        INX H        MOV MB         LHLD   24326Q
004140Q  DCX H         SHLD  24326Q XCHG           MOV AE
004146Q  SUB I    0Q   MOV EA       MOV AD         SBC I     0Q
004154Q  ORA E         ADD I   377Q SBC A          LXI H  22742Q
004163Q  MOV MA        CALL   631Q  CMA            LXI H  22742Q
004173Q  ANA M         RRC          JNC   4320Q    LXI H  24332Q
004203Q  MOV MI   0Q   INX H        MOV MI    0Q   MOV AI   210Q
004212Q  MOV BI   23Q  LXI H 24332Q SUB M          INR L
004221Q  MOV CA        MOV AB       SBC M          JC     4135Q
004227Q  CALL  1223Q   RRC          JNC   4300Q    LXI H  24330Q
004241Q  MOV CM        INR L        MOV BM         MOV LI   326Q
004246Q  MOV MC        INX H        MOV MB         CALL   1304Q
004254Q  MOV EA        LXI H 24210Q MOV CM         INR L
004262Q  MOV BM        CALL  2640Q  CALL  3617Q    RRC
004272Q  JNC   4300Q   MOV AI   377Q RET           LXI H  24332Q
004303Q  MOV CM        INR L        MOV BM         LXI H      1Q
004311Q  DAD B         SHLD  24332Q JMP   4210Q    XRA A
004321Q  RET           LXI H 24437Q MOV MI    0Q   MOV AI    37Q
004331Q  LXI H 24437Q  SUB M        JC    4364Q    MOV CM
004341Q  MOV BI    0Q  DCR H        MOV LI   336Q  DAD B
004347Q  DAD B         MOV MI    0Q INX H          MOV MI    0Q
004355Q  LXI H 24437Q  INR M        JNZ   4327Q    RET
004365Q  LXI H 24444Q  MOV MC       INX H          MOV MB
004373Q  INR L         MOV MI    0Q LXI H 23153Q   MOV MI   377Q
004403Q  LXI H 24444Q  MOV CM       INR L          MOV BM
004411Q  CALL  2366Q   LXI H 24447Q MOV MA         SUB I    201Q
004422Q  JZ    4434Q   MOV AM       DCR L          ADD M
004430Q  MOV MA        JMP   4403Q  DCR L          MOV AM
004436Q  ANA I    37Q  MOV LI    37Q MOV MA         MOV CA
004444Q  MOV BI    0Q  DCR H        MOV LI   336Q  DAD B
004452Q  DAD B         MOV AM       INX H          MOV BM
004456Q  LXI H 24440Q  MOV MA       INX H          MOV MB
004464Q  MOV CB        MOV AC       SUB I     0Q   JNZ    4475Q
004473Q  XRA A         RET          MOV LI    37Q  MOV CM
004500Q  MOV BI    0Q  DCR H        MOV LI   336Q  DAD B
004506Q  DAD B         SHLD  24442Q LXI B     6Q   LHLD   24440Q
004520Q  DAD B         XCHG         LXI H 24444Q   MOV CM
004526Q  INR L         MOV BM       CALL  3062Q    CMA
004534Q  RRC           JNC   4577Q  LXI H 24440Q   MOV CM
004544Q  INR L         MOV BM       INR L          MOV MC
004550Q  INX H         MOV MB       LHLD  24440Q   MOV AM
004556Q  INX H         MOV BM       LXI H 24440Q   MOV MA
004564Q  INX H         MOV MB       MOV CB         MOV AC
004570Q  SUB I    0Q   JNZ   4512Q  XRA A          RET
004577Q  MOV AI  377Q  RET          LXI B     6Q   LHLD   24440Q
004610Q  DAD B         MOV AM       INX H          MOV BM
004614Q  MOV CA        CALL  2052Q  LXI B     4Q   LHLD   24440Q
```

```
004626Q  DAD B              MOV AM              INX H               MOV BM
004632Q  MOV CA             CALL     2052Q      LHLD    24440Q      MOV AM
004642Q  INX H              MOV BM              LHLD    24442Q      MOV MA
004650Q  INX H              MOV MB              LXI H   24440Q      MOV CM
004656Q  INR L              MOV BM              CALL     1405Q      RET
004664Q  LXI H   24450Q     MOV MC              INX H               MOV MB
004672Q  CALL     1444Q     LXI H   24440Q      MOV MA              INX H
004702Q  MOV MB             MOV LI    37Q       MOV CM              MOV BI      0Q
004710Q  DCR H              MOV LI   336Q       DAD B               DAD B
004715Q  MOV AM             INX H               MOV BM              LHLD    24440Q
004723Q  MOV MA             INX H               MOV MB              LXI H   24437Q
004731Q  MOV CM             MOV BI      0Q      DCR H               MOV LI   336Q
004737Q  DAD B              DAD B               XCHG                LXI H   24440Q
004745Q  MOV CM             INR L               MOV BM              MOV AC
004751Q  STAX D             INX D               MOV AB              STAX D
004755Q  LXI B       6Q     LHLD    24440Q      DAD B               MOV MI      0Q
004766Q  INX H              MOV MI      0Q      LXI B       6Q      LHLD    24440Q
004777Q  DAD B              XCHG                LXI H   24450Q      MOV CM
005005Q  INR L              MOV BM              CALL     2767Q      LXI B       4Q
005015Q  LHLD    24440Q     DAD B               MOV MI      0Q      INX H
005024Q  MOV MI      0Q     RET                 LXI H   24452Q      MOV MC
005033Q  INX H              MOV MB              LXI B       4Q      LHLD    24440Q
005043Q  DAD B              MOV AM              INX H               MOV BM
005047Q  MOV CA             CALL     2052Q      LXI B       4Q      LHLD    24440Q
005061Q  DAD B              XCHG                LXI H   24452Q      MOV CM
005067Q  INR L              MOV BM              MOV AC              STAX D
005073Q  INX D              MOV AB              STAX D              LHLD    24440Q
005101Q  INX H              INX H               PUSH H              LHLD    24452Q
005107Q  INX H              XCHG                POP H               MOV ME
005113Q  INX H              MOV MD              RET                 JMP     5136Q
005121Q   100Q 173Q 160Q 163Q 174Q 100Q 173Q 162Q 163Q 174Q 056Q 175Q 175Q
005136Q  LXI H   23155Q     MOV MI    377Q      CALL     1444Q      LXI H   23110Q
005151Q  MOV MA             INX H               MOV MB              MOV EI   100Q
005156Q  MOV CA             CALL     2640Q      MOV CI      0Q      MOV BI      0Q
005166Q  MOV EI     151Q    CALL     2640Q      MOV CI      0Q      MOV BI      0Q
005177Q  MOV EI     155Q    CALL     2640Q      LXI H   23110Q      MOV CM
005210Q  INR L              MOV BM              CALL     4365Q      LXI H   23112Q
005220Q  MOV MA             MOV LI    110Q      MOV CM              INR L
005225Q  MOV BM             CALL     4664Q      LXI H   23110Q      MOV CM
005235Q  INR L              MOV BM              CALL     1405Q      LXI H   23155Q
005245Q  MOV MI     377Q    CALL     1444Q      LXI H   23110Q      MOV MA
005256Q  INX H              MOV MB              MOV LI    113Q      MOV MI      0Q
005264Q  MOV AI      14Q    LXI H   23113Q      SUB M               JC      5326Q
005275Q  MOV CM             MOV BI      0Q      LXI H    5121Q      DAD B
005304Q  MOV AM             MOV EA              LXI H   23110Q      MOV CM
005312Q  INR L              MOV BM              CALL     2640Q      LXI H   23113Q
005322Q  INR M              JNZ      5264Q      MOV LI    110Q      MOV CM
005331Q  INR L              MOV BM              CALL     5027Q      RET
005337Q  LXI H   24162Q     MOV CM              MOV BI      0Q      LXI H   23164Q
005350Q  DAD B              DAD B               MOV AM              INX H
005354Q  MOV BM             LXI H   24454Q      MOV MA              INX H
005362Q  MOV MB             DCR H               MOV LI     162Q     INR M
005367Q  INR H              MOV LI     54Q      MOV AM              INR L
005374Q  MOV BM             RET                 LXI H   24162Q      INR M
005402Q  MOV CM             MOV BI      0Q      LXI H   23164Q      DAD B
005411Q  DAD B              MOV AM              INX H               MOV BM
005415Q  RET                CALL     5337Q      DCR H               MOV LI    210Q
005424Q  MOV MA             INX H               MOV MB              RRC
005430Q  JNC      5445Q     CALL      476Q      LXI H   24163Q      MOV CM
005442Q  DCR L              MOV MC              RET                 DCR L
005446Q  MOV CM             INR L               MOV BM              CALL    3334Q
005454Q  MOV CI     134Q    CALL      526Q      JMP      5416Q      RET
005465Q  LXI H   24203Q     MOV AM              ADD A               ORA I       1Q
005474Q  MOV MA             LXI H   22744Q      MOV MI     377Q     RET
005503Q  LXI H   24203Q     MOV AM              ORA A               RAR
005511Q  MOV MA             RET                 LXI B      440Q     CALL    4365Q
005521Q  CMA                RRC                 JNC      5534Q      LXI B    440Q
005531Q  CALL     4664Q     LXI H   23155Q      MOV MI     377Q     CALL    1444Q
005544Q  LXI H   24210Q     MOV MA              INX H               MOV MB
005552Q  LXI H   22740Q     MOV MA              INR L               MOV MB
005560Q  CALL      617Q     LXI H   24456Q      MOV MA              MOV EA
005570Q  LXI H   22740Q     MOV CM              INR L               MOV BM
005576Q  CALL     2640Q     LXI H   24210Q      MOV CM              INR L
005606Q  MOV BM             CALL     5027Q      LXI H   24456Q      MOV AM
005616Q  SUB I       33Q    RNZ                 JMP      5636Q
005624Q   040Q 124Q 040Q 116Q 040Q 111Q 072Q 040Q 040Q 044Q
005636Q  LXI B    5624Q     CALL      643Q      CALL      617Q      LXI H   24456Q
005652Q  MOV MA             SUB I     164Q      JNZ      5666Q      CALL    5465Q
005663Q  JMP      5716Q     MOV AM              SUB I     156Q      JNZ     5702Q
005674Q  CALL     5503Q     JMP      5716Q      MOV AM              SUB I    151Q
005705Q  JNZ      5716Q     LXI SP  22712Q      JMP       403Q      CALL     476Q
005721Q  RET                JMP     5733Q
005725Q   100Q 173Q 100Q 151Q 155Q 175Q
005733Q  LXI H   24170Q     MOV CM              INR L               MOV BM
005741Q  CALL     2052Q     CALL     1444Q      LXI H   24170Q      MOV MA
005753Q  INX H              MOV MB              MOV LI    221Q      MOV AM
005760Q  RRC                JNC      6043Q      LXI H   23155Q      MOV MI     377Q
005771Q  LXI H   24457Q     MOV MI      0Q      MOV AI       5Q     LXI H   24457Q
006003Q  SUB M              JC       6064Q      MOV CM              MOV BI      0Q
006012Q  LXI H    5725Q     DAD B               MOV AM              MOV EA
006020Q  LXI H   24170Q     MOV CM              INR L               MOV BM
006026Q  CALL     2640Q     LXI H   24457Q      INR M               JNZ      5776Q
006040Q  JMP      6064Q     MOV LI    210Q      MOV CM              INR L
006047Q  MOV BM             MOV LI    170Q      MOV EM              INR L
```

```
006054Q  MOV DM          CALL   2767Q  MOV LI   221Q  MOV MI   377Q
006064Q  LXI B     2Q    LHLD  24170Q  DAD B          SHLD  24172Q
006076Q  LXI H  24162Q   MOV MI    0Q  CALL    5376Q  LXI H  24206Q
006111Q  MOV MA          INX H        MOV MB         CMA
006115Q  RRC             JNC   6133Q  DCR L          MOV CM
006123Q  INR L           MOV BM       CALL    2052Q  JMP    6103Q
006133Q  MOV LI   162Q   MOV MI    0Q  RET            LXI H  24170Q
006143Q  MOV CM          INR L        MOV BM         LXI H  23110Q
006151Q  MOV MC          INX H        MOV MB         LHLD   24170Q
006157Q  MOV AM          INX H        MOV BM         LXI H  24170Q
006165Q  MOV MA          INX H        MOV MB         LXI H  23110Q
006173Q  MOV CM          INR L        MOV BM         CALL    1405Q
006201Q  LXI B     2Q    LHLD  24170Q  DAD B          SHLD   24172Q
006213Q  RET             LXI H  24460Q  MOV MC         JMP    6271Q
006223Q  LXI H  24172Q   MOV AM       ANA I     17Q  SUB I     0Q
006233Q  JNZ     6264Q   CALL   6140Q  LXI H  24170Q  MOV CM
006245Q  INR L           MOV BM       MOV CB         MOV AC
006251Q  SUB I     0Q    JNZ    6264Q  LXI SP 22712Q  JMP     403Q
006264Q  LHLD   24172Q   MOV AM       RET            CALL   6223Q
006274Q  LXI H  24461Q   MOV MA       RLC            RRC
006302Q  JNC     6317Q   LHLD  24172Q  INX H          SHLD   24172Q
006314Q  JMP     6271Q   DCR L        MOV AM         RRC
006322Q  JNC     6336Q   LHLD  24172Q  MOV MI   200Q  INX H
006333Q  SHLD   24172Q   LXI H  24461Q  MOV AM         RET
006343Q  LXI H  24464Q   MOV MC       DCR H          MOV LI   162Q
006352Q  INR M           MOV CM       MOV BI    0Q   LXI H  23164Q
006361Q  DAD B           DAD B        XCHG           LXI H  22736Q
006367Q  MOV ME          INR L        MOV MD         CALL    1444Q
006375Q  LXI H  24464Q   ORA M        MOV CA         MOV AB
006403Q  ORA I     0Q    LHLD  22736Q  MOV MC         INX H
006412Q  MOV MA          LXI H  24164Q  MOV MC         INX H
006420Q  MOV MA          DCR L        MOV AM         INR L
006424Q  MOV BM          ANA I   360Q  MOV CA         MOV AB
006431Q  ANA I   377Q     MOV BA       LXI D     2Q   MOV LC
006440Q  MOV HB          DAD D        SHLD   24164Q  LXI H  24166Q
006450Q  MOV MI    0Q    MOV AI   376Q  SUB I     4Q   MOV LI   162Q
006460Q  SUB M           JNZ    6526Q  JMP     6513Q
006467Q   116Q 123Q 040Q 123Q 124Q 101Q 103Q 113Q 040Q 117Q 126Q 105Q 122Q 106Q
006505Q   114Q 117Q 127Q 040Q 040Q 044Q
006513Q  MOV CI    11Q   LXI D   6467Q  CALL    446Q   JMP      5Q
006526Q  RET             LXI H  24465Q  MOV MC         DCR H
006534Q  MOV LI   166Q   MOV AM       RRC            JNC    6612Q
006543Q  MOV LI   162Q   MOV CM       MOV BI    0Q   LXI H  23164Q
006553Q  DAD B           DAD B        MOV AM         INX H
006557Q  MOV BM          ANA I   360Q  MOV CA         MOV AB
006564Q  ANA I   377Q     MOV EC       MOV DA         LXI H  24465Q
006573Q  MOV CM          CALL   2167Q  MOV LI   164Q  MOV MA
006602Q  INX H           MOV MB       INR L          MOV MI    0Q
006607Q  JMP     6712Q   INR H        MOV LI    65Q  MOV CM
006616Q  LHLD   24164Q   MOV MC       INX H          SHLD   24164Q
006626Q  XCHG            MOV AE       ANA I     17Q  SUB I     0Q
006634Q  JNZ     6712Q   LXI H  24164Q  MOV AM         INR L
006644Q  MOV BM          SUB I     20Q  MOV CA         MOV AB
006651Q  SBC I     0Q    INR H        MOV LI    66Q  MOV MC
006657Q  INX H           MOV MA       CALL    1444Q  LHLD   24466Q
006667Q  MOV MA          INX H        MOV MB         LXI H  24164Q
006675Q  MOV MA          INX H        MOV MB         LXI B     2Q
006703Q  LHLD   24164Q   DAD B        SHLD   24164Q  RET
006713Q  LXI H  24471Q   MOV MI    0Q  INR L          MOV MI   377Q
006723Q  MOV CI   377Q   CALL   6214Q  LXI H  24473Q  MOV MA
006734Q  SUB I    173Q   JNZ    6760Q  DCR L          MOV AM
006743Q  RRC             JNC    6751Q  DCR L          INR M
006751Q  MOV LI    72Q   MOV MI   377Q  JMP     7011Q  MOV AM
006761Q  SUB I    175Q   JNZ    7011Q  DCR L          MOV AM
006770Q  RRC             JNC    7001Q  MOV MI    0Q   JMP    7003Q
007001Q  DCR L           DCR M        MOV LI    71Q  MOV AM
007006Q  SUB I     0Q    RZ           MOV LI    73Q  MOV CM
007014Q  CALL   6527Q    JMP    6723Q  RET            MOV CI    0Q
007025Q  CALL   6214Q    LXI H  24477Q  MOV MA         SUB I    100Q
007036Q  JNZ    7126Q    MOV CI   377Q  CALL   6214Q  LXI H  23112Q
007051Q  MOV MA          MOV CI    0Q  CALL   6214Q  LXI H  24477Q
007062Q  MOV MA          SUB I    173Q  JNZ    7111Q  MOV CI   377Q
007072Q  CALL   6214Q    LXI H  23112Q  MOV MA         MOV CI    2Q
007103Q  CALL   6343Q    JMP    7165Q  MOV CI   100Q  CALL   6527Q
007116Q  MOV CI   100Q   CALL   6527Q  JMP    7165Q  MOV AM
007127Q  SUB I    173Q   JNZ    7160Q  MOV CI   377Q  CALL   6214Q
007141Q  LXI H  23112Q   MOV MA       MOV AI     2Q  ORA I     4Q
007151Q  MOV CA          CALL   6343Q  JMP    7165Q  MOV CI   100Q
007162Q  CALL   6527Q    RET            LXI H  24500Q  MOV MC
007172Q  INX H           MOV MB       MOV LI   113Q  MOV MI    0Q
007200Q  MOV CI    0Q    DCR C        MOV LI   111Q  MOV MC
007206Q  MOV LI   100Q   MOV CM       INR L          MOV BM
007213Q  MOV CB          MOV AC       RLC            RRC
007217Q  JNC    7246Q    MOV LI   113Q  MOV MI   377Q  XRA A
007227Q  MOV LI   100Q   SUB M        INR L          MOV CA
007234Q  MOV AI    0Q    SBC M        DCR L          MOV MC
007241Q  INX H           MOV MA       JMP    7274Q  DCR L
007247Q  MOV AM          INR L        MOV BM         SUB I     0Q
007254Q  MOV CA          MOV AB       SBC I     0Q   ORA C
007261Q  JNZ    7274Q    MOV LI   111Q  MOV MI    0Q  MOV LI   103Q
007272Q  MOV MI    60Q   XRA A        LXI H  24500Q  SUB M
007301Q  INR L           MOV CA       MOV AI     0Q  SBC M
007306Q  JNC    7466Q    MOV LI   111Q  INR M          MOV CM
007315Q  MOV BI    0Q    MOV LI   103Q  DAD B          XCHG
```

```
007323Q  LXI H  22734Q
007331Q  MOV EI    12Q
007341Q  INR L
007347Q  CMA
007353Q  MOV EA
007362Q  PUSH H
007370Q  POP H
007374Q  MOV CA
007400Q  MOV AL
007404Q  RAL
007410Q  JNZ    7362Q
007416Q  MOV DA
007422Q  RET
007432Q  DAD B
007440Q  MOV EI    12Q
007450Q  INR L
007460Q  MOV MC
007466Q  MOV LI   113Q
007475Q  MOV LI   111Q
007503Q  MOV LI   103Q
007513Q  LXI H  24210Q
007521Q  LXI H  23155Q
007533Q  LXI H  24511Q
007541Q  JC     7600Q
007547Q  SUB M
007555Q  DAD B
007563Q  MOV CM
007571Q  LXI H  24512Q
007601Q  LXI H  24514Q
007607Q  INR L
007615Q  MOV LI   122Q
007626Q  LXI H  24514Q
007634Q  ANA I    360Q
007642Q  MOV BA
007652Q  SUB I    201Q
007662Q  JNZ    7673Q
007673Q  MOV AM
007701Q  JNC    7752Q
007711Q  INR L
007715Q  MOV EA
007721Q  DCR C
007731Q  XCHG
007737Q  LXI H  24521Q
007746Q  DAD B
007756Q  MOV AM
007764Q  MOV LI   116Q
007771Q  MOV AI    0Q
007776Q  RET
010003Q  MOV BM
010011Q  MOV AI    71Q
010020Q  SUB I    60Q
010026Q  JNC    10035Q
010035Q  MOV AM
010044Q  MOV MC
010053Q  MOV AM
010060Q  ADD I   127Q
010070Q  CALL   5376Q
010100Q  MOV MB
010106Q  INR H
010115Q  MOV LI   163Q
010122Q  INR H
010127Q  JMP    10332Q
010142Q  MOV CM
010150Q  LXI H  24527Q
010156Q  JNC    10172Q
010167Q  MOV AI   377Q
010176Q  CALL   2366Q
010206Q  RRC
010215Q  JZ     10172Q
010223Q  INR L
010231Q  RET
010237Q  INR L
010245Q  INX H
010253Q  CALL   2366Q
010265Q  MOV CM
010273Q  MOV MC
010300Q  MOV BI    0Q
010311Q  RLC
010317Q  SUB I    201Q
010326Q  MOV AI   377Q
010335Q  MOV CM
010341Q  MOV LI   130Q
010346Q  DCR H
010353Q  MOV BM
010363Q  XRA A
010371Q  INR L
010377Q  INX H
010405Q  JNC    10443Q
010415Q  MOV BM
010422Q  INX H
010430Q  INR L
010436Q  INX H
010443Q  LXI H  24526Q
010454Q  XRA A

MOV ME
MOV DI     0Q
MOV BM
MOV DA
INX D
DAD D
PUSH A
MOV AB
RAL
MOV HA
ORA A
MOV AL
CALL    7346Q
XCHG
MOV DI     0Q
MOV BM
INX H
MOV AM
INR M
DAD B
MOV MA
MOV MI   377Q
MOV AM
DCR L
MOV CA
MOV AM
INR L
INR M
MOV MC
MOV MI     0Q
MOV MI     0Q
MOV AM
MOV CA
CALL    2366Q
JZ      7755Q
INR L
SUB I     60Q
MOV CI     3Q
MOV DM
MOV AD
JNZ     7713Q
LHLD   24516Q
MOV CM
SHLD   24516Q
RRC
SUB M
SBC M
MOV LI   116Q
RET
MOV LI   125Q
MOV CM
MOV LI   125Q
LXI H  23154Q
INR L
MOV MA
MOV AM
RET
LXI H  24526Q
JNC    10222Q
XRA A
SUB M
RET
MOV BM
MOV MB
LXI H  23112Q
INR L
INX H
CALL    2366Q
JNZ    10262Q
RET
INR L
MOV MC
MOV LI   206Q
CALL    3236Q
RET
MOV BM
MOV MB
LXI H  24530Q
DCR H
MOV MB
MOV BM
MOV MB
MOV AM
RET

INR L
LXI H  24500Q
JMP     7423Q
MOV AE
LXI H     0Q
JNC     7370Q
MOV AC
RAL
MOV LA
POP A
MOV AH
RAR
LXI B     60Q
LHLD   22734Q
LXI H  24500Q
CALL    7346Q
MOV MB
RRC
MOV CM
MOV MI    55Q
INX H
LXI H  24512Q
INR L
MOV AM
MOV BI     0Q
MOV EA
MOV BM
JNZ     7533Q
INX H
INX H
LXI H  23153Q
INR L
MOV AB
LXI H  24521Q
MOV AM
MOV MI   377Q
MOV MA
MOV LI   116Q
MOV AE
RAL
LHLD   24516Q
DAD D
MOV BI     0Q
JMP     7626Q
JNC     7777Q
INR L
MOV BA
MOV AM
LXI H  24523Q
JC     10023Q
MOV AM
SUB I     67Q
RET
SUB I     12Q
RET
LXI H  24525Q
MOV MA
RRC
INR M
DCR L
MOV AM
MOV MI   377Q
MOV BM
RLC
SUB I    201Q
MOV CI     0Q
MOV MA
MOV AM
RET
JZ     10137Q
LXI H  24530Q
LXI H  23156Q
MOV CI     0Q
MOV MA
MOV BM
MOV MB
LXI H  24526Q
JNC    10326Q
XRA A
RET
MOV BM
INX H
MOV CM
RRC
LXI H  24530Q
LXI H  23156Q
CALL   10132Q
MOV CM
MOV LI   176Q
LXI H  23156Q
LXI H  24200Q
MOV AI   377Q
SUB I    201Q
CALL   10233Q

MOV MD
MOV CM
MOV AD
CMA
MOV AI    21Q
XTHL
RAL
MOV BA
MOV AH
DCR A
RAR
MOV EA
XCHG
MOV ME
MOV CM
LXI H  24500Q
JMP     7274Q
JNC     7510Q
MOV BI     0Q
CALL    1444Q
MOV MB
MOV MI     0Q
SUB M
INR L
LXI H  24210Q
CALL    2640Q
RET
MOV MB
MOV MI     0Q
MOV MI   377Q
MOV BM
ANA I    377Q
MOV MA
SUB I     55Q
JMP     7626Q
SUB I     12Q
MOV EM
ADD A
MOV DA
DAD D
XCHG
XCHG
INR L
XRA A
MOV CA
MOV AC
INR L
MOV MC
MOV AM
SUB I    107Q
RET
LXI H  24524Q
JNC    10057Q
MOV AM
MOV MI     1Q
INX H
JNC    10115Q
JMP    10070Q
MOV MC
RET
LXI H  24206Q
CALL    2524Q
RRC
JNZ    10137Q
MOV BI     0Q
RLC
SUB I    200Q
MOV AM
XRA A
MOV CM
MOV MC
MOV BI     0Q
LXI H  23156Q
LXI H  24530Q
MOV CI     0Q
MOV BI     0Q
LXI H  24174Q
INR H
MOV MB
INR L
JNC    10365Q
MOV CM
MOV MC
RRC
INR L
MOV MC
MOV CM
MOV MC
RET
JNZ    10456Q
CMA
```

```
0110462Q  RRC              JNC    10365Q  XRA A              RET
0110470Q  RET              CALL    5376Q  LXI H   24206Q     MOV MA
0110500Q  INX H                           CMA                RRC
0110504Q  JNC    10622Q    DCR L          MOV CM             INR L
0110512Q  MOV BM           CALL    4365Q  CMA                RRC
0110520Q  JNC    10534Q    LXI H   24206Q  MOV CM             INR L
0110530Q  MOV BM           CALL    4664Q  CALL     5376Q     LXI H   24206Q
0110542Q  MOV MA           INX H          MOV MB             RRC
0110546Q  JNC    10561Q    CALL    1444Q  MOV CA             CALL    5027Q
0110560Q  RET              DCR L          MOV CM             INR L
0110564Q  MOV BM           CALL    5027Q  LXI H   24162Q     MOV CM
0110574Q  MOV BI      0Q   LXI H   23164Q  DAD B              DAD B
0110603Q  XCHG             LXI H   24216Q  MOV CM             INR L
0110611Q  MOV BM           MOV AC         STAX D             INX D
0110615Q  MOV AB           STAX D         JMP     10471Q     RET
0110623Q  CALL    5376Q    LXI H   24206Q  MOV MA             INX H
0110633Q  MOV MB           CMA            RRC                JNC    11050Q
0110641Q  DCR L            MOV CM         INR L              MOV BM
0110645Q  CALL    4365Q    CMA            RRC                JNC    10675Q
0110655Q  LXI H   24206Q   MOV CM         INR L              MOV BM
0110663Q  CALL    4664Q    CALL    1444Q  MOV CA             CALL    5027Q
0110675Q  CALL    5376Q    LXI H   24206Q  MOV MA             INX H
0110705Q  MOV MB           RRC            RC                 LXI H   23153Q
0110713Q  MOV MI     377Q  LXI H   24206Q  MOV CM             INR L
0110722Q  MOV BM           CALL    2366Q  LXI H   24532Q     MOV MA
0110732Q  SUB I      201Q  JZ     10623Q  LXI B      4Q     LHLD    24440Q
0110745Q  DAD B            MOV AM         INX H              MOV BM
0110751Q  MOV EA           MOV DB         LXI H   24532Q     MOV CM
0110757Q  CALL    2167Q    LXI H   23160Q  MOV MA             INX H
0110767Q  MOV MB           DCR L          MOV AM             INR L
0110773Q  MOV BM           ANA I     360Q  MOV CA             MOV AB
0111000Q  ANA I      377Q  INR L          MOV MC             INX H
0111005Q  MOV MA           MOV LI    155Q  MOV MI       0Q   MOV CI       0Q
0111014Q  MOV BI      0Q   CALL    2366Q  LXI H   24532Q     MOV MA
0111025Q  SUB I      201Q  JZ     11045Q  MOV CI       0Q   MOV BI       0Q
0111036Q  MOV EM           CALL    2640Q  JMP     11012Q     JMP    10623Q
0111050Q  RET              JMP    11144Q  LXI H   24535Q     MOV MC
0111060Q  LHLD    23160Q   MOV MC         INX H              SHLD    23160Q
0111070Q  XCHG             MOV AE         ANA I      17Q     SUB I        0Q
0111076Q  JNZ     11143Q   LXI H   23160Q  MOV AM             INR L
0111106Q  MOV BM           SUB I      20Q  MOV CA             MOV AB
0111113Q  SBC I       0Q   LXI H   24536Q  MOV MC             INX H
0111122Q  MOV MA           LHLD    24536Q  MOV AM             INX H
0111130Q  MOV BM           MOV CA         LXI D       2Q     MOV LC
0111136Q  MOV HB           DAD D          SHLD    23160Q     RET
0111144Q  CALL    5376Q    MOV CA         CALL    4365Q      CMA
0111154Q  RRC              RC             MOV CI       0Q    DCR C
0111161Q  LXI H   24533Q   MOV MC         CALL    5376Q      LXI H   24206Q
0111173Q  MOV MA           INX H          MOV MB             CMA
0111177Q  RRC              JNC    11417Q  INR H              MOV LI     133Q
0111206Q  INR M            LXI B      4Q  LHLD    24440Q     DAD B
0111216Q  MOV AM           INX H          MOV BM             MOV CA
0111222Q  MOV LC           MOV HB         INX H              SHLD    24174Q
0111230Q  CALL    10127Q   RRC            JNC     11165Q     LXI H   24176Q
0111242Q  MOV CM           INR L          MOV BM             LXI H   23156Q
0111250Q  MOV MC           INX H          MOV MB             MOV CI       0Q
0111255Q  MOV BI      0Q   CALL    2366Q  LXI H   23112Q     MOV MA
0111266Q  MOV LI     156Q  MOV CM         INR L              MOV BM
0111273Q  INR L            MOV MC         INX H              MOV MB
0111277Q  DCR L            MOV AM         INR L              MOV BM
0111303Q  LXI H   24200Q   SUB M          INR L              MOV CA
0111311Q  MOV AB           SBC M          ORA C              ADD I      377Q
0111316Q  SBC A            INR H          MOV LI     134Q    MOV MA
0111323Q  DCR L            MOV AM         ORA I      300Q    MOV CA
0111330Q  CALL    11054Q   LXI H   24534Q  MOV AM             RRC
0111340Q  JNC     11401Q   LXI H   23160Q  MOV AM             INR L
0111350Q  MOV BM           LXI H   24200Q  SUB M              INR L
0111356Q  MOV CA           MOV AB         SBC M              ORA C
0111362Q  ADD I      377Q  SBC A          INR H              MOV LI     134Q
0111370Q  MOV MA           MOV CI    200Q  CALL    11054Q     JMP    11333Q
0111401Q  DCR H            MOV LI    200Q  MOV CM             INR L
0111406Q  MOV BM           MOV LI    174Q  MOV MC             INX H
0111413Q  MOV MB           JMP    11230Q  RET                JMP    11631Q
0111423Q  LXI H   24544Q   MOV AM         INR L              MOV BM
0111431Q  SUB I       0Q   MOV CA         MOV AB             SBC I        0Q
0111437Q  ORA C            JNZ    11445Q  XRA A              RET
0111445Q  MOV CI      0Q   MOV BI      0Q  CALL     2366Q     LXI H   24556Q
0111457Q  MOV MA           SUB I     201Q  JZ     11470Q     MOV AI     377Q
0111467Q  RET              XRA A          RET                LXI H   24560Q
0111475Q  MOV MI      0Q   INX H          MOV MI       0Q    LXI H   23153Q
0111505Q  MOV MI     377Q  LXI B      4Q  LHLD    24440Q     DAD B
0111516Q  MOV AM           INX H          MOV BM             LXI H   23110Q
0111524Q  MOV MA           INX H          MOV MB             LXI H   23110Q
0111532Q  MOV CM           INR L          MOV BM             CALL    2366Q
0111540Q  LXI H   24556Q   MOV MA         SUB I      201Q    ADD I      377Q
0111550Q  SBC A            MOV HI     46Q MOV CA             MOV AM
0111555Q  INR L            MOV DM         LXI H   24546Q     SUB M
0111563Q  INR L            MOV EA         MOV AD             SBC M
0111567Q  ORA E            ADD I     377Q SBC A              ANA C
0111574Q  RRC              JNC    11623Q  MOV LI     156Q    MOV AM
0111603Q  RLC              CMA            RRC                JNC    11527Q
0111611Q  LHLD    24560Q   INX H          SHLD    24560Q     JMP    11527Q
0111623Q  MOV LI     160Q  MOV AM         INR L              MOV BM
0111630Q  RET              LXI B     406Q CALL     4365Q     CMA
```

```
011640Q  RRC          JNC      11652Q  LXI B      406Q  CALL      4664Q
011652Q  CALL   1444Q  LXI H    24540Q  MOV MA           INX H
011662Q  MOV MB       MOV CA           CALL     5027Q  CALL      5376Q
011672Q  LXI H  24204Q MOV MA           INX H           MOV MB
011700Q  CALL   5376Q  LXI H    24542Q  MOV MA           INX H
011710Q  MOV MB       CALL      5376Q  LXI H   24212Q  MOV MA
011720Q  INX H        MOV MB           CALL     5376Q  LXI H   24214Q
011730Q  MOV MA       INX H            MOV MB          MOV LI    204Q
011735Q  MOV CM       INR L            MOV BM          CALL     4365Q
011743Q  CMA          RRC              RC              LXI H   24542Q
011751Q  MOV AM       RRC              JNC     11775Q  MOV LI    146Q
011760Q  MOV MI    0Q INX H            MOV MI     0Q   CALL     11472Q
011770Q  MOV CA       CALL      7166Q  RET             MOV CM
011776Q  INR L        MOV BM           CALL     7601Q  LXI H   24544Q
012006Q  MOV MA       INX H            MOV MB          MOV LI    157Q
012013Q  MOV MI    0Q LHLD     24440Q  INX H           INX H
012022Q  MOV AM       INX H            MOV BM          LXI H   23156Q
012030Q  MOV MA       INX H            MOV MB          LXI H   24550Q
012036Q  MOV MA       INX H            MOV MB          LXI H   23155Q
012044Q  MOV MI   377Q CALL     11423Q  RRC             JNC     12127Q
012055Q  MOV LI   156Q MOV AM           RLC             CMA
012062Q  RRC          JNC       12046Q  LHLD    24544Q  DCX H
012072Q  SHLD  24544Q LXI H    23156Q  MOV CM          INR L
012102Q  MOV BM       LXI H    24550Q  MOV MC          INX H
012110Q  MOV MB       MOV LI     140Q  MOV CM          INR L
012115Q  MOV BM       MOV LI     156Q  MOV EM          CALL     2640Q
012124Q  JMP    12046Q LHLD     24440Q  INX H          INX H
012134Q  XCHG         LXI H    24550Q  MOV CM          INR L
012142Q  MOV BM       MOV AC           STAX D          INX D
012146Q  MOV AB       STAX D           DCR H           MOV LI    212Q
012153Q  MOV CM       INR L            MOV BM          MOV LI    210Q
012160Q  MOV MC       INX H            MOV MB          INR H
012164Q  MOV LI   144Q MOV AM           INR L           MOV BM
012171Q  SUB I     0Q MOV CA           MOV AB          SBC I      0Q
012177Q  ORA C        ADD I      377Q  SBC A           MOV LI    157Q
012205Q  ORA M        RRC              JNC     12225Q  DCR H
012213Q  MOV LI   214Q MOV CM          INR L           MOV BM
012220Q  MOV LI   210Q MOV MC          INX H           MOV MB
012225Q  LXI H  24212Q MOV AM          RRC             JNC     12261Q
012235Q  CALL   1444Q  LXI H    24210Q  MOV MA          INX H
012245Q  MOV MB       MOV EA           MOV DB          INR H
012251Q  MOV LI    140Q MOV CM         INR L           MOV BM
012256Q  CALL   2767Q  RET             CALL     5376Q  LXI H   24204Q
012270Q  MOV MA       INX H            MOV MB          CALL     5376Q
012276Q  LXI H  24206Q MOV MA          INX H           MOV MB
012304Q  CALL   5376Q  LXI H    24212Q  MOV MA          INX H
012314Q  MOV MB       CALL      5376Q  LXI H   24214Q  MOV MA
012324Q  INX H        MOV MB           MOV LI    210Q  MOV MA
012331Q  INX H        MOV MB           LXI B     420Q  CALL     4365Q
012341Q  CMA          RRC              JNC     12354Q  LXI B     420Q
012351Q  CALL   4664Q  CALL      1444Q  LXI H   24562Q  MOV MA
012363Q  INX H        MOV MB           MOV CA          CALL     5027Q
012371Q  LXI H  24204Q MOV CM          INR L           MOV BM
012377Q  CALL   4365Q  CMA              RRC             RC
012405Q  LHLD  24440Q  INX H           INX H           MOV AM
012413Q  INX H        MOV BM           LXI H   24174Q  MOV MA
012421Q  INX H        MOV MB           LXI H   23153Q  MOV MI     0Q
012430Q  MOV LI    155Q MOV MI     377Q  CALL    10127Q  RRC
012440Q  JNC    12555Q LXI H    24174Q  MOV CM          INR L
012450Q  MOV BM       LXI H    23156Q  MOV MC          INX H
012456Q  MOV MB       LXI H    23156Q  MOV AM          INR L
012464Q  MOV BM       LXI H    24176Q  SUB M           INR L
012472Q  MOV CA       MOV AB           SBC M           ORA C
012476Q  JZ     12540Q MOV CI      0Q  MOV BI      0Q  CALL     2366Q
012510Q  LXI H  24565Q MOV MA          RLC             CMA
012516Q  RRC          JNC      12457Q  MOV LI    162Q  MOV CM
012525Q  INR L        MOV BM           MOV LI    165Q  MOV EM
012532Q  CALL   2640Q  JMP      12457Q  MOV LI    212Q  MOV CM
012543Q  INR L        MOV BM           MOV LI    210Q  MOV MC
012550Q  INX H        MOV MB           JMP     12666Q  LXI H   24174Q
012560Q  MOV CM       INR L            MOV BM          LXI H   23156Q
012566Q  MOV MC       INX H            MOV MB          LXI H   24200Q
012574Q  MOV MC       INX H            MOV MB          LXI H   23155Q
012602Q  MOV MI   377Q MOV CI      0Q  MOV BI      0Q  CALL     2366Q
012613Q  LXI H  24565Q MOV MA          SUB I     201Q  JZ      12666Q
012624Q  MOV AM       RLC              CMA             RRC
012630Q  JNC    12604Q LXI H    23156Q  MOV CM          INR L
012640Q  MOV BM       LXI H    24200Q  MOV MC          INX H
012646Q  MOV MB       INR H            MOV LI    162Q  MOV CM
012653Q  INR L        MOV BM           MOV LI    165Q  MOV EM
012660Q  CALL   2640Q  JMP      12604Q  LHLD    24440Q  INX H
012672Q  INX H        XCHG             LXI H   24200Q  MOV CM
012700Q  INR L        MOV BM           MOV AC          STAX D
012704Q  INX D        MOV AB           STAX D          MOV LI    212Q
012711Q  MOV AM       RRC              JNC     12742Q  CALL     1444Q
012721Q  LXI H  24210Q MOV MA          INX H           MOV MB
012727Q  MOV EA       MOV DB           INR H           MOV LI    162Q
012734Q  MOV CM       INR L            MOV BM          CALL     2767Q
012742Q  RET          CALL      5376Q  MOV CA          CALL     4365Q
012752Q  CMA          RRC              RC              LHLD    24440Q
012760Q  INX H        INX H            PUSH H          LXI B       4Q
012766Q  LHLD  24440Q DAD B            MOV AM          INX H
012774Q  MOV BM       MOV CA           MOV LC          MOV HB
013000Q  INX H        XCHG             POP H           MOV ME
```

```
0130040  INX H          MOV MD          RET             CALL    5376Q
0130120  LXI H  24566Q  MOV MA          INX H           MOV MB
0130200  CALL    5376Q  LXI H  24570Q  MOV MA          INX H
0130300  MOV MB         RRC             RC              CALL    5376Q
0130360  LXI H  24572Q  MOV MA          INX H           MOV MB
0130440  RRC           JNC     13067Q  MOV LI    170Q  MOV CM
0130530  INR L         MOV BM          DCR H           MOV LI    210Q
0130600  MOV MC        INX H           MOV MB          RET
0130640  JMP     13020Q  MOV LI    166Q  MOV CM        INR L
0130730  MOV BM        INR L           MOV EM          INR L
0130770  MOV DM        CALL    3062Q   RRC             JNC     13124Q
0131070  LXI H  24572Q  MOV CM        INR L           MOV BM
0131150  DCR H         MOV LI    210Q  MOV MC          INX H
0131220  MOV MB        RET             JMP     13020Q  RET
0131300  CALL    5376Q  MOV CA        CALL     7601Q  MOV CA
0131400  LXI H  100000Q DAD B          XCHG            LXI H  22732Q
0131500  MOV ME         INR L           MOV MD          CALL    5376Q
0131560  MOV CA        CALL     7601Q  MOV CA          LXI H  100000Q
0131660  DAD B         XCHG            LXI H  22732Q   MOV AM
0131740  INR L         MOV BM          SUB E           MOV CA
0132000  MOV AB        SBC D           JC      13217Q  CALL    5376Q
0132100  LXI H  24210Q  MOV MA          INX H           MOV MB
0132160  RET           CALL    5376Q   LXI H  24210Q   MOV MA
0132260  INX H         MOV MB          CALL    5376Q   LXI H  24210Q
0132360  MOV MA        INX H           MOV MB          RET
0132420  CALL    5376Q  ANA I     360Q  MOV CA          MOV AB
0132510  ANA I     377Q  LXI H  24204Q  MOV MC          INX H
0132600  MOV MA        CALL    5376Q   LXI H  24212Q   MOV MA
0132700  INX H         MOV MB          MOV LI    210Q  MOV MA
0132750  INX H         MOV MB          CALL    5376Q   LXI H  24214Q
0133050  MOV MA        INX H           MOV MB          MOV LI    204Q
0133120  MOV CM        INR L           MOV BM          CALL    4365Q
0133200  RRC           JNC     13362Q  LXI B       4Q  LHLD    24440Q
0133320  DAD B         MOV AM          INX H           MOV BM
0133360  MOV CA        CALL    3236Q   CMA             RRC
0133440  JNC     13362Q  LXI H  24214Q  MOV CM          INR L
0133540  MOV BM        MOV LI    210Q  MOV MC          INX H
0133610  MOV MB        RET             LXI H  24574Q   MOV MC
0133670  CALL    10063Q  MOV LI    174Q  SUB M         SUB I       2Q
0133770  INR L         MOV MA          DCR L           MOV AM
0134030  DCR H         MOV LI    162Q  ADD M           MOV MA
0134100  CALL    5376Q  ANA I     360Q  MOV CA          MOV AB
0134170  ANA I     377Q  MOV BA          CALL    4365Q  CMA
0134260  RRC           RC              CALL    1444Q   LXI H  24210Q
0134360  MOV MA        INX H           MOV MB          LXI H  23153Q
0134440  MOV MI    377Q  MOV LI    155Q  MOV MI    377Q  LXI B       4Q
0134550  LHLD    24440Q  DAD B         MOV AM          INX H
0134630  MOV BM        LXI H  23110Q   MOV MA          INX H
0134710  MOV MB        LXI H  23110Q   MOV CM          INR L
0134770  MOV BM        CALL    2366Q   LXI H  24576Q   MOV MA
0135070  SUB I     201Q  JZ      13701Q  MOV AM          RLC
0135160  CMA           RRC             JNC     13543Q  DCR H
0135240  MOV LI    210Q  MOV CM        INR L           MOV BM
0135310  INR H         MOV LI    176Q  MOV EM          CALL    2640Q
0135400  JMP     13472Q  MOV AM        RLC             RLC
0135460  RRC           JNC     13676Q  MOV AM          XRA I       300Q
0135550  INR A         MOV LI    200Q  MOV MA          MOV LI    175Q
0135630  MOV CA        MOV AM          SUB C           JC      13676Q
0135710  MOV LI    200Q  MOV AM        DCR H           MOV LI    162Q
0135770  ADD M         MOV CA          MOV BI       0Q  LXI H  23164Q
0136060  DAD B         DAD B           MOV AM          INX H
0136120  MOV BM        LXI H  23110Q   MOV MA          INX H
0136200  MOV MB        MOV LI    154Q  MOV MI    377Q  LXI H  23110Q
0136300  MOV CM        INR L           MOV BM          CALL    2524Q
0136360  LXI H  24577Q  MOV MA          SUB I     201Q  JZ      13676Q
0136470  MOV AM        RLC             CMA             RRC
0136530  JNC     13625Q  DCR H         MOV LI    210Q  MOV CM
0136620  INR L         MOV BM          INR H           MOV LI    177Q
0136670  MOV EM        CALL    2640Q   JMP     13625Q  JMP     13472Q
0137010  RET           CALL    5376Q   LXI H  24210Q   MOV MA
0137110  INX H         MOV MB          DCR L           MOV CM
0137150  INR L         MOV BM          CALL    3236Q   LXI H  24221Q
0137250  MOV MA        LXI SP  22712Q  JMP      403Q   LXI H  24601Q
0137370  MOV MI     0Q  CALL    5376Q   LXI H  24206Q   MOV MA
0137500  INX H         MOV MB          CMA             RRC
0137540  JNC     14010Q  INR H         MOV LI    201Q  MOV AM
0137630  RRC           CC        476Q  LXI H  24601Q   MOV MI    377Q
0137740  DCR H         MOV LI    206Q  MOV CM          INR L
0140010  MOV BM        CALL    3334Q   JMP     13741Q  RET
0140110  CALL    5376Q  MOV CA        CALL     7601Q  LXI H  22745Q
0140230  MOV MA        RET             JMP     14142Q  LXI H  23101Q
0140330  MOV AM        SUB I     200Q  JNZ     14116Q  MOV MI     0Q
0140430  MOV LI     74Q  MOV CM        INR L           MOV BM
0140500  MOV EC        MOV DB          MOV CI     24Q  CALL     462Q
0140570  SUB I      0Q  JZ      14116Q  JMP     14102Q
0140670  104Q 111Q 123Q 113Q 040Q 105Q 117Q 106Q 040Q 040Q 044Q
0141020  LXI B  14067Q  CALL     643Q   LXI H  24603Q   MOV MI     32Q
0141150  RET           LXI H  23101Q   MOV CM          MOV BI     0Q
0141240  LHLD    23076Q  DAD B         MOV AM          LXI H  24603Q
0141340  MOV MA        LXI H  23101Q   INR M           RET
0141420  CALL     1444Q  LXI H  24210Q  MOV MA          INX H
0141520  MOV MB        LXI H  23155Q   MOV MI    377Q  DCR H
0141610  MOV LI     347Q  MOV AM        RRC             JNC     14301Q
0141700  CALL    5376Q  LXI H  24206Q  MOV MA          INX H
```

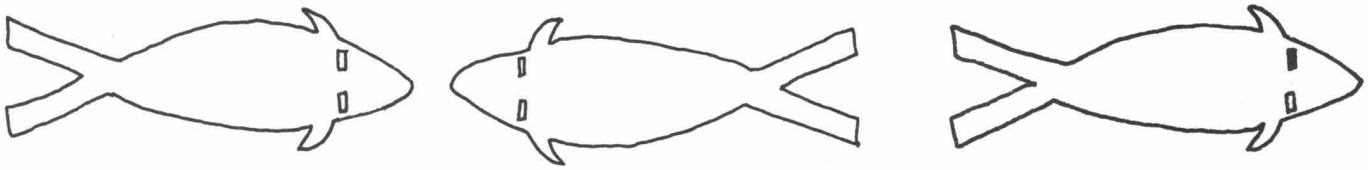
```
014200Q  MOV MB          CMA             RRC             JNC     14215Q
014206Q  DCR L           MOV CM          INR L           MOV BM
014212Q  CALL    3334Q   LXI B   22750Q  MOV EC          MOV DB
014222Q  MOV CI    12Q   CALL     446Q   CALL     476Q   LXI H   24602Q
014235Q  MOV MI     1Q   LXI H   22750Q  INX H           MOV AM
014244Q  LXI H   24602Q   SUB M           JC      14411Q  INR M
014254Q  MOV CM          MOV BI      0Q  LXI H   22750Q  DAD B
014263Q  MOV AM          MOV EA          LXI H   24210Q  MOV CM
014271Q  INR L           MOV BM          CALL     2640Q  JMP     14237Q
014301Q  LXI H   24604Q   MOV MI    377Q  CALL    14030Q  LXI H   24603Q
014314Q  MOV AM          LXI H   23103Q  SUB M           RZ
014322Q  LXI H   24603Q   MOV AM          SUB I      32Q  JNZ     14341Q
014333Q  LXI H   22747Q   MOV MI    377Q  RET             MOV AM
014342Q  SUB I     140Q   JNZ     14356Q  INR L           MOV AM
014351Q  CMA             MOV MA          JMP     14306Q  MOV AM
014357Q  SUB I      40Q  SBC A           CMA             INR L
014364Q  ANA M           RRC             JNC     14406Q  DCR H
014372Q  MOV LI    210Q  MOV CM          INR L           MOV BM
014377Q  INR H           MOV LI    203Q  MOV EM          CALL     2640Q
014406Q  JMP     14306Q  RET             CALL     1444Q  LXI H   24210Q
014420Q  MOV MA          INX H           MOV MB          CALL     5376Q
014426Q  LXI H   23110Q   MOV MA          INX H           MOV MB
014434Q  MOV LI    153Q  MOV MI    377Q  MOV LI     155Q  MOV MI    377Q
014444Q  LXI H   23110Q   MOV CM          INR L           MOV BM
014452Q  CALL    2366Q   LXI H   24605Q  MOV MA          MOV CI      0Q
014463Q  MOV BI     0Q   CALL     2366Q  LXI H   24606Q  MOV MA
014474Q  DCR L           MOV AM          SUB I      201Q SUB I      1Q
014502Q  SBC A           INR L           MOV CA          MOV AM
014506Q  SUB I     201Q  SUB I       1Q  SBC A           ORA C
014514Q  RRC             RC              DCR L           MOV CM
014520Q  CALL    10005Q  ADD A           ADD A           ADD A
014526Q  ADD A           MOV CA          PUSH B          LXI H   24606Q
014534Q  MOV CM          CALL    10005Q  POP D           ORA E
014542Q  MOV EA          LXI H   24210Q  MOV CM          INR L
014550Q  MOV BM          CALL     2640Q  JMP     14444Q  RET
014560Q  CALL    5376Q   LXI H   24206Q  MOV MA          INX H
014570Q  MOV MB          RRC             RC              LXI H   23153Q
014576Q  MOV MI    377Q  MOV LI     155Q  MOV MI     377Q  CALL     1444Q
014607Q  LXI H   24210Q   MOV MA          INX H           MOV MB
014615Q  LXI H   24206Q   MOV CM          INR L           MOV BM
014623Q  CALL    2366Q   LXI H   24607Q  MOV MA          SUB I      201Q
014634Q  JZ      14711Q  MOV AM          RLC             RLC
014642Q  RLC             RLC             ANA I      17Q  MOV CA
014647Q  CALL    10041Q  MOV EA          LXI H   24210Q  MOV CM
014657Q  INR L           MOV BM          CALL     2640Q  LXI H   24607Q
014667Q  MOV AM          ANA I      17Q  MOV CA          CALL    10041Q
014676Q  MOV EA          MOV CI      0Q  MOV BI       0Q  CALL     2640Q
014706Q  JMP     14615Q  RET             CALL     5376Q  ANA I      360Q
014717Q  MOV CA          MOV AB          ANA I    377Q   MOV BA
014724Q  CALL    3530Q   RET             LXI B    426Q   CALL     4365Q
014736Q  CMA             RRC             JNC     14751Q  LXI B    426Q
014746Q  CALL    4664Q   CALL     5376Q  LXI H   24610Q  MOV MA
014760Q  INX H           MOV MB          CALL     5376Q  LXI H   24212Q
014770Q  MOV MA          INX H           MOV MB          CALL     5376Q
014776Q  LXI H   24214Q   MOV MA          INX H           MOV MB
015004Q  CALL    14412Q  LXI H   23153Q  MOV MI    377Q  LXI H   24210Q
015017Q  MOV CM          INR L           MOV BM          CALL     2366Q
015025Q  LXI H   23113Q   MOV MA          SUB I      201Q JZ      15042Q
015036Q  MOV CM          MOV LI    102Q  MOV MC          LXI H   24210Q
015045Q  MOV CM          INR L           MOV BM          CALL     2052Q
015053Q  LXI H   24610Q   MOV CM          INR L           MOV BM
015061Q  CALL    3606Q   RRC             JNC     15117Q  LXI H   24210Q
015073Q  MOV CM          INR L           MOV BM          CALL     5027Q
015101Q  LXI H   24212Q   MOV CM          INR L           MOV BM
015107Q  MOV LI    210Q  MOV MC          INX H           MOV MB
015114Q  JMP     15143Q  LXI H   24210Q  MOV CM          INR L
015124Q  MOV BM          CALL     5027Q  LXI H   24214Q  MOV CM
015134Q  INR L           MOV BM          MOV LI     210Q MOV MC
015141Q  INX H           MOV MB          RET             CALL     5376Q
015147Q  LXI H   24206Q   MOV MA          INX H           MOV MB
015155Q  CMA             RRC             JNC     15200Q  DCR L
015163Q  MOV CM          INR L           MOV BM          CALL     4365Q
015171Q  RRC             CC      4602Q   JMP     15144Q  RET
015201Q  LXI H   24612Q   MOV MI      0Q  INX H           MOV MI      0Q
015211Q  CALL    5376Q   LXI H   24206Q  MOV MA          INX H
015221Q  MOV MB          CMA             RRC             JNC     15256Q
015227Q  DCR L           MOV CM          INR L           MOV BM
015233Q  CALL    7601Q   MOV CA          LXI H   24612Q  MOV EM
015243Q  INR L           MOV DM          MOV LC          MOV HB
015247Q  DAD D           SHLD    24612Q  JMP     15211Q  INR H
015257Q  MOV LI    212Q  MOV CM          INR L           MOV BM
015264Q  CALL    7166Q   RET             CALL     5376Q  MOV CA
015274Q  CALL    7601Q   LXI H   22730Q  MOV MA          INR L
015304Q  MOV MB          CALL     5376Q  MOV CA          CALL     7601Q
015314Q  LXI H   22730Q   MOV CA          MOV AM          INR L
015322Q  MOV DM          SUB C           MOV EA          MOV AD
015326Q  SBC B           MOV CE          MOV BA          CALL     7166Q
015334Q  RET             LXI H   24614Q  MOV MI       1Q  INX H
015343Q  MOV MI      0Q  CALL     5376Q  LXI H   24206Q  MOV MA
015354Q  INX H           MOV MB          CMA             RRC
015360Q  JNC     15477Q  DCR L           MOV CM          INR L
015366Q  MOV BM          CALL     7601Q  MOV CA          LXI H   22726Q
015376Q  MOV MC          INR L           MOV MB          LXI H   22726Q
```

```
015404Q  MOV EM          INR L           MOV DM          LXI H   24614Q
015412Q  MOV CM          INR L           MOV BM          JMP     15463Q
015420Q  MOV AC          SUB E           MOV AB          SBC D
015424Q  JP      15434Q  MOV HB          MOV LC          XCHG
015432Q  MOV BH          MOV CL          LXI H      0Q   XCHG
015440Q  MOV AB          ORA C           RZ              XCHG
015444Q  MOV AB          RAR             MOV BA          MOV AC
015450Q  RAR             MOV CA          JNC     15456Q  DAD D
015456Q  XCHG            DAD H           JMP     15440Q  CALL    15420Q
015466Q  LXI H   24614Q  MOV ME          INX H           MOV MD
015474Q  JMP     15345Q  INR H           MOV LI    214Q  MOV CM
015503Q  INR L           MOV BM          CALL     7166Q  RET
015511Q  CALL     5376Q  MOV CA          CALL     7601Q  LXI H   22724Q
015523Q  MOV MA          INR L           MOV MB          CALL     5376Q
015531Q  MOV CA          CALL    7601Q   MOV CA          LXI H   22722Q
015541Q  MOV MC          INR L           MOV MB          LXI H   22722Q
015547Q  MOV EM          INR L           MOV DM          INR L
015553Q  MOV CM          INR L           MOV BM          CALL     7346Q
015561Q  MOV CE          MOV BD          CALL     7166Q  RET
015567Q  CALL     5376Q  MOV CA          CALL     7601Q  MOV CA
015577Q  CALL     1073Q  RET             CALL     5376Q  LXI H   24206Q
015611Q  MOV MA          INX H           MOV MB          CMA
015615Q  RRC             JNC    16003Q   LXI H   23153Q  MOV MI    377Q
015626Q  LXI H   24206Q  MOV CM          INR L           MOV BM
015634Q  CALL     2366Q  LXI H   24621Q  MOV MA          SUB I     157Q
015645Q  JNZ     15673Q  MOV CI      0Q  MOV BI      0Q  CALL     2366Q
015657Q  SUB I     141Q  SUB I       1Q  SBC A          LXI H   23144Q
015667Q  MOV MA          JMP    15603Q   MOV AM          SUB I     164Q
015676Q  JNZ     15723Q  DCR H           MOV LI    206Q  MOV CM
015705Q  INR L           MOV BM          CALL     7601Q  LXI H   23072Q
015715Q  MOV MA          INX H           MOV MB          JMP     16000Q
015723Q  MOV AM          SUB I     155Q  JNZ     15747Q  MOV CI      0Q
015733Q  MOV BI      0Q  CALL     2366Q  LXI H   23103Q  MOV MA
015744Q  JMP     16000Q  MOV AM          SUB I     160Q  JNZ     15765Q
015755Q  LXI H   22744Q  MOV MI     377Q JMP     16000Q  MOV AM
015766Q  SUB I     144Q  JNZ    16000Q   LXI H   22744Q  MOV MI      0Q
016000Q  JMP     15603Q  RET             CALL     5376Q  MOV CA
016010Q  CALL     7601Q  LXI H   22720Q  MOV MA          INR L
016020Q  MOV MB          CALL     5376Q  MOV CA          CALL     7601Q
016030Q  MOV CA          LXI H   22716Q  MOV MC          INR L
016036Q  MOV MB          LXI H   22716Q  MOV EM          INR L
016044Q  MOV DM          INR L           MOV CM          INR L
016050Q  MOV BM          CALL     7346Q  CALL     7166Q  RET
016060Q  LXI H   23150Q  MOV CM          INR L           MOV BM
016066Q  MOV LI    110Q  MOV MC          INX H           MOV MB
016073Q  LXI H   24622Q  MOV MI      0Q  INX H           MOV MI      0Q
016103Q  LXI H   23110Q  MOV CM          INR L           MOV BM
016111Q  MOV CB          XRA A           SUB C           JNC     16145Q
016117Q  LHLD    24622Q  INX H           SHLD    24622Q  LHLD    23110Q
016131Q  MOV AM          INX H           MOV BM          LXI H   23110Q
016137Q  MOV MA          INX H           MOV MB          JMP     16103Q
016145Q  LXI H   24622Q  MOV CM          INR L           MOV BM
016153Q  CALL     7166Q  RET             CALL     5376Q  MOV CA
016163Q  CALL     7601Q  LXI H   22714Q  MOV MA          INR L
016173Q  MOV MB          CALL     5376Q  MOV CA          CALL     7601Q
016203Q  MOV EA          MOV DB          LXI H   22714Q  MOV CM
016211Q  INR L           MOV BM          CALL     1744Q  RET
016217Q  CALL     1444Q  LXI H   24210Q  MOV MA          INX H
016227Q  MOV MB          LXI H   23155Q  MOV MI     377Q  LXI H   24210Q
016240Q  MOV CM          INR L           MOV BM          MOV EI     63Q
016245Q  CALL     2640Q  MOV CI      0Q  MOV BI      0Q  MOV EI     61Q
016256Q  CALL     2640Q  MOV CI      0Q  MOV BI      0Q  MOV EI    141Q
016267Q  CALL     2640Q  RET             LXI H   24624Q  MOV MC
016277Q  INX H           MOV MB          LHLD    23074Q  MOV MI      0Q
016306Q  LXI H   24627Q  MOV MI       1Q MOV AI      13Q  LXI H   24627Q
016320Q  SUB M           JC     16344Q   MOV CM          MOV BI      0Q
016327Q  LHLD    23074Q  DAD B           MOV MI      40Q  LXI H   24627Q
016340Q  INR M           JNZ    16313Q   MOV MI      0Q  MOV LI     231Q
016350Q  MOV MI      0Q  LXI H   23153Q  MOV MI     377Q  LXI H   24627Q
016362Q  MOV AM          SUB I      13Q  SBC A           MOV CA
016367Q  PUSH B          MOV LI    224Q  MOV CM          INR L
016374Q  MOV BM          CALL     2366Q  LXI H   24630Q  MOV MA
016404Q  SUB I     201Q  POP D          ADD I     377Q  SBC A
016412Q  ANA E           RRC             JNC     16513Q  MOV AM
016420Q  SUB I      72Q  JNZ    16443Q   INR L           MOV CM
016427Q  LHLD    23074Q  MOV MC          LXI H   24627Q  MOV MI      0Q

016440Q  JMP     16357Q  MOV AM          SUB I      56Q  JNZ     16457Q
016451Q  DCR L           MOV MI      10Q JMP     16510Q  DCR L
016460Q  INR M           MOV CM          MOV BI      0Q  LHLD    23074Q
016467Q  DAD B           PUSH H          LXI H   24630Q  MOV CM
016475Q  CALL     757Q   POP H           MOV MA          LXI H   24630Q
016505Q  MOV CM          INR L           MOV MC          JMP     16357Q
016513Q  CALL     666Q   RET             LXI H   24632Q  MOV MC
016523Q  INX H           MOV MB          LXI B     14Q   LHLD    23074Q
016533Q  DAD B           XCHG            MOV CI     7Q   PUSH D
016540Q  LXI H   24632Q  MOV EM          INR L           MOV DM
016546Q  MOV AD          ORA A           RAR             MOV DA
016552Q  MOV AE          RAR             MOV EA          DCR C
016556Q  JNZ     16546Q  POP H           MOV ME          LXI B     40Q
016566Q  LHLD    23074Q  DAD B           XCHG            LXI H   24632Q
016576Q  MOV AM          INR L           MOV BM          ANA I     177Q
016603Q  MOV CA          MOV AB          ANA I      0Q   MOV BA
```

```
016610Q  MOV AC
016616Q  LHLD    23074Q
016625Q  ADD A
016635Q  LHLD    23074Q
016643Q  ORA C
016652Q  RET
016662Q  CALL     5376Q
016672Q  CALL    16517Q
016704Q  INX H
016712Q  INR L
016716Q  MOV CI     17Q
016730Q  CALL     5376Q
016740Q  MOV MB
016746Q  CALL    16273Q
016760Q  CALL     5376Q
016770Q  MOV MB
016776Q  MOV BM
017003Q  CALL      462Q
017016Q  LXI H   24210Q
017024Q  RET
017034Q  INX H
017042Q  INR L
017046Q  MOV CI     20Q
017060Q  CALL     5376Q
017070Q  MOV MB
017076Q  INR L
017102Q  MOV CI     23Q
017113Q  LXI H   24637Q
017122Q  LXI H   23101Q
017133Q  DAD B
017141Q  MOV AC
017147Q  MOV AM
017157Q  MOV CM
017163Q  MOV DB
017174Q  MOV MA
017205Q  INX H
017213Q  MOV CM
017217Q  STAX D
017223Q  RET
017234Q  MOV MI      0Q
017245Q  CMA
017253Q  LHLD    24440Q
017261Q  MOV BM
017265Q  INX H
017275Q  CALL     2366Q
017306Q  JZ      17317Q
017322Q  MOV MI     377Q
017332Q  MOV AB
017341Q  LXI H   24635Q
017352Q  INR L
017362Q  SUB M
017374Q  INR M
017406Q  MOV MA
017412Q  MOV LI     237Q
017422Q  CALL     5376Q
017432Q  MOV MB
017442Q  MOV MA
017446Q  CALL     4365Q
017456Q  LXI H   24206Q
017464Q  CALL     4664Q
017476Q  CALL     5376Q
017510Q  MOV MA
017516Q  LXI H   24210Q
017524Q  XRA A
017532Q  MOV CA
017541Q  LHLD    24640Q
017553Q  MOV CM
017557Q  MOV DB
017567Q  JZ      17604Q
017601Q  INX H
017607Q  MOV AM
017617Q  MOV CA
017623Q  MOV EA
017633Q  INX H
017637Q  INR L
017644Q  MOV AB
017651Q  INX H
017657Q  MOV LI     101Q
017670Q  SUB M
017677Q  LHLD    23076Q
017705Q  MOV CI      0Q
017717Q  INR M
017727Q  LXI H   22747Q
017737Q  MOV MI     200Q
017750Q  LXI H   24652Q
017756Q  MOV LI     255Q
017767Q  SUB M
017776Q  INR L
020002Q  INX H
020010Q  INR L
020014Q  ORA C
020023Q  RET
020032Q  MOV AI      50Q
020041Q  INR C

STAX D
DAD B
DCR E
DAD D
MOV CA
CALL     5376Q
MOV CA
CALL     5376Q
MOV MB
MOV BM
CALL      462Q
LXI H   24210Q
RET
MOV CI      0Q
LXI H   24210Q
LXI H   23074Q
MOV EC
SUB I      377Q
MOV MA
RET
MOV BM
CALL      462Q
LXI H   24210Q
RET
MOV BM
CALL      462Q
LXI H   24210Q
RET
MOV BM
CALL      446Q
MOV AM
MOV CM
XCHG
STAX D
SUB I      200Q
INR L
MOV CI      25Q
LXI H   23101Q
INX H
INR L
INX D
LXI H   24637Q
CALL     5376Q
RRC
DAD B
MOV CA
SHLD    23156Q
LXI H   24635Q
CALL    17113Q
CALL     5376Q
ANA I      377Q
MOV MA
MOV MI      0Q
JC      17400Q
JNZ     17355Q
INX H
MOV AM
LXI H   24210Q
RET
INX H
CMA
MOV CM
CALL     1444Q
MOV CA
INX H
MOV MA
LXI H   24640Q
MOV AI      0Q
DCX H
INR L
MOV CI      24Q
CALL     5376Q
MOV MB
LXI D       4Q
MOV AM
CALL     2167Q
MOV MB
MOV BM
ANA I      377Q
MOV MA
MOV MI       1Q
JC      17524Q
DAD B
MOV BI      0Q
JNZ     17663Q
MOV MI      0Q
CALL    14025Q
MOV MC
MOV MI      0Q
JC      20032Q
DAD B
MOV BM
MOV CA
JNZ     20024Q
MOV LI     255Q
RET
MOV BI      0Q

RET
MOV AM
JNZ     16625Q
MOV CA
MOV BI      0Q
MOV CA
CALL     7601Q
LXI H   24210Q
LXI H   23074Q
MOV EC
SUB I      377Q
MOV MA
CALL     5376Q
MOV BI      0Q
MOV MA
MOV CM
MOV DB
JNZ     17024Q
INX H
LXI H   24210Q
LXI H   23074Q
MOV EC
SUB I      377Q
MOV MA
LXI H   23074Q
MOV EC
RET
SUB I       0Q
MOV BI      0Q
LXI H   24635Q
LXI H   23101Q
JNZ     17223Q
MOV BM
CALL      462Q
MOV MI      0Q
XCHG
MOV BM
MOV AB
MOV MI      0Q
MOV CA
RC
MOV AM
MOV LC
MOV CI      0Q
MOV MA
JMP     17271Q
ANA I      360Q
MOV BA
SUB I      201Q
MOV AI      177Q
CALL    17113Q
CALL     5376Q
MOV MB
SUB I       0Q
MOV MA
CALL     5376Q
MOV MB
RRC
INR L
MOV CA
CALL     7601Q
MOV MB
INX H
SUB M
SBC M
SHLD    24640Q
MOV BM
CALL      462Q
LXI H   24210Q
RET
LHLD    24440Q
INX H
DCR L
ANA I      360Q
INR L
MOV LI     155Q
MOV AI      177Q
MOV CM
MOV AM
CALL     2640Q
JMP     17524Q
INR H
RET
INX H
MOV MA
MOV AI      47Q
MOV CM
DAD B
LXI H   24652Q
MOV AB
MOV LI     255Q
INR M
LXI H   24162Q
LXI H   23164Q

LXI B       14Q
MOV EI       7Q
LXI D       40Q
MOV AM
CALL     7166Q
CALL    16273Q
MOV CA
MOV MA
MOV CM
MOV DB
JNZ     16741Q
INX H
MOV CA
CALL    16517Q
INX H
INR L
MOV CI      26Q
CALL     5376Q
MOV MB
MOV MA
MOV CM
MOV DB
JNZ     17071Q
INX H
MOV CM
MOV DB
JMP     17224Q
RNZ
LHLD    23076Q
MOV CM
INR M
MOV LI      74Q
MOV EC
LXI H   24637Q
LHLD    24440Q
LXI H   23156Q
MOV AC
STAX D
LXI H   23101Q
CALL     4365Q
LXI B       4Q
INX H
MOV HB
MOV BI      0Q
SUB I      201Q
LXI H   23154Q
MOV CA
CALL     2524Q
JZ      17400Q
LXI H   24636Q
LXI H   24636Q
LXI H   24210Q
INR H
JZ      17433Q
INX H
LXI H   24206Q
MOV CA
JNC     17476Q
MOV BM
CALL     5027Q
LXI H   24640Q
CALL     5376Q
MOV MB
INR L
JNC     17726Q
LXI H   23074Q
MOV EC
SUB I       0Q
MOV MA
LHLD    23076Q
DAD D
MOV DM
MOV MA
MOV AM
MOV CA
MOV MC
MOV MI      0Q
LXI H   23101Q
MOV BI      0Q
MOV EA
LXI H   23101Q
RET
MOV LI     101Q
JMP     20035Q
MOV MB
LXI H   24655Q
MOV BI      0Q
MOV AM
SUB M
SBC M
MOV AM
JNZ     17762Q
MOV CM
DAD B
```

```
020050Q  DAD B              XCHG               LXI H  24216Q  MOV AM
020056Q  INR L              MOV BM             ORA I     11Q  MOV CA
020063Q  MOV AB             ORA I      0Q      MOV BA         MOV AC
020070Q  STAX D             INX D              MOV AB         STAX D
020074Q  LXI H  24162Q      MOV CM             MOV BI     0Q  LXI H  23164Q
020105Q  DAD B              DAD B              MOV AM         INX H
020111Q  MOV BM             RRC                CMA            RRC
020115Q  JNC    20127Q      LXI H  24162Q      DCR M          JMP    20074Q
020127Q  LXI H  24162Q      MOV CM             MOV BI     0Q  LXI H  23164Q
020140Q  DAD B              DAD B              MOV AM         INX H
020144Q  MOV BM             RRC                JNC    20157Q  LXI SP  22712Q
020154Q  JMP      403Q      LXI H  24162Q      MOV CM         INR L
020164Q  MOV MC             DCR L              MOV CM         MOV BI     0Q
020171Q  LXI H  23164Q      DAD B              DAD B          MOV AM
020177Q  INX H              MOV BM             ANA I     4Q   MOV CA
020204Q  MOV AB             ANA I      0Q      LXI H  24167Q  MOV MC
020213Q  CALL     631Q      RRC                CC      5513Q  LXI H  24203Q
020225Q  MOV AM             RRC                CC      5416Q  LXI H  24162Q
020235Q  MOV CM             MOV BI     0Q      LXI H  23164Q  DAD B
020244Q  DAD B              MOV AM             INX H          MOV BM
020250Q  ANA I    360Q      MOV CA             MOV AB         ANA I    377Q
020256Q  LXI H  24642Q      MOV MC             INX H          MOV MA
020264Q  DCR H              MOV LI    216Q     MOV CM         INR L
020271Q  MOV BM             MOV LI    210Q     MOV MC         INX H
020276Q  MOV MB             LXI B       2Q     LHLD   24642Q  DAD B
020306Q  MOV AM             LXI H  24650Q      MOV MA         LXI B      3Q
020316Q  LHLD   24642Q      DAD B              MOV AM         LXI H  24651Q
020326Q  MOV MA             SUB I     201Q     SUB I     1Q   SBC A
020334Q  LXI D        4Q    LHLD   24642Q      DAD D          MOV CA
020344Q  MOV AM             SUB I     201Q     ADD I    377Q  SBC A
020352Q  ORA C              LXI H  24650Q      MOV EA         MOV AM
020360Q  SUB I    141Q      SBC A              ORA E          MOV CA
020365Q  MOV AI   172Q      SUB M              SBC A          ORA C
020372Q  RRC                JNC    20407Q      MOV CI     0Q  DCR C
020401Q  CALL   13363Q      JMP    21173Q      LHLD   24642Q  INX H
020413Q  INX H              MOV AM             INX H          MOV BM
020417Q  MOV CA             CALL   17750Q      MOV LA         MOV HI     0Q
020426Q  JMP    21037Q      CALL   10471Q      JMP    21173Q  CALL   13007Q
020442Q  JMP    21173Q      CALL   12262Q      JMP    21173Q  CALL   13242Q
020456Q  JMP    21173Q      CALL   13130Q      JMP    21173Q  CALL   12743Q
020472Q  JMP    21173Q      CALL   11420Q      JMP    21173Q  CALL   11051Q
020506Q  JMP    21173Q      CALL   14412Q      JMP    21173Q  CALL   15201Q
020522Q  JMP    21173Q      CALL   10623Q      JMP    21173Q  CALL   15144Q
020536Q  JMP    21173Q      CALL   15270Q      JMP    21173Q  CALL   15335Q
020552Q  JMP    21173Q      CALL   16004Q      JMP    21173Q  CALL   17434Q
020566Q  JMP    21173Q      CALL   17110Q      JMP    21173Q  CALL   16653Q
020602Q  JMP    21173Q      CALL   17025Q      JMP    21173Q  CALL   16613Q
020616Q  JMP    21173Q      CALL   16742Q      JMP    21173Q  CALL   17072Q
020632Q  JMP    21173Q      CALL   17727Q      JMP    21173Q  CALL   15511Q
020646Q  JMP    21173Q      CALL   14560Q      JMP    21173Q  CALL   13734Q
020662Q  JMP    21173Q      CALL   14712Q      JMP    21173Q  CALL   14730Q
020676Q  JMP    21173Q      CALL   15603Q      JMP    21173Q  CALL   13702Q
020712Q  JMP    21173Q      CALL   16060Q      JMP    21173Q  CALL    5465Q
020726Q  JMP    21173Q      CALL    5503Q      JMP    21173Q  CALL   14011Q
020742Q  JMP    21173Q      CALL   16217Q      JMP    21173Q  MOV CI     0Q
020755Q  CALL   13363Q      JMP    21173Q      CALL   14025Q  JMP    21173Q
020771Q  CALL   15567Q      JMP    21173Q      CALL   16157Q  JMP    21173Q
021005Q  LXI H  23104Q      MOV MI    377Q     MOV CI     0Q  MOV EI     0Q
021016Q  MOV DI       0Q    CALL      446Q     JMP    21173Q  MOV CI     0Q
021030Q  DCR C              CALL   13363Q      JMP    21173Q  DAD H
021040Q  LXI B  21051Q      DAD B              MOV EM         INX H
021046Q  MOV DM             XCHG               PCHL
021051Q   031Q 041Q 037Q  041Q 045Q  041Q 053Q 041Q  061Q 041Q  067Q 041Q  075Q 041Q
021067Q   103Q 041Q 111Q  041Q 117Q  041Q 125Q 041Q  133Q 041Q  141Q 041Q  147Q 041Q
021105Q   155Q 041Q 163Q  041Q 171Q  041Q 177Q 041Q  205Q 041Q  213Q 041Q  221Q 041Q
021123Q   227Q 041Q 235Q  041Q 243Q  041Q 251Q 041Q  257Q 041Q  265Q 041Q  273Q 041Q
021141Q   301Q 041Q 307Q  041Q 315Q  041Q 323Q 041Q  331Q 041Q  337Q 041Q  345Q 041Q
021157Q   353Q 041Q 363Q  041Q 371Q  041Q 377Q 041Q  005Q 042Q  026Q 042Q
021173Q  LXI H  24210Q      MOV AM             INR L          MOV BM
021201Q  ANA I    360Q      MOV CA             MOV AB         ANA I    377Q
021207Q  DCR L              MOV MC             INX H          MOV MA
021213Q  MOV LI   163Q      MOV CM             DCR L          MOV MC
021220Q  LXI H  24162Q      MOV CM             MOV BI     0Q  LXI H  23164Q
021231Q  DAD B              DAD B              MOV AM         INX H
021235Q  MOV BM             CMA                RRC            JNC    21355Q
021243Q  LXI H  24162Q      MOV CM             MOV BI     0Q  LXI H  23164Q
021254Q  DAD B              DAD B              MOV AM         INX H
021260Q  MOV BM             LXI H  24206Q      MOV MA         INX H
021266Q  MOV MB             INR L              SUB M          INR L
021272Q  MOV CA             MOV AB             SBC M          ORA C
021276Q  JZ     21311Q      MOV LI    206Q     MOV CM         INR L
021305Q  MOV BM             CALL     2052Q     LXI H  24162Q  MOV CM
021315Q  MOV BI       0Q    LXI H  23164Q      DAD B          DAD B
021324Q  XCHG               LXI H  24216Q      MOV AM         INR L
021332Q  MOV BM             ORA I     11Q      MOV CA         MOV AB
021337Q  ORA I        0Q    MOV BA             MOV AC         STAX D
021344Q  INX D              MOV AB             STAX D         MOV LI    162Q
021351Q  INR M              JMP    21220Q      LXI H  24172Q  MOV AM
021361Q  ANA I      17Q     SUB I        0Q    JNZ    21376Q  CALL   6140Q
021373Q  JMP    21431Q      LHLD   24172Q      MOV AM         SUB I     201Q
021404Q  SUB I      1Q      SBC A              LHLD   24170Q  MOV CA
021413Q  MOV AM             INX H              MOV DM         MOV ED
021417Q  PUSH B             XRA A              SUB E          POP D
021423Q  SBC A              ANA E              RRC            CC     6140Q
```

```
021431Q   LXI H   24163Q   MOV CM            DCR C             DCR L
021437Q   MOV MC            MOV LI    166Q   MOV MI    377Q    LXI B      2Q
021447Q   LHLD    24210Q   DAD B             MOV AM            SUB I    201Q
021456Q   JNZ     21475Q   LXI H   24210Q   MOV CM            INR L
021466Q   MOV BM           CALL     2052Q   JMP     21664Q    XRA A
021476Q   LXI H   24167Q   SUB M             JNC     21615Q    MOV LI   210Q
021507Q   MOV CM           INR L             MOV BM            INR H
021513Q   MOV LI    244Q   MOV MC            INX H             MOV MB
021520Q   LHLD    24644Q   MOV AM            INX H             MOV BM
021526Q   MOV CB           XRA A             SUB C             JNC     21551Q
021534Q   DCX H            MOV AM            INX H             MOV BM
021540Q   LXI H   24644Q   MOV MA            INX H             MOV MB
021546Q   JMP     21520Q   LXI H   24170Q   MOV CM            INR L
021556Q   MOV BM           LHLD    24644Q   MOV MC            INX H
021564Q   MOV MB           LXI H   24210Q   MOV CM            INR L
021572Q   MOV BM           MOV LI    170Q   MOV MC            INX H
021577Q   MOV MB           LXI B       2Q   LHLD    24170Q    DAD B
021607Q   SHLD    24172Q   JMP     21664Q   LXI H   23153Q    MOV MI   377Q
021622Q   LXI H   24210Q   MOV CM            INR L             MOV BM
021630Q   CALL     2366Q   LXI H   24647Q   MOV MA            SUB I    201Q
021641Q   JZ      21653Q   MOV CM            CALL     6527Q    JMP     21622Q
021653Q   DCR H            MOV LI    210Q   MOV CM            INR L
021660Q   MOV BM           CALL     2052Q   RET               LXI B    420Q
021670Q   CALL     4365Q   CMA               RRC               JNC     21706Q
021700Q   LXI B     420Q   CALL     4664Q   CALL     1444Q    LXI H   24210Q
021714Q   MOV MA           INX H             MOV MB            MOV CA
021720Q   CALL     5027Q   LXI H   23155Q   MOV MI    377Q    MOV LI   113Q
021732Q   MOV MI      2Q   LHLD    23076Q   MOV AM            LXI H   23113Q
021743Q   SUB M            JC      21776Q   INR M             MOV CM
021751Q   DCR C            MOV BI      0Q   LHLD    23076Q    DAD B
021760Q   MOV AM           MOV EA           LXI H   24210Q    MOV CM
021766Q   INR L            MOV BM           CALL     2640Q    JMP     21734Q
021776Q   RET              MOV CI      31Q  MOV EI       0Q   MOV DI       0Q
022005Q   CALL     462Q    LXI H   23105Q   MOV MA            INR L
022015Q   MOV MA           INR L             MOV MA            RET
022021Q   LXI H   24220Q   MOV AM            RRC               JNC     22400Q
022031Q   MOV MI       0Q   MOV LI    203Q   MOV MI       0Q   MOV LI   170Q
022041Q   MOV MI       0Q   INX H             MOV MI       0Q   LXI H   23150Q
022051Q   MOV MI       0Q   INX H             MOV MI       0Q   LHLD    23114Q
022061Q   MOV AM           INX H             MOV BM            LXI H   23110Q
022067Q   MOV MA           INX H             MOV MB            LXI B     21Q
022075Q   LHLD    23074Q   DAD B             MOV AM            SUB I    136Q
022104Q   JNZ     22202Q   LXI B       22Q   LHLD    23074Q    DAD B
022116Q   MOV AM           MOV CA            CALL    10005Q    ADD A
022124Q   ADD A            ADD A             ADD A             LXI D      23Q
022132Q   LHLD    23074Q   DAD D             MOV CA            MOV AM
022140Q   PUSH B           MOV CA            CALL    10005Q    POP D
022146Q   ORA E            MOV BI       0Q   MOV CA            LXI H   22712Q
022155Q   MOV MC           INR L             MOV MB            LXI H   22712Q
022163Q   MOV EM           INR L             MOV DM            LXI B     400Q
022171Q   CALL    15420Q   LXI H   23110Q   MOV ME            INX H
022201Q   MOV MD           LXI B   25000Q   LXI H   23110Q    MOV EM
022211Q   INR L            MOV DM            CALL     1744Q    CALL     4322Q
022221Q   CALL     5116Q   CALL     1444Q   LXI H   24216Q    MOV MA
022233Q   INX H            MOV MB            MOV LI    162Q    MOV MI       1Q
022241Q   MOV AI     376Q   LXI H   24162Q   SUB M             JC      22313Q
022252Q   MOV CM           MOV BI       0Q   LXI H   23164Q    DAD B
022261Q   DAD B            XCHG              LXI H   24216Q    MOV AM
022267Q   INR L            MOV BM            ORA I        11Q  MOV CA
022274Q   MOV AB           ORA I        0Q   MOV BA            MOV AC
022301Q   STAX D           INX D             MOV AB            STAX D
022305Q   MOV LI     162Q   INR M             JNZ     22241Q    CALL    21777Q
022316Q   LHLD    23074Q   INX H             MOV AM            SUB I     40Q
022325Q   JZ      22425Q   CALL    21665Q    MOV LI      74Q   MOV CM
022336Q   INR L            MOV BM            MOV EC            MOV DB
022342Q   MOV CI      17Q   CALL     462Q    SUB I     377Q    JZ      22375Q
022354Q   CALL      666Q   LXI H   22747Q   MOV MI       0Q   LXI B     40Q
022367Q   LHLD    23074Q   DAD B             MOV MI       0Q   JMP     22425Q
022400Q   LXI H   23104Q   MOV AM            RRC               JNC     22425Q
022410Q   MOV MI       0Q   CALL    21777Q   CALL    21665Q    DCR H
022421Q   MOV LI     347Q   MOV MI     377Q   CALL     5722Q    MOV CI     377Q
022432Q   CALL     6214Q   LXI H   24777Q   MOV MA            SUB I     173Q
022443Q   JNZ     22454Q   CALL     6713Q   JMP     22430Q    MOV AM
022455Q   SUB I     174Q   JNZ     22472Q   MOV CI       0Q   CALL     6343Q
022467Q   JMP     22526Q   MOV AM            SUB I     100Q   JNZ     22506Q
022500Q   CALL     7023Q   JMP     22526Q   MOV AM            SUB I     175Q
022511Q   JNZ     22522Q   CALL    17745Q   JMP     22526Q    MOV CM
022523Q   CALL     6527Q   JMP     22430Q   EI                HLT
022533Q    000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
```



OSI's WORD PROCESSOR

News Release *Received: 78 Jan 3*

Ohio Scientific announces its new Word Processor OS-WP1. The OS-WP1 is a text editor which operates at both the character and line levels. It has internal GET and PUT file commands which transfer individual files from memory to disk. The OS-WP1 has a full set of printer control commands which can be used with virtually any impact or matrix computer printer or word processing printer. The formatted output mode allows the user to perform left and right justification of text without line numbers at a designated width of from 20 to 70 characters.

The OS-WP1 is ideal for writing letters, manuals, reports, and all normal everyday business forms. It can be used directly with the Lear Siegler ADM-3A or with the Hazeltine 1500 and is adaptable to virtually any other conventaional CRT terminal via documentation provided.

The complete Word Processor package, two diskettes and a manual is now available for only $79.00 for use on any disk-based Ohio Scientific computer system. Available from Ohio Scientific, Hiram, OH 44234

TANDY COMPUTERS 1978 CATALOG

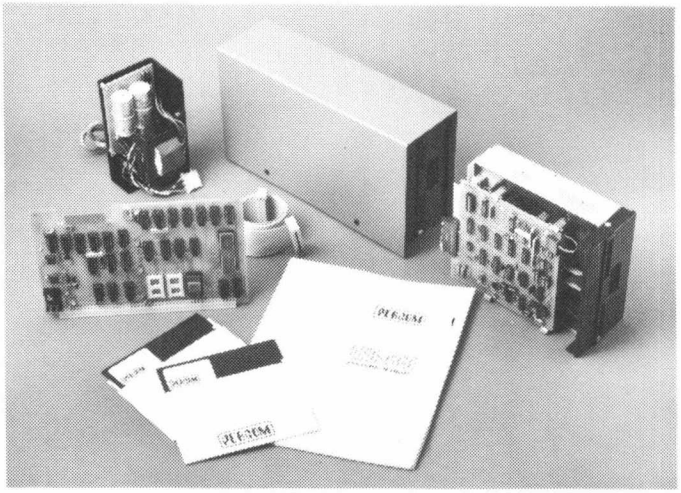
News Release *Received: 78 March 20*

A microcomputer mail-order catalog has just been issued by Tandy Computers, the newly created retail division of Tandy Corporation, parent company of Radio Shack.

The 52-page 4-color catalog details a full line of popular brand microcomputers and accessories, software packages, parts and literature currently in stock. Kits and fully assembled micromputer systems listed in the catalog range in price from several hundred dollars to more than $20,000.

Among the nationally known brands carried are Radio Shack's TRS-80, the IMSAI 8080, Vector 1 and 1+, Xitan, Equinox 100, Polymorphic System 8813, and many others offering beginners, hobbyists, educators and business users a wide selection from which to choose.

In addition, the store carries a complete selection of microprocessor mainframes, peripherals, software, printed circuit accessories and discrete parts and literature. The new store also offers programming assistance and on-premises computer service. Copies of the new Tandy Computers 1978 Catalog are available by telephoning toll-free 1-800-433-1679, or by writing to: Tandy Computers, Dept. R7, P.O. Box 2932, Fort Worth, Texas 76101



PERCOM's MINIFLOPPY FOR SS-50 BUS

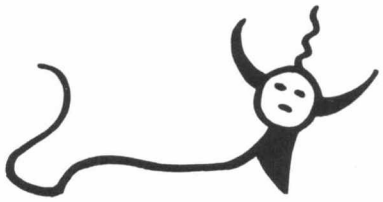
news release *Received: 77 Dec 12*

PerCom Data Company has introduced the LFD-400, a minifloppy disk system for the SS-50 bus. A complete 1-drive LFD-400 system includes a controller PC board, PROMware disk operating system, disk drive and drive power supply, interconnecting cable, two minidiskettes, an operator's manual, and an enclosure to house the drive and drive power supply.

The controller board includes special low-voltage-drop regulators, a proprietary "bit shifting" compensation circuit, an inactivity time-out circuit to increase drive motor life, and provision for 3 K byte of PROM. The controller PROM may be used completely independently of the disk and operating system.

The LFD-400 PROMware DOS, miniDOS^{tm}, allows SS-50 bus owners to use their existing software with simple patches. The miniDOS^{tm} program includes load and save routines, and is contained in a 2708 EPROM, and therefore is ready on power-up.

For more information, contact PerCom Data Company, Inc., 318 Barnes, Garland, TX 75042 (214) 276-1968.



```
022731Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 377Q 120Q 000Q
022747Q  377Q 120Q 000Q 001Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
022765Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
023055Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 170Q
023073Q  000Q 134Q 000Q 200Q 000Q 000Q 200Q 004Q 176Q 000Q 000Q 000Q 000Q 000Q
023111Q  000Q 000Q 000Q 006Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
023127Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 377Q
023145Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
023163Q  000Q 003Q 000Q 001Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
023201Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
024207Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 377Q 377Q 000Q 000Q 000Q
024225Q  000Q 000Q 377Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
024243Q  000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q 000Q
024655Q  000Q 156Q 163Q 145Q 161Q 151Q 156Q 156Q 164Q 147Q 145Q 143Q 162Q 147Q
024673Q  143Q 163Q 163Q 160Q 153Q 141Q 144Q 141Q 163Q 144Q 144Q 163Q 165Q 155Q
024711Q  154Q 144Q 166Q 162Q 146Q 167Q 146Q 157Q 146Q 143Q 146Q 162Q 156Q 155Q
024727Q  146Q 144Q 146Q 162Q 144Q 155Q 144Q 165Q 153Q 160Q 163Q 163Q 157Q 162Q
024745Q  157Q 151Q 157Q 145Q 170Q 156Q 142Q 164Q 156Q 164Q 146Q 146Q 167Q 166Q
024763Q  156Q 143Q 154Q 162Q 163Q 172Q 172Q 146Q 163Q 163Q 164Q
NO PROGRAM ERRORS
```

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 25 | May 1978 | Volume 3, Issue 5

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

Programming Language Translation Techniques *W. M. McKeeman* 4

Yet Another Body *Ken Anderson* 17
Some additions to Lichen Wang's Robot Control Language

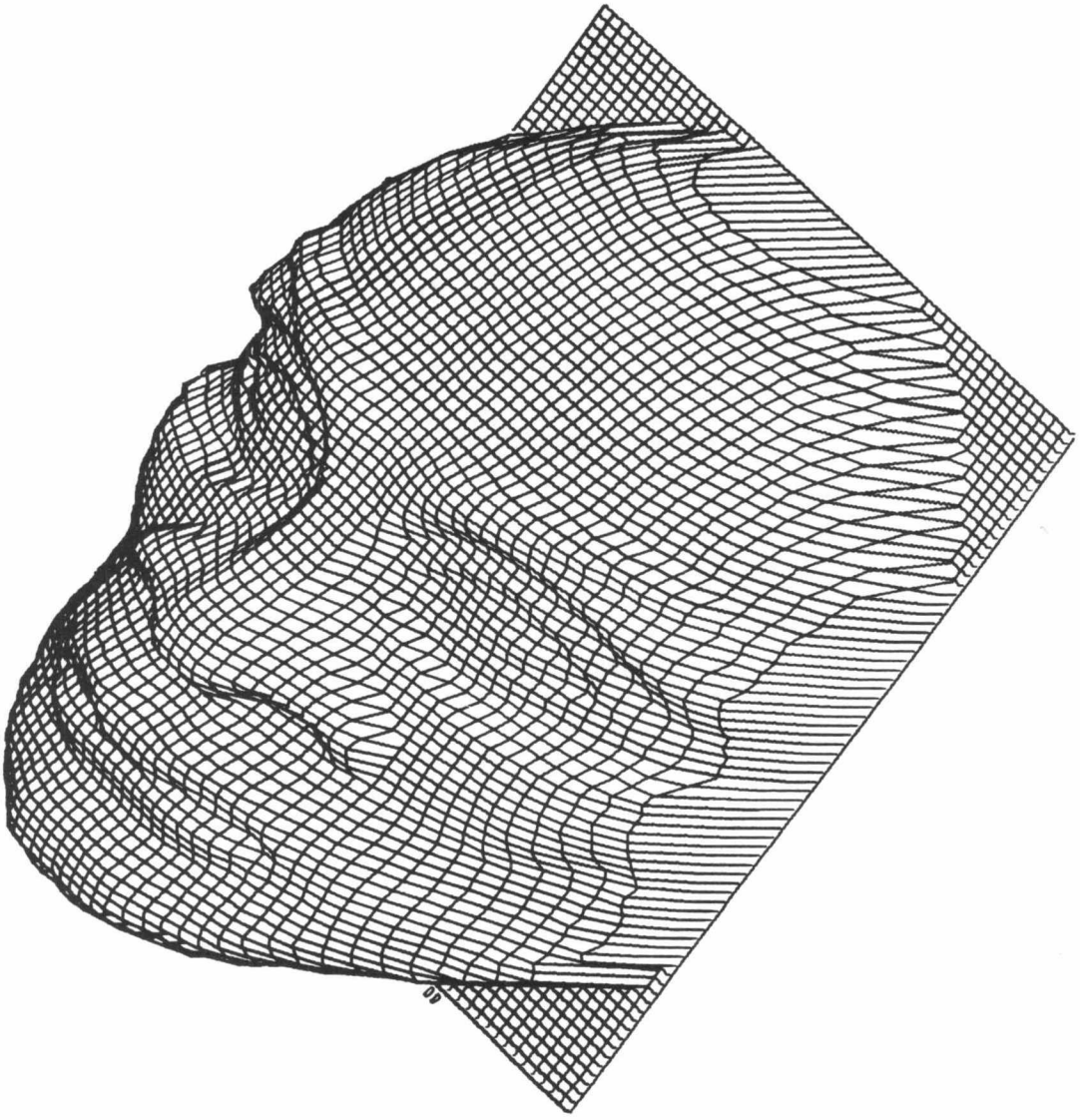
FORTH for Microcomputers *John James* 21
Description of a stack-oriented language

Resident Memory Test Systems *Terry Ritter* 28
Discussion with an example for the M6800

EDITHA: A Revision of the Fylstra KIM-1 Editor Program *H. T. Gordon* 34

HARDWARE

Proposed Standard for the S-100 Bus *George Morrow and Howard Fullmer* 40
Preliminary Specs from the IEEE Microprocessor Standards Committee



© David Dameron 1979

DISASSEMBLER FOR Z80

News Release *Received: 78 Apr 2*

alphaBIT Microsystems announces the UN-Z80 disassembler, output compatible with the TDL Z80 assembly language. Features: (1) module to be disassembled can be anywhere in the memory space in one or multiple extents. (2) program, byte or word format can be specified for each extent. (3) labels for forward and backward references are automatically resolved. (4) display or printer output will generate addresses, object and source. (5) optionally, source code may be generated and stored on disk or tape, for subsequent input to an assembler, Note: All I/O for UN-Z80 is byte oriented. User is responsible for providing the necessary interface routines for I/O, which is facilitated by JUMP vectors.

Price: $35.00 on North Star format floppy diskette or CUTER format tape cassette. Specify load address of 100H or 2A00H. Inquire about other media. $5.00 additional for any other load address. $2.00 for documentation only, applicable to purchase within 21 days. California residents add 6% or 6.5% sales tax. Contact alphaBIT Microsystems, Box 1107, 2000 Center Street, Berkeley, CA 94704.

6800 INDUSTRIAL BASIC

News Release *Received: 78 April 4*

Wintek 4K BASIC is a 6800 BASIC interpreter optimized for industrial applications. Control of interrupts, direct memory read and write, assembly language subroutine, and flexible input/output are oriented toward process control and monitoring. 4K BASIC retains many of the features of an interactive high level language, including advanced control structures including FOR. . . NEXT, IF. . .THEN. . . ELSE, and ON. . . GOTO/ON. . . GOSUB.

The interpreter may reside in RAM or in PROM. If the BASIC program is also stored in PROM, the interpreter will immediately enter the "RUN" mode, allowing unattended operation in dedicated applications. 4K BASIC is available on cassette ($95) or in PROM on a WINCE ROM Module ($299). The source listing ($95) and an OEM license ($2,000) are also available. WINTEK Corporation, 902 North Ninth Street, Lafayette, IN 47904; (317) 742-6802.

SOLOS TO NORTHSTAR DOS

News Release *Received: 78 March 15*

Microcomputer Resources, Inc. announces a software package that ties Processor Technology's SOLOS operating system to North Star DOS and BASIC. It allows the use of the CUTS tape port for archive storage of data. The VDM has start-stop and speed control which allow changes in speed while the computer is using it. All three I/O ports are supported through BASIC, and may be accessed by the '#' operator. The tape routines are accessed as I/O devices. The cursor control keys on the SOL are interfaced to North Star BASIC, allowing most edits in the line editor without the use of control keys. The user may change from the VDM to any of the SOLOS output drivers while running the DOS. Additionally, the package allows the user to list the directory of a disk while in BASIC. Documentation for the software is included on the disk. It includes several example programs written in BASIC, and data files that explain how to operate the system. This package sells for $10.00 for the diskette and program, and $2.00 to cover shipping and handling. Microcomputer Resrouces, Inc., 3000 Medical Park Drive, Suite 107, Tampa, FL 33612; (813) 977-5940.

MORE AD INFINITUM

Received: 78 March 20

In the spirit of James R. Van Zandt's "practical cure" for the software piracy problem, let me propose also that anyone needing transportation borrow Mr. Van Zandt's car and send a small rental ($2.00, say) to the car's owner, whose name and address is on the registration certificate. This rental buys three things: (1) the right to use the car, (2) a full tank of gasoline, and (3) the right to claim certain rewards from the owner.

The right to use the car would be valuable for a lot of us who want an alternative less expensive than the local car dealer but less underhanded than stealing.

Notice that I recommend that you borrow Mr. Van Zandt's car, not mine. If very many people follow his suggestions with my software, I will not be able to afford to maintain my car in operable condition, much less fill it with gasoline. And just as Mr. Van Zandt is not likely to spend very much of his money (the fruits of his labors) on buying cars for others to borrow, so I and others in the software business are unlikely to spend much labor on making software for others to copy.

Yes, I do have a better Idea: when a good program hits the market (your local computer store or the mail order ads), *buy* it. The author has already decided how much he needs to make on each copy to recover his investment. But if he *sells* more than he anticipated, he will be strongly motivated to write more (and hopefully better) software.

It is my opinion that not only is Mr. Van Zandt's suggestion economically unviable, but to suggest it is irresponsible. Such suggestions can only encourage good programmers like Bill Gates to withhold their products from hobbyists altogether. Let us rather encourage programmers to offer low-priced software by enlarging their market.

Tom Pittman

Ann Miya

PROGRAMMING LANGUAGE TRANSLATION TECHNIQUES

BY WILLIAM M. McKEEMAN
Department of Information Science
U C Santa Cruz
Santa Cruz, CA 95064

The problem of writing a programming language translator consists of setting the design objective, selecting the techniques to be used, assembling the tools, carrying out the implementation, and finally "proving" the result. Properly done, the result is an understandable program that can be easily changed, is relatively machine independent, and is free of features annoying to its eventual users. In practice this is a big, but achievable order.

In the following pages the reader will find illustrations of various principles, techniques and tools useful to the translator writer. The presentation is tutorial, suppressing some important details in the cause of brevity. Further information, including a comprehensive list of references, can be found in Gries [71] and McClure [72].

Some important attributes of a translator are listed in Table 1.1. Much of the initial planning of an implementation is concerned with choosing among them. The contents of the table are too much to assimilate all at once, thus the initial sections of this paper present two translators, illustrating two recognition algorithms and a simple code generation scheme. The paper then proceeds to elaborate upon the theme, working up to an organization subsuming most practical translators.

Table 1.1
Translator Attributes

The compiler is written in:
- (a) assembly language, or
- (b) systems programming language, or
- (c) general purpose language.

The recognition algorithm is:
- (a) table-driven, bottom up, or
- (b) recursive, top-down.

There are simplifications and optimizations on the:
- (a) token string, and/or
- (b) computation tree, and/or
- (c) language-directed code, and/or
- (d) machine code.

The main measures of efficiency are:
- (a) overall size of the translator, and
- (b) minimum memory in which the translator can be run, and
- (c) translator speed, and
- (d) ease of writing, and
- (e) ease of changing the translator, and
- (f) speed of generated code, and
- (g) density of generated code.

The mutually self-describing grammars in Table 1.2 illustrate a grammatical notation combining the familiar context-free constructs with regular expression operators. Quoting the literals [Rohl 67] allows grammatical description down to the character level if desired. The classes LETTER and CHARACTER must be extended to reflect the actual available character sets.

Table 1.2
Mutually Self-describing Grammars

```
GOAL = TOKEN_LIST;
TOKEN_LIST = | TLI | TLN | TLL | TLS;
TLI=( | TLN | TLL | TLS) IDENTIFIER;
TLN=(TLI | TLL | TLS) NUMBER;
TLL=(TLI | TLN | TLS) LITERAL;
TLS=TOKEN_LIST SEPARATOR;
IDENTIFIER=LETTER+;
LETTER='A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' | 'I'
        | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' | 'Q' | 'R'
        | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' | 'Y' | 'Z' | '_' ;
NUMBER=DIGIT+;
DIGIT='0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9';
LITERAL="" CHARACTER+"";
CHARACTER=LETTER | DIGIT | SEPARATOR | "" "" ;
SEPARATOR= '=' | ';' | '|' | '*' | '-' | '(' | ')' | ' ';

GOAL=RULE+;
RULE=IDENTIFIER '=' FORM ';';
FORM=DEFINITION ( '|' DEFINITION)*;
DEFINITION=CLASS ('-' CLASS)*;
CLASS=(PRIMARY (| '*' | '+' | NUMBER ) )*;
PRIMARY=IDENTIFIER | LITERAL | '(' FORM ')';
```

The first grammar (a lexical grammar) describes both grammars as strings of symbols where the actual detail of the internal structure of the symbols and separators is given. Because two identifiers together look like one, a TLI (token list followed by an identifier) may never be followed by another identifier. Similar comments apply to quoted items and numeric items. The second grammar (a grammar-grammar) describes both grammars as sequences of rules where the symbol ('|') separates alternatives. Within a definition we may take the difference between two phrase classes ('–') or repeat them ('*'. '+'. INTEGER). We will use this notation extensively in the following pages.

A translator accepts a *source text* convenient for programming and transforms it into a *target code* suitable for executing. To understand the translation process, one must first have a good grasp of the meaning of a source text and a target code.

For example, the source text

A = –A+5*B/(B –1);

might be translated in the following zero-address target code:

```
LIT A;
LIT A;
LOAD;
NEG;
LIT 5;
LIT B;
LOAD;
MUL;
LIT B;
LOAD;
LIT 1;
NEG;
ADD;
DIV;
ADD;
STORE;
```

Table 1.5

Table 1.3 is a grammar for assignment statements. Given that the variables must contain integers, such statements have well defined meanings. We presume that the reader is familiar enough with Algol or PL/1 to be able to follow such examples as well as simple programs.

Table 1.3

A Source Language Grammar

```
ASSIGNMENT = VARIABLE '=' EXPRESSION ;

EXPRESSION = (|'-')TERM (('+'|'-')TERM)* ;

TERM = PRIMARY (('*'|'/') PRIMARY)* ;

PRIMARY = CONSTANT | VARIABLE | '(' EXPRESSION ')' ;

VARIABLE = IDENTIFIER ;
```

Note: CONSTANT and IDENTIFIER are processed in the scanner, making their definition here irrelevant.

Table 1.4 is a grammar for the zero-address target code in the example above. Even though the meaning of the target code is probably obvious to our readers, we will use a few words to describe it.

There is an evaluation stack into which values can be pushed. Furthermore, the top values (hence the last ones pushed into the stack) are available to be used in computations. The LIT instruction pushes one value onto the stack. The value is either an address (e.g., the address of the variable A in LIT A) or a constant (e.g., the value 5 in LIT 5). The LOAD instruction assumes the top value on the stack is an address. The address is removed from the stack and the value found in the indicated cell in memory is pushed onto the stack in its place. The STORE instuction must be supplied with two items at the stack top. One is the address of a cell in memory. The other is a value to be stored into the indicated cell (the address is below the value in the stack). After the instruction has completed its action, both address and value are removed from the stack. The remainder of the instructions are arithmetic operations. NEG changes the sign of the top value on the stack and leaves it where it found it. ADD, MUL and DIV operate on the two top elements on the stack, removing them and then placing the result back on the stack.

For both the top-down and bottom-up translators, the example above will be worked in detail.

The step-by-step execution of the example zero-address target code is depicted in Figure 1.1.

Table 1.4

A Target Language Grammar

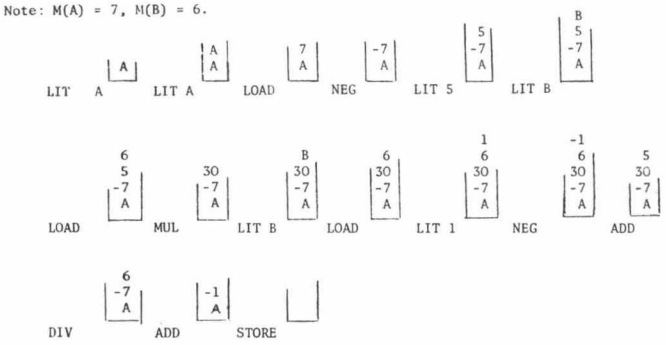
```
ASSIGNMENT = VARIABLE EXPRESSION 'STORE' ;

EXPRESSION = 'LIT' CONSTANT | VARIABLE 'LOAD'
           | EXPRESSION 'NEG'
           | EXPRESSION EXPRESSION ('ADD'|'MUL'|'DIV') ;

VARIABLE = 'LIT' IDENTIFIER ;
```

Note: Both IDENTIFIER and CONSTANT are represented by bit patterns in the machine code. The first is to be interpreted as the address of the variable and the second as the value of the corresponding constant in the source language.

Figure 1.1

Successive stack configurations during execution of the code in Table 1.5.

Note: M(A) = 7, M(B) = 6.



2. A Top-down Translator

The recursive recognition technique is one of those ideas that is so simple that nobody ever bothers to write much about it. It is, however, often the easiest method of writing good translators. The prerequisites are a syntactic definition of the source language (which then becomes the translator writer's flow chart) and a recursive language in which to write (e.g., assembler, PL/360, Algol and PL/1 but not Fortran or XPL). A partial narration of the program carrying out the translation depicted in Table 1.5 follows. Refer to Figure 2.1 and Table 2.1.

Initially (line 85), token is undefined and is then set to 'A' by the first call of scan (line 87). Then assignment is called, immediately calls variable (line 16) which checks to make sure an identifier has been found (line 77) (notice that we hid this decision inside the undefined function identifier to avoid unnecessary detail). Since 'A' is an identifier, the test is passed and the pair 'LIT' 'A' is emitted, and 'A' is discarded by a second call of scan (line 81), leaving '=' in token. Returning from variable to assignment (line 16) the '=' is noted, then discarded and replaced by the next token '–'. The right-hand side of the assignment is then processed with a call to expression. Table 2.1 gives a complete summary of the remaining actions; the reader is encouraged to complete the simulation of the recursive translator.

The success of the recursive technique is due to several circumstances. First, the mapping from a syntactic description to a recognizer (Table 1.3 to Figure 2.1) is so simple that an experienced programmer can do it as fast as he can write, although Frank DeRemer points out that the simplicity rapidly disappears if good diagnostics and error recovery are required. Second, the programmer can insert his generator code between any two statements of the recognizer. This implies that whenever any structure entity of the source language has been recognized, the programmer has a convenient opportunity to attach an interpretation to it. Third, because the local variables of recursive procedures are in fact in a run-time stack, the programmer may associate temporary information with any source language construct (see the variables t in procedures expression and term, lines 24 to 42) without laboriously building stack data structures in his translator. One must reflect on the pervasive nature of stacks in translators to appreciate the value of getting them for "free". Perhaps the most important advantage of the recursive technique, aside from its simplicity, is the fact that it can handle complex languages without catastrophic costs in size or speed.

The recursive technique is of no particular help beyond the production of a language-directed code (e.g., zero-address code for assignments). Since the great bulk of the work in writing a compiler for the average modern computer comes in turning the language-directed code into good machine language, the helpful properties detailed above can look rather irrelevant to a hard-pressed implementor.

In addition to the need for a recursive language in which to write, the recursive technique has one serious disadvantage. The generator and parser are thoroughly mixed together, preventing the programmer from treating them separately for purposes of documentation, maintenance, memory management, testing, and so forth. In particular, it is not unusual to find target code dependencies in the source language recognizer, which prevent the recognizer from being machine independent.

Figure 2.1

A Simple Recursive Translator for Assignments into Zero-address Code

```
1  translate:
2      procedure;
3          /* scan and emit are left undefined in this
4             example to avoid irrelevant detail */
5      scan:
6          procedure returns (character varying);
7              /* produces the next token each time called */
8          end scan;
9          declare token character varying;
10     emit:
11         procedure (op character varying);
12             /* assembles zero-address code */
13         end emit;
14 assignment:
15     procedure;
16         call variable;
17         if token = '=' then token = scan;
18         else call error;
19         call expression;
20         call emit ('STORE');
21     end assignment;
22 expression:
23     procedure;
24         declare t character (1);
25         /* check for unary minus */
26         if token = '-' then
27         do;
28             token = scan;
29             call term;
30             call emit ('NEG');
31         end;
32         else call term;
33         do while token = '-' |token = '+';
34             t = token; token = scan;
35             call term;
36             if t = '-' then call emit ('NEG');
37             call emit ('ADD');
38         end;
39     end expression;
40 term:
41     procedure;
42         declare t character (1);
43         call primary;
44         do while token = '*' |token = '/';
45             t = token; token = scan;
46             call primary;
47             if t = '*' then call emit ('MUL');
48             else call emit ('DIV');
49         end;
50     end term;
51 primary:
52     procedure;
53         /* check for a constant */
54         if constant (token) then
55         do;
56             call emit ('LIT');
57             call emit (token);
58             token = scan;
59         end;
60         /* check for parenthesized expression */
61         else if token = '(' then
62         do;
63             token = scan;
64             call expression;
65             if token = ')' then token = scan;
66             else call error;
67         end;
68         /* assume that it is a variable */
69         else
70         do;
71             call variable;
72             call emit ('LOAD');
```

```
73          end;
74 end primary;
75 variable:
76      procedure;
77          if identifier (token) then
78          do;
79              call emit ('LIT');
80              call emit (token);
81              token = scan;
82          end;
83          else call error;
84      end varaible;
85      /* main body of translator */
86      /* get initial token value */
87      token = scan;
88      call assignment;
89 end translate;
```

Table 2.1
Summary of the Actions of the Top-down Translator

Active line in the top-down recognizer	Value of token	Source text remaining	Target code produced
start	undefined	A=-A+5*B/(B-1)	
87			
	A	=-A+5*B/(B-1)	
88			
16			
77			
80			
	A	=-A+5*B/(B-1)	LIT A
81			
	=	-A+5*B/(B-1)	
17			
	-	A+5*B/(B-1)	
19			
26			
28			
	A	+5*B/(B-1)	
43			
54			
61			
69			
71			
77			
80			
	A	+5*B/(B-1)	LIT A
81			
	+	5*B/(B-1)	LOAD
72			
	+	5*B/(B-1)	
44			
30			
	+	5*B/(B-1)	NEG
33			
34			
	5	*B/(B-1)	
35			
43			
54			
57			
	5	*B/(B-1)	LIT 5
58			
44			
45			
	*	B/(B-1)	
46			
54			
61			
71			
77			
80			
	B	/(B-1)	LIT B
81			
	/	(B-1)	
72			
	/	(B-1)	LOAD
47			
	/	(B-1)	MUL
44			
45			
	(	B-1)	
46			
54			
61			
63			
	B	-1)	
64			
26			
32			
43			
54			
61			
71			
77			
80			
	B	-1)	LIT B
81			
	-	1)	
72			
	-	1)	LOAD
44			
33			
34			
	1	)	
35			
43			
54			
57			
	1	)	LIT 1
58			
	)		
44			
36			
	)		NEG
37			
	)		ADD
33			
65			
47			
48			
			DIV
44			
36			
37			
			ADD
33			
20			
			STORE
89			

3. A Bottom-Up Translator

The actions of a bottom-up translator separate into the actions that produce the canonical parse and those which the synthesizer takes when acting on the canonical parse. Because the canonical parse is most easily understood in relation to a context-free grammar, Table 3.1 is substituted for Table 1.3 without changing the source language. The procedure synthesize (Figure 3.1, line 47), called with the sequence of rule numbers in the left-hand column of Table 3.2, will produce the sequential language-directed code in the right-hand column of Table 3.2.

The remainder of Figure 3.1 is a source language independent parsing algorithm driven by the mixed LR(0), LR(1) recognition tables in Table 3.3. The tables are interrogated in parse (line 100) where state_type is set to the values 0 through 3 corresponding to the tabulated entries blank, S, SR and R which in turn stand for syntax error, stack the present state and apply rule N, and apply rule N.

Table 3.1

Context-free Grammar for the Assignment Source Language

rule number	rule
1	ASSIGNMENT=VARIABLE '=' EXPRESSION '⊥' ;
2	EXPRESSION=TERM
3	| '-' TERM
4	| EXPRESSION '+' TERM
5	| EXPRESSION '-' TERM;
6	TERM=PRIMARY
7	| TERM '*' PRIMARY
8	| TERM '/' PRIMARY;
9	PRIMARY=CONSTANT
10	| VARIABLE
11	| '(' EXPRESSION ')' ;
12	VARIABLE=IDENTIFIER;

Figure 3.1
A Bottom-up Translator

```
1  translate:
2     procedure;
3        /* auxiliary procedures */
4        /* token handlers */
5     scan:
6        procedure returns (character varying);
7           /* produces next token each time it is called */
8        end scan;
9        /* global residence of the current token */
10       declare token character varying;
11    unscan:
12       procedure(s character varying);
13          /* reverses the effect of scan */
14       end unscan;
15       /* stack handlers */
16       /* global residence of last token stacked or uncovered */
17       declare top_of_token_stack character varying;
18       /* global residence of last state stacked or uncovered */
19       declare top_of_state_stack fixed;
20    push:
21       procedure(s fixed, t character varying);
22          /* push s onto the state stack and t onto the
23             token stack. s and t are resident in
24             top_of_state_stack and top_of_token_stack */
25       end push;
26    pop:
27       procedure(n fixed);
28          /* discards n items from each stack, placing
29             newly uncovered values in top_variables */
30       end pop;
31       /* LR table access */
32    table:
33       procedure(s fixed, t character varying)
34       returns(a fixed, n fixed);
35          /* two-valued procedure returning values found;
36             in Table 3.3 */
37       end table;
38    left_part:
39       procedure(p fixed) returns(t character varying);
40          /* t is the left part of rule p in Table 3.1 */
41       end left_part;
42    length:
43       procedure(p fixed) returns(k fixed);
44          /* k is the number of items on the right side
45             of rule p in Table 3.1 */
46       end length;
47    synthesize:
48       procedure(p fixed);
49       emit:
50          procedure(op character varying);
51             /* assembles one item of zero-address code */
52          end emit;
53       /* switch on rule number */
54       do case p;
55          /* no rule number zero */
56          ;
57          /* ASSIGNMENT= VARIABLE '=' EXPRESSION '⊥' */
58          do;
59             call emit('STORE');
60             return from translate;
61          end;
62          /* EXPRESSION = TERM */
63          ;
64          /* EXPRESSION = '-' TERM */
65          call emit('NEG');
66          /* EXPRESSION= EXPRESSION '+' TERM */
67          call emit('ADD');
68          /* EXPRESSION= EXPRESSION '-' TERM */
69          do;
70             call emit('NEG');
71             call emit('ADD');
72          end;
73          /* TERM = PRIMARY */
74          ;
75          /* TERM = TERM '+' PRIMARY */
76          call emit('MUL');
77          /* TERM = TERM '/' PRIMARY */
78          call emit('DIV');
79          /* PRIMARY = CONSTANT */
80          do;
81             call emit('LIT');
82             call emit(top_of_token_stack);
83          end;
84          /* PRIMARY = VARIABLE */
85          call emit('LOAD');
86          /* PRIMARY = '(' EXPRESSION ')' */
87          ;
89          do;
90             call emit('LIT') ;
91             call emit (top_of_token_stack);
92          end;
93       end;
94    end synthesize;
95 parse:
96    procedure;
97       declare (state_type, N) fixed;
98       do forever;
99          /* get the values from Table 3.3 */
```

```
100     state_type, N = table(top_of_state_stack, token);
101     do case state_type;
102        /* syntax error */
103        call error;
104        /* stack */
105        do;
106           call push(n, token);
107           token = scan;
108        end;
109        /* stack and reduce, LR(0) case */
110        do;
111           call push(0, token);
112           call synthesize(N);
113           call pop(length(N));
114           token = left_part(N);
115        end;
116        /* reduce, LR(1) case */
117        do;
118           call synthesize(N);
119           call pop(length(N));
120           call unscan(token);
121           token = left_part(n);
122        end;
123     end;
124  end;
125  end parse;
126  /* main body of translate */
127  /* initialize read-only syntax tables */
128  call initialize_tables;
129  /* initialize the stacks */
130  call push(1, ' ');
131  /* read the first token of the source text */
132  token = scan;
133  call parse;
134  end translate;
```

Table 3.2
Association of the Canonical Parse and Code Generation

rule applied	partially parsed text	emitted code
	A=-A+5*B/(B-1) |	
12	VARIABLE=IDENTIFIER	LIT A
	VARIABLE=-A+5*B/(B-1) |	
12	VARIABLE=IDENTIFIER	LIT A
	VARIABLE=-VARIABLE+5*B/(B-1) |	
10	PRIMARY=VARIABLE	LOAD
	VARIABLE=-PRIMARY+5*B/(B-1) |	
6	TERM=PRIMARY	
	VARIABLE=-TERM+5*B/(B-1) |	
3	EXPRESSION=-TERM	NEG
	VARIABLE=EXPRESSION+5*B/(B-1) |	
9	PRIMARY=CONSTANT	LIT 5
	VARIABLE=EXPRESSION+PRIMARY*B/(B-1) |	
6	TERM=PRIMARY	
	VARIABLE=EXPRESSION+TERM*B/(B-1) |	
12	VARIABLE=IDENTIFIER	LIT B
	VARIABLE=EXPRESSION+TERM*VARIABLE/(B-1) |	
10	PRIMARY=VARIABLE	LOAD
	VARIABLE=EXPRESSION+TERM*PRIMARY/(B-1) |	
7	TERM=TERM*PRIMARY	MUL
	VARIABLE=EXPRESSION+TERM/(B-1) |	
12	VARIABLE=IDENTIFIER	LIT B
	VARIABLE=EXPRESSION+TERM/(VARIABLE-1) |	
10	PRIMARY=VARIABLE	LOAD
	VARIABLE=EXPRESSION+TERM/(PRIMARY-1) |	
6	TERM=PRIMARY	
	VARIABLE=EXPRESSION+TERM/(TERM-1) |	
2	EXPRESSION=TERM	
	VARIABLE=EXPRESSION+TERM/(EXPRESSION-1) |	
9	PRIMARY=CONSTANT	LIT 1
	VARIABLE=EXPRESSION+TERM/(EXPRESSION-PRIMARY) |	
6	TERM=PRIMARY	
	VARIABLE=EXPRESSION+TERM/(EXPRESSION-TERM) |	
5	EXPRESSION=EXPRESSION-TERM	NEG ADD
	VARIABLE=EXPRESSION+TERM/(EXPRESSION) |	
11	PRIMARY=(EXPRESSION)	
	VARIABLE=EXPRESSION+TERM/PRIMARY |	
8	TERM=TERM/PRIMARY	DIV
	VARIABLE=EXPRESSION+TERM |	
4	EXPRESSION=EXPRESSION+TERM	ADD
	VARIABLE=EXPRESSION |	
1	ASSIGNMENT=VARIABLE=EXPRESSION |	STORE
	ASSIGNMENT	

Translation starts on line 126 where the syntax tables are initialized, the initial state is stacked and the initial value of token is scanned. The recognizer is called and the loop on line 98 is entered. The variables state_type and N are assigned the values 2 (i.e., entry SR) and 12 (the intersection of row IDENTIFIER and column 1 of Table 3.3). The case statement selects the stack-and-reduce case (line 109) where a dummy-state (0) and the identifier 'A' are stacked. Synthesize is called with parameter 12 (line 112) (line 54) selects the last case (line 88) causing 'LIT' 'A' to be emitted. Returning to the parser (line 113) one item is popped off each stack, leaving the initial state (1) exposed. Token is set to VARIABLE. Execution continues on line 100 where state-type and N assume new values 1 and 15, respectively.

There are programs that produce syntax tables similar to Table 3.3 from input similar to Tables 3.1 or 1.3. How it is done is an interesting subject but of no direct concern to the translator implementor who need only write the procedure synthesize. There are, however, fewer opportunities to attach interpretations to the parsing actions since synthesize is called only from SR and R decision and not S decision. And, except for the state and token stack, the programmer must build his own data structures.

Table 3.3

Syntax Tables for Mixed LR(0) and LR(1) Parsing Algorithm

	state 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
|									SR 1		R 2	R 3	R 4	R 5	
(		S 3	S 3	S 3	S 3	S 3	S 3	S 3							
)										SR 11	R 2	R 3	R 4	R 5	
=															S 2
+									S 4	S 4	R 2	R 3	R 4	R 5	
-		S 5	S 5						S 6	S 6	R 2	R 3	R 4	R 5	
*											S 7	S 7	S 7	S 7	
/											S 8	S 8	S 8	S 8	
ASSIGNMENT															
EXPRESSION		S 9	S 10												
TERM		S 11	S 11	S 13	S 12	S 14									
PRIMARY		SR 6	SR 6	SR 6	SR 6	SR 6	SR 7	SR 8							
CONSTANT		SR 9	SR 9	SR 9	SR 9	SR 9	SR 9	SR 9							
VARIABLE	S 15	SR 10	SR 10	SR 10	SR 10	SR 10	SR 10	SR 10							
IDENTIFIER	SR 12	SR 12	SR 12	SR 12	SR 12	SR 12	SR 12	SR 12							

Table 3.3 is equivalent to LALR(1) parsing tables [DeRemer 69]

4. An Elaborate Compiler

A compiler accepts source text records and produces records full of machine code ready for execution. During the process the program may assume several intermediate forms, including:

records
string of characters
string of tokens
phrase-structure tree
computation tree
language-specific sequential code
machine-specific sequential code
records

The design of a translator depends heavily on which of the above forms are to be used. Generally speaking, the more complex the source text, the higher the payoff in making the compiler more elaborate. Once the sequence of intermediate forms has been chosen, getting from one form to another represents the major implementation problem. Performing transformations on the intermediate forms to simplify the remaining tasks or improve the quality of the resulting machine code may be a subsidiary task. The overall data flow for an elaborate compiler is given in Figure 4.1. The remainder of this article is devoted to explaining the meaning of the various algorithms and data structures depicted in the figure.

5. Transformation on the Token String

The problem of translating a very complex language (e.g., the IBM PL/1) can be simplified by passing repeatedly over the token string, each time producing an equivalent token string with fewer primitive concepts. The final set of primitives is usually taken to include GOTO, IF, assignments and other things that map directly onto a conventional instruction set.

Table 5.1 displays three successive equivalence transformations on a PL/1 program. Initially there is an array assignment which implies a loop (stage 2). The translator has had to create some new identifiers which are started with '$' to avoid confusion. The control expressions of the do-loop are evaluated only once implying that the head of the loop contains only simple variables (stage 3). The loop itself can then be factored into IF and GOTO constructs (stage 4). The advantages are the already mentioned reduction in overall translator complexity and also the fragmentation of the translator to aid in making each piece small enough to permit running in a small memory. It has some utility in documenting the meaning of language constructs where the transformations are simple enough to be easily understood.

The technique has some disadvantages. It can be slow since repeated passes over the token string are required. Clever methods of speeding things up may cancel the gain in simplicity that led to its use in the first place. There is also a lack of theory governing the transformations, leaving a potentially confusing series of ad-hoc algorithms. Finally, the transformations may obscure relations that would have been useful to the generators. For instance, the fact that the example in Table 5.1 can be accomplished with a single memory to memory block transfer instruction will never be recovered from the "simplified" form at stage 4.

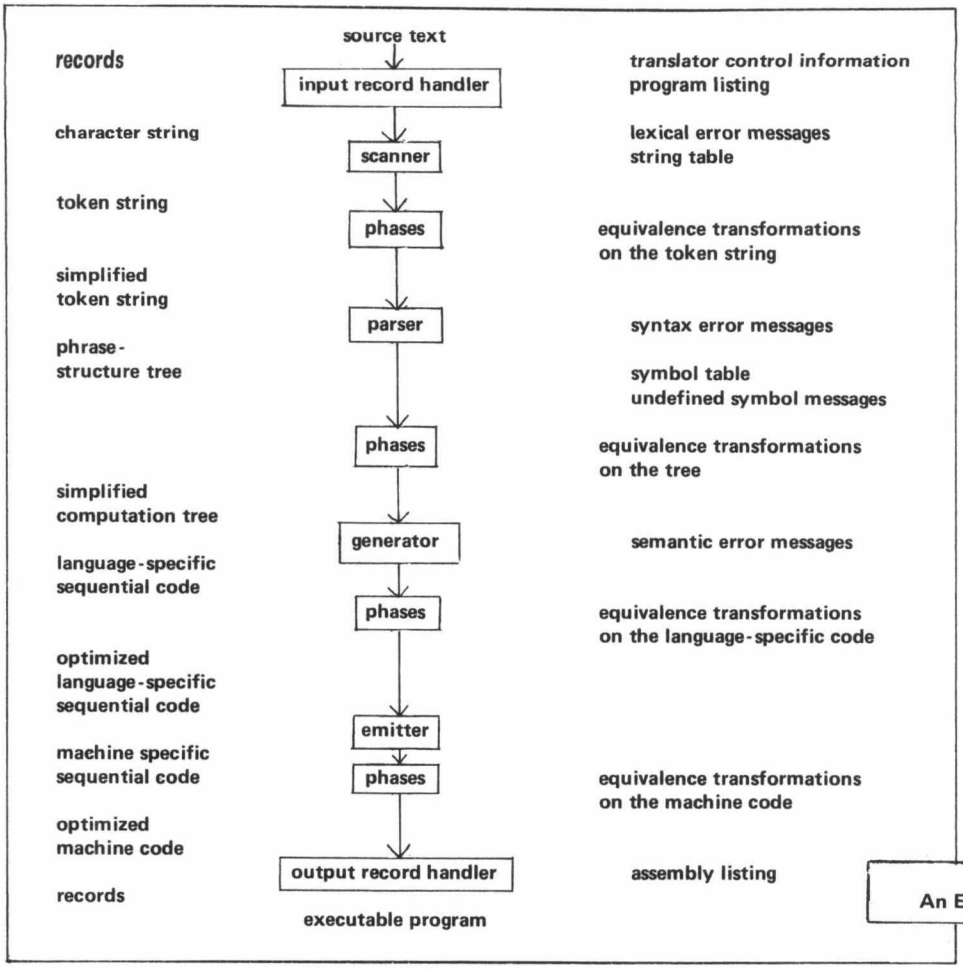


Figure 4.1
An Elaborate Compiler

```
DECLARE (A, B) (20) FIXED;
A=B;
DECLARE (A, B) (20) FIXED;
DECLARE $I1 FIXED;
DO $I1=LBOUND(A) TO HBOUND(A);
   A($I1)=B($I1);
END;
DECLARE (A,B) (20) FIXED;
DECLARE ($I1, $I2, $I3, $I4) FIXED;
$I2=LBOUND(A);
$I3=1
$I4=HBOUND(A);
DO $I1=$I2 BY $I3 TO $I4;
   A($I1)=B($I1);
END;
DECLARE (A, B) (20) FIXED;
DECLARE ($I1, $I2, $I3, $I4) FIXED;
$I2=LBOUND(A);
$I3=1;
$I4=HBOUND(A);
$I1=$I2;
$I5=IF $I1>$I2 THEN GO TO $I6;
   A($I1)=B($I1);
   $I1=$I1 + 1;
   GO TO $I5;
$I6:
```

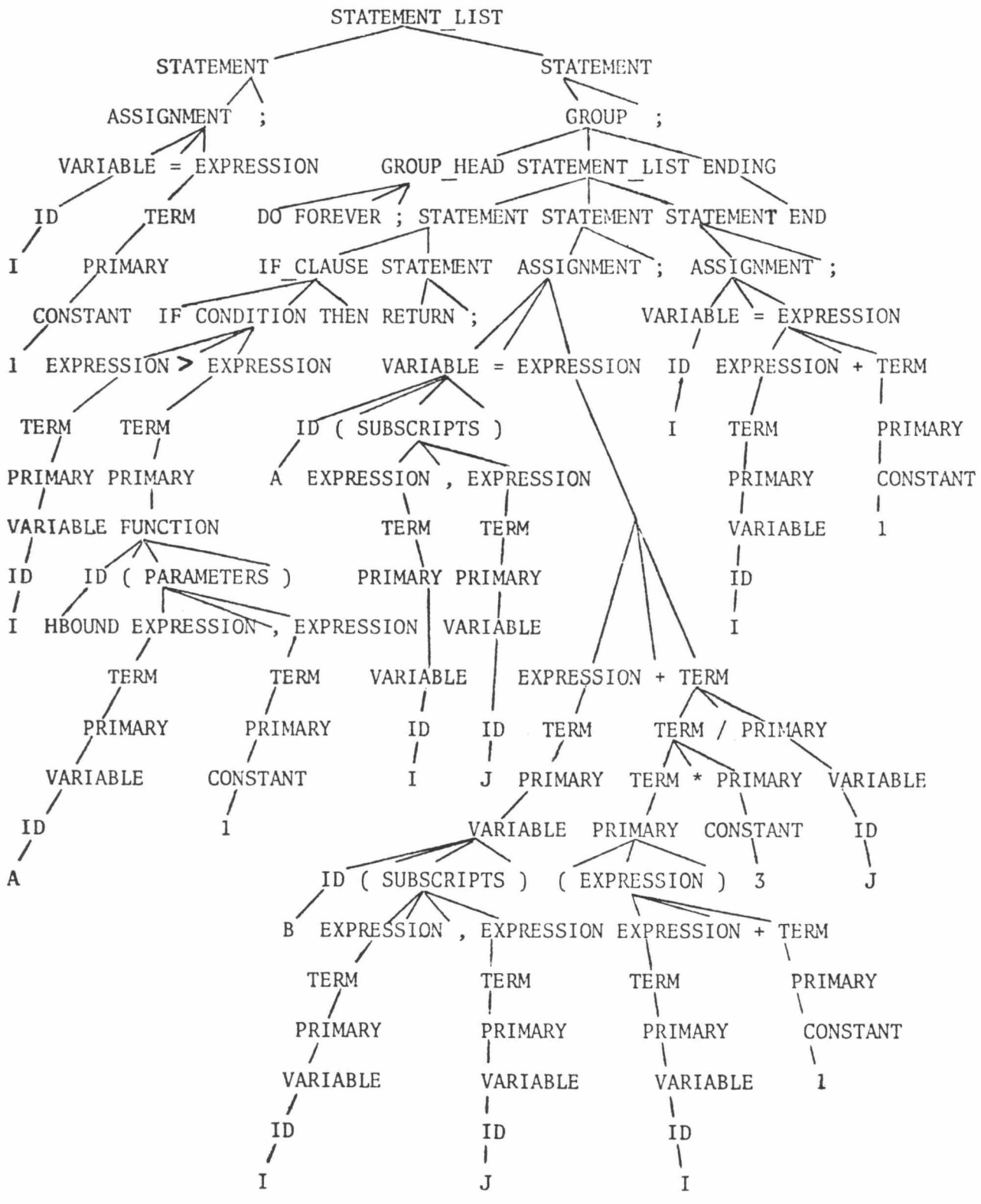
Table 5.1
Successive Transformations of a PL/I Program Fragment

```
I=1;
DO FOREVER;
   IF I > HBOUND(A, 1) THEN RETURN;
   A(I, J) = B(I, J) + (I + 1) * 3/J;
   I = I + 1;
END
```

Figure 6.1
A Source Text

Figure 6.2

Phrase-structure Tree for Text in Figure 6.1



6. Transformation on the Computation Tree

For a given program (Figure 6.1) the recognizer can automatically compute the phrase-structure tree (Figure 6.2), which can then be reduced to a computation tree by renaming the nodes with language-directed actions and deleting those that call for no operation. The computation tree (Figure 6.3) then constitutes a data structure with an interpretation that is invariant under certain transformations. Both reducing the number of primitives (simplification) and improving the code efficiency (optimization) are useful goals. In the example, the concepts of pre-evaluating constant subexpressions and linearizing integer multiplication are illustrated. The reader should be aware that there are many other useful transformations. Frank DeRemer calls the tree in Figure 6.3 the abstract syntax tree. The transformation from phase-structure tree to abstract syntax tree can be formalized to the point of allowing the compiler writer to specify it in tabular form, thus avoiding the rather messy task of manipulating the tree directly.

The loop has an induction variable (I) which is stepped by a constant amount (1). This fact, discovered by examination of the computation tree, allows multiplication by a constant (or unchanging variable) to be turned into addition. The generally slower execution time of multiplication together with common subscript algorithms which imply a fixed-point multiplication means that there is considerable to be gained. Figures 6.4 through 6.13 depict subsequent stages of the computation tree as the various phases of the transformations are carried out. Each figure should be compared with an earlier figure (usually Figure 6.3) to get a before/after view of the transformation being illustrated.

In Figure 6.4, the array A, having declared elsewhere to have an upper bound of 8, yields a constant for the function HBOUND. In Figure 6.5, the double subscripts are turned into simple single subscripts with polynomial values. The multiplication of an induction variable is linearized by creating a new induction variable ($I2) which is stepped by the multiplier (Figure 6.7). Then addition of constants are twice pushed off onto the new induction variable (Figures 6.8 and 6.9). Then the use of the variable I directly after it has been set to a constant value allows some pre-evaluation (Figure 6.10). Finally, the remaining multiplication by a constant is pushed out to the leaves of the computation tree (Figure 6.11) so that it too can be linearized (Figure 6.12). The resulting tree is displayed in Figure 6.13.

Tree transformations consume more computational resources than most other phases of translation both because the tree occupies a lot of memory and also following the link takes a lot of time. Thus the implementor must be careful in deciding which transformations are economically justified. Tree transformations are easier to carry out when the source language does not include a GO TO construct to break up sequences of code, thereby making difficult the detection of induction variables and the like.

Figure 6.3

Computation Tree from Figure 6.2

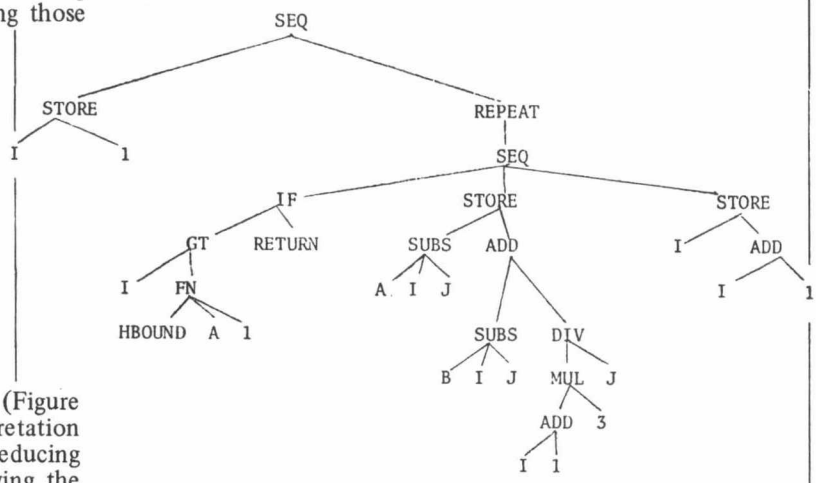


Figure 6.4

Pre-evaluating Constant Subexpressions

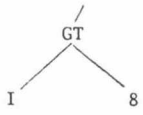


Figure 6.5

Replacing Double Subscripts with a Single Polynomial

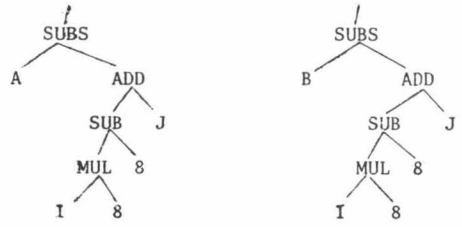


Figure 6.6

Combining Common Subexpressions

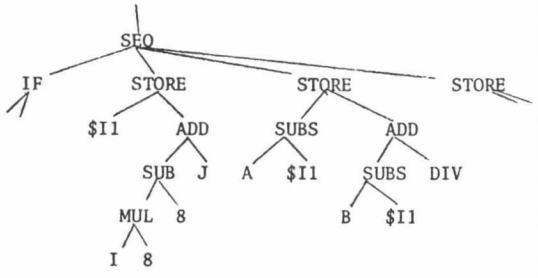


Figure 6.7

Linearizing an Integer Multiplication

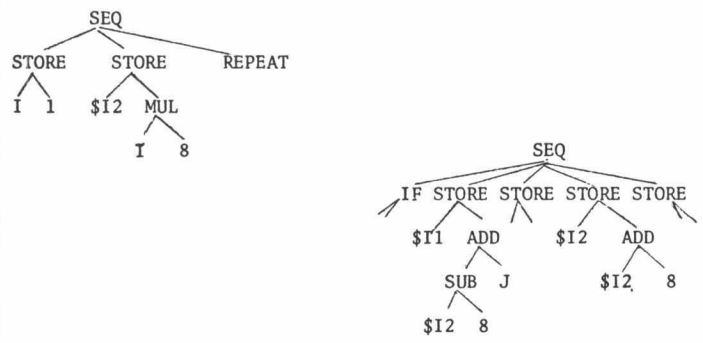
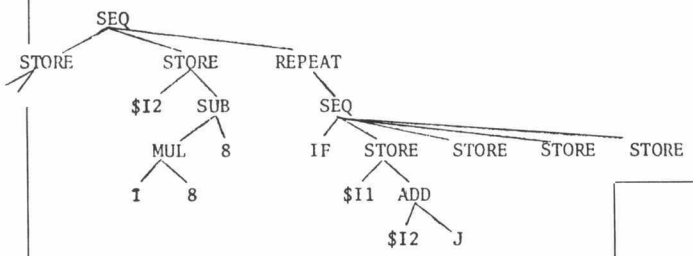


Figure 6.8

Removing Linear Terms from a Loop



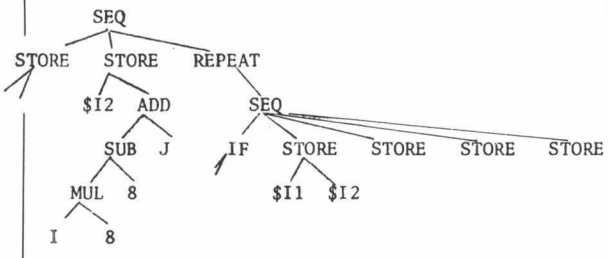


Figure 6.9

Combining Variables with a Common Value

and delete

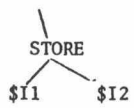


Figure 6.10

Pre-evaluating Constant Subexpressions

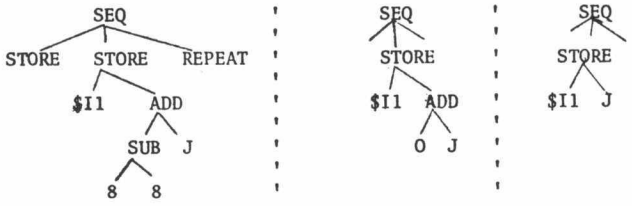
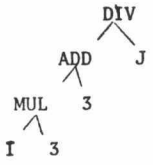


Figure 6.11

Pre-multiplying by a Constant



-42-

Figure 6.12

Linearizing an Integer Multiplication

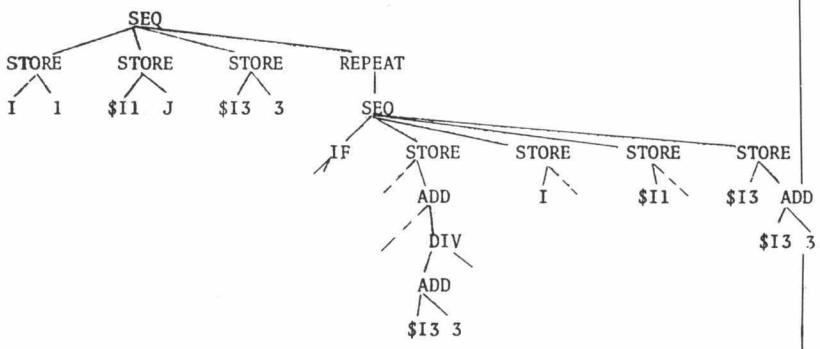
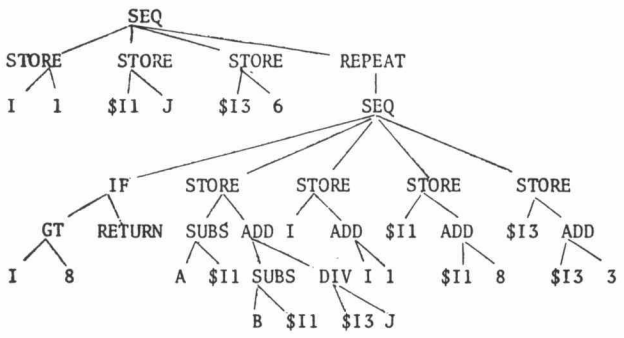


Figure 6.13

Computation Tree After Simplification and Optimization



7. Transformations on Lanuguage-Specific Sequential Code

The computation tree yields the language-specific sequential code partly by resort to zero-address code (a left-to-right sweep of the tree) and partly by insertion of transfer instructions where implied by the tree (IF, THEN, ELSE, loop, procedure calls, etc.). The transfers can either be conventional branches or more restricted transfers corresponding to CASE, CALL, RETURN, REPEAT, etc. Such a code for the tree in Figure 6.13 is given in Figure 7.1.

Most of the transformations that can be carried out on the computation tree can also be carried out on the language-specific sequential code [Allen 71, Cocke & Schwartz 70]. In addition, the branching structure itself is open to some optimizations (e.g. if the destination of a branch instruction is an unconditional branch, the first branch can be reformulated to go directly to its ultimate destination).

8. Transformations on Machine Code

The translator emits machine code by passing over the language-specific code with a simulated language-specific machine which, instead of evaluating, emits code (Table 8.1). Figure 8.1 displays several stages of the stack during the execution of such an algorithm [McKeeman et al 70].

Table 8.1
Source Text and Language-specific Code

X=X+1; If X < 3 THEN GO TO L;
LIT X; LIT X; LOAD; LIT 1; ADD; STORE;
LIT X; LOAD; LIT 3; LT; LIT $L1; BRANCH_FALSE;
LIT L; BRANCH; $L1: . . .

Most computers have peculiarities that are utilized by assembly language programmers but are not directly available to the compiler, partly because the opportunities to make use of them do not become apparent until after the machine code itself has been emitted, and partly because they are special to the target machine in question [McKeeman 65]. Table 8.2 shows a typical situation of this kind. At stage 1 we have a good machine code for the two statements. Because addition is commutative, the addresses of a pair, LOAD ADD, can be interchanged (stage 2) which permits the slow LOAD = 1 to be replaced with a fast load immediate LI 1 and also saves the cell containing the constant 1 (stage 3). In any pair, STORE X LOAD X, the LOAD can be dropped if it is not the destination of a branch (stage 4). Any pair of branches, where the first is a conditional skip and the second is unconditional can be reduced to a single conditional branch (stage 5). Finally, on a machine with an add-one-to-memory instruction, the sequence LI 1, ADD X, STORE X can be combined into one instruction (stage 6).

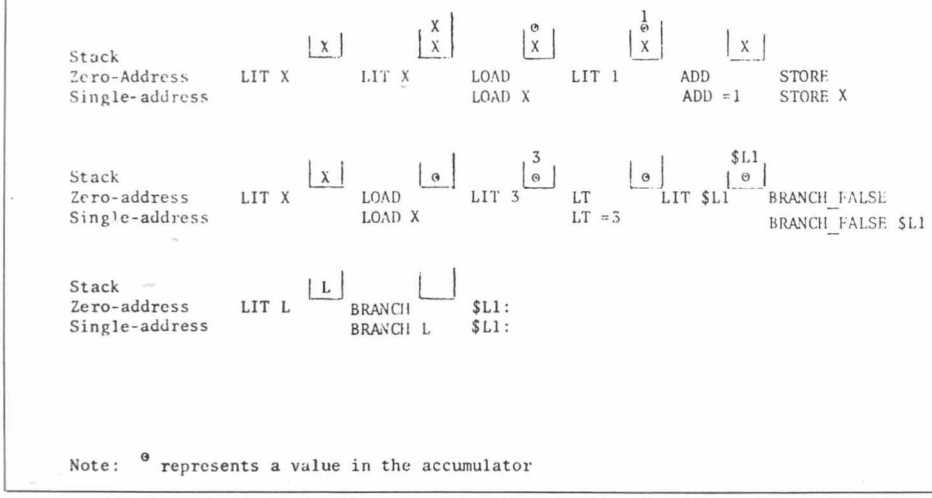
Figure 7.1

Sequential Form of Figure 6.13

```
LIT 1;  LIT 1;  STORE;  LIT $I1;  LIT J;  LOAD;  STORE;  LIT $I3;
LIT 6;  STORE;  LIT $I4;  CALL ...

$I4:  LIT I;  LOAD;  LIT 8;  GT;  LIT $I5;  BRANCH_FALSE;  RETURN;
$I5:  LIT A;  LIT $I1;  LOAD;  ADD;  LIT B;  LIT $J1;  LOAD;  ADD;
LOAD;  LIT $I3;  LOAD;  LIT J;  LOAD;  DIV;  ADD;  STORE;  LIT I;
LIT I;  LOAD;  LIT 1;  ADD;  STORE;  LIT $I1;  LIT $I1;  LOAD;
LIT 8;  ADD;  STORE;  LIT $I3;  LIT $I3;  LOAD;  LIT 3;  ADD;  STORE;
LIT $I4;  REPEAT;
```

Figure 8.1

An Emitter Turning Zero-address Code into Single-address Code



Note: ⊖ represents a value in the accumulator

Table 8.2

Transformations on Machine Code

	LOAD X ADD =1 STORE X
Stage 1	LOAD X LT =3 BRANCH_FALSE $L1 BRANCH L $L1:

	LOAD =1 ADD X STORE X
Stage 2	LOAD X LT =3 BRANCH_FALSE $L1 BRANCH L $L1:

	LI 1 ADD X STORE X
Stage 3	LOAD X LT =3 BRANCH_FALSE $L1 BRANCH L $L1:

	LI 1 ADD X STORE X
Stage 4	LT =3 BRANCH_FALSE $L1 BRANCH L $L1:

	LI 1 ADD X STORE X
Stage 5	LT =3 BRANCH_TRUE L

	INDEX X
Stage 6	LT =3 BRANCH_TRUE L

Innumerable ingenuities can be applied. Since the savings can be dramatic (as in the example), the technique is useful and popular. Nothing makes the hero-author of a compiler happier than producing really subtle correct code, beating the assembly programmer at his own game. One must, however, guard against introducing errors into the code as a result of not carefully considering the conditions under which the transformations leave the program invariant.

9. Fragmentation Planes in the Translator

There are many reasons to want to break a compiler into pieces. The ability to run the translator in a small memory is one which the computer salesman and computer purchaser feel is important. The problem of managing large programming projects has demanded a task organization which results in many small well-defined subtasks. Further, there seems to be a definite limit on how complex a task can be attempted without incurring exponentially growing delays and costs. There even seems to be a critical size beyond which repairing an error introduces on the average of more than one error. And, finally, no matter how well a system *seems* to perform, it is good engineering practice to know how well and under what range of conditions the system will perform. Such knowledge comes from understanding; understanding comes from not having to consider something complex all in one mental step.

There are some principles that guide the choice of natural boundaries between program modules. A boundary is good if there are a few references across it into other modules and those references are relatively infrequently exercised. A boundary is good if the information flow through it can be simply described. A module is good if its function can be simply described. A data flow is good if it can be totally produced before any of it is consumed (meaning no feedback is needed across the boundary). A set of modules is good if execution activity can be isolated in small subsets (working sets) over substantial spans of time. And such a set is well-designed if particular dependencies are isolated in small subsets (e.g. target language does not appear until the very last stages in Figure 4.1). It is also of value if one module can be changed without involving the processing of many other modules. We can suppress detail by considering a subset of the modules as a single module where the internal data transactions and control transfers are ignored. The boundary around such a supermodule is good if the resulting module obeys the rules above.

The techniques of engineering programs are being developed. One of them is modular programming which, when applied to translators, depends upon the organization of the translator (e.g. Table 4.1) and upon the application of the principles above. The time should soon come when translators are well-engineered as a rule, not as an exception.

REFERENCES

Allen, Frances E., "A Basis for Program Optimization," *Computer Software*, IFIP (1971), page 64.

Cocke, J. and Schwartz J.T., "Programming Languages and their Compilers (preliminary notes)", Courant Institute of Mathematical Sciences, N.Y.U., N.Y. (April 1970).

DeRemer, F.L., *Practical Translators for LR(k) Languages*, Ph.D. dissertation, M.I.T. (September 1969).

Dijkstra, E.W., *Notes on Structured Programming, Report 241*, Technische, Hogeschool Eindhoven (1969).

Gries, D., *Compiler Construction for Digital Computers*, John Wiley & Sons (1971).

Lehman, M. & Belady, L., "Programming System Dynamics. . ." *IBM Research RC 3546* (September 17, 1971).

McClure, R.M., "An Appraisal of Compiler Technology," *Proc. SJCC* (1972) page 1-9.

McKeeman, W.M., Horning, J.J. & Wortman, D.B., *A Compiler Generator*, Prentice Hall (1970).

McKeeman, W.M., "Peephole Optimization," *Comm. ACM* 8, 7 page 443 (July 1965).

Parnas, D.L., "Information Distribution Aspects of Design Methodology," *Computer Software*, IFIP (1971) page 26.

Rohl, J.S., "A Note on Backus-Naur Form," *Computer Journal*, Vol. 10, pages 336-337 (1967-68).

MIND'S EYE SOFTWARE

News Release *Received: 78 April 5*

Mind's Eye Personal Software produces and distributes software for the personal and hobbyist computer user. Initially, this consists of cassette tapes for the Commodore PET Personal Computer.

Mind's Eye will sell tapes either on a direct mail-order basis, or at a quantity discount rate for resale by other vendors. The direct mail price will be approximately 10% less than the suggested retail price for the other vendors.

A Catalog will be available soon which will list and describe the programs available from Mind's Eye. We hope, as we grow, to offer an extremely large variety of programs for your perusal. The Catalog will also contain many other things of interest, such as royalty policies, expected programming standards, and a section of advertisements by vendors of PET related hardware.

To attract a variety of programs and to assure their quality, royalties in the range of 10-20% of the RETAIL price of the software will be offered. Submitted programs will be checked and "editorial" suggestions made to the authors to maintain a high level of reliability and creativity. Authors who are willing to make suggested changes will receive higher royalties as a result.

As this is a *Personal* software company, please do not hesitate to send programs which you feel may be of limited interest. Mind's Eye has carefully considered the problem of low-volume sales of individual programs, and has designed its facilities to be capable of handling low-volume programs profitably.

Mind's Eye is especially interested in programs which 1) solve a particular problem that you have had, 2) educational items, including ones which are "trivial", 3) programs dependent on the PET operating system, 4) machine language programs, 5) useful utilities, and naturally, 6) games and amusements. Contact P.O. Box 354, Palo Alto, CA 94301; (415) 326-4039.

FLOPPY CONTROLLER USES NEW MOTOROLA CHIP

News Release *Received: 78 Mar 13*

WINTEK Corp. has incorporated the new Motorola MCM 6843 floppy disk controller IC into a low cost floppy disk controller. The 4½" x 6½" module interfaces to any full size or mini floppy disk drive. The module supports both hard and soft sectoring, IBM 3740 or user programmable read/write format, automatic CRC generation/checking, and programmable step and setting times. $199.00 unit price. WINTEK Corp., 902 N. 9th Street, Lafayette, IN 47904; (317) 742-6802.

YET ANOTHER BODY

SOME ADDITIONS TO LICHEN WANG'S ROBOT CONTROL LANGUAGE

BY KEN ANDERSON
KEA Microdesign
P.O. Box 6531 Stn. A
Toronto, Canada M5W/X4

Dear Jim: *Received: 78 Mar 24*

Enclosed is a program listing of 'Yet Another Body' for Lichen Wang's Tiny Robot Language. In this body, the VDM/Sol display is shared to map robot movement during the 'Body Mode' as well as display mind contents during the 'Mind Mode'. Please note that a graphics density of 128 x 48 cells is accomplished on the VDM/Sol display by addition of a 'GraphicAdd' piggyback board. This program is easily converted for Poly/SSM type displays by modification of the cell mask table (see listing).

Ken Anderson

For a full description of the TINMAN language, also known as WSFN, and 8080 source listing, see *DDJ* Volume 2, Issue 8, pp. 8-19 'An Interactive Programming Language for Control of Robots'. Hardware requirements are a Sol or VDM display equipped with GraphicAdd graphics board, an ASCII keyboard and an 8080 system with 8K of RAM at 0-1FFF. On initial startup of WSFN, the computer asks 'Are my graphics software-selectable?' Keying 'Y' will enable the GraphicAdd graphics for this and subsequent re-entries to WSFN. Then the 'Mind Mode' is entered. If valid mind contents exist at 1C00 in RAM (see the Mind, below), the mind is displayed—if not, then the mind is first brainwashed. This feature allows previous 'memories' to be reloaded from mass storage before re-entry to WSFN.

The Mind

This is functionally identical to Lichen Wang's description. However, this mind is located in RAM at 1C00-1FFF. After every keypress, the 'Mind Mode' is entered, and the VDM is updated with the latest mind contents. Thus, during all mind operations, the display appears as described in *DDJ*. A few more commands have been added for convenience:

ESC (1B hex)	Exits to Monitor
CTL B (02 hex)	Brainwashes the mind for a fresh start
;	is translated to + to save wear on the shift key
← (01 hex)	back-arrow backs up cursor to correct typos

NOTE—does not move over brackets.

Yet Another Body

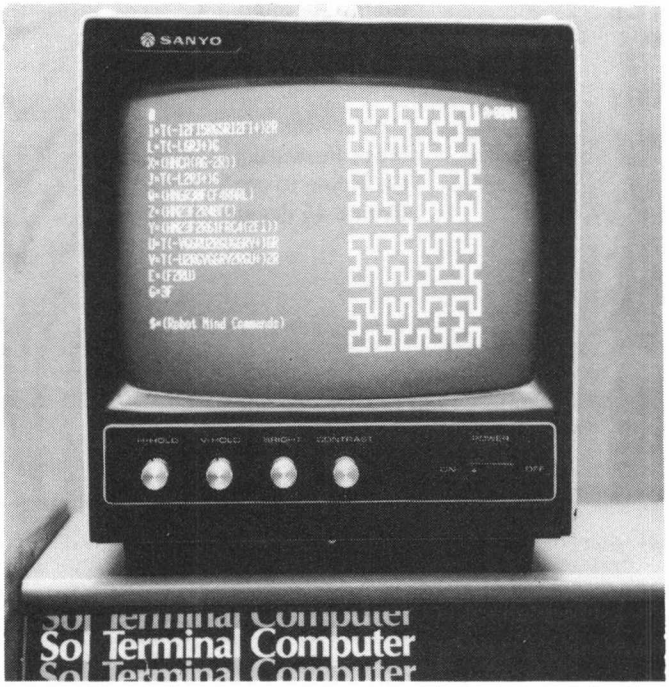
The graphics capability of 128 x 48 bit-mapped cells is used for tracing robot movement. The display is arranged to 'wrap-around' movements that go beyond screen limits.

'Body Mode' is entered whenever a body command is executed. This mode is maintained until the next keypress, which returns to 'Mind Mode'.

Body commands are identical to those previously described in *DDJ*.

One additional command is:

S Switch trace—the trace is left as per definition W or B until it crosses over itself—then the intersection is switched B/W or W/B. This command allows complicated patterns to develop without eventual 'white-out' of the display.



Sol with GraphicAdd demonstrating ROBOT program. A mix of Body and Mind modes was arranged by: Z to initialize movement then b to dump Mind on display then E to map movement without first clearing display (i.e., Mind)

Robot Command Summary

MIND

(space)	No operation
+ (or ;)	Increment accumulator
–	Decrement accumulator
A	Current value of accumulator
T <then><else>	Test if A>0 : then <then> if A = 0 : then <else>
? <then><else>	Randomly choose <then> or <else>
D <char><commands>	Define <character> to be <commands> NOTE: recursive macros are allowed!
n <command>	repeat <command> n times where n = integer or A: current value

EDITING

DEL	Erase line
← (or CTL A)	Erase character, but not brackets
CTL B	Brainwash mind
ESC	Escape to monitor

BODY

F	Move Forward one grid cell
H	Go Home to center screen
R	Rotate 45 degrees clockwise
N	Face North (up)
W	Leave White trace
B	Leave Black trace
S	Switch trace state on overlaps
C	Clear screen (to B or W)

Examples:

1. Windmill

DX (CA –8 (HNARP+))	
DP (. . . . .)	where P is a curve
X	draws a windmill

2. Spiral

DGF	step length
DM (HN 23F CA (AG –2R))	
A–	zero accumulator
50+	set A = 50
M	draws a spiral

3. Hilbert Curve

```
DUT (–V G 6R U 2R G U G 6R V +) 6R
DVT (–U 2R G V G 6R V 2R G U +) 2R
DG4F
DZ (HN 23F 2R 63F C F 2R U)
A–
5+                  set A = 5
Z                   draw curve
```

```
ASSEMBLER LISTING OF 'YET ANOTHER BODY'

NOTE: 'SCREEN' = MIND LOCATION IN RAM
            = 1C00 - 1FFF
      'VDM'  = DISPLAY RAM ADDRESS
            = CC00 - CFFF

      *PATCHES TO TINMAN
       CHANGE ALL SCREEN REFERENCES 'CC' TO '1C'
       BY LOADING AND EXECUTING:

0000                    0010 CHANG  ORG   0CA00H          *RAM AREA
CA00 21 00 00           0020        LXI   H,0             *START AT BEGINNING
CA03 7E                 0030 TEST   MOV   A,M             *GET BYTE
CA04 FE CC              0040        CPI   0CCH            *SCREEN REF?
CA06 C2 0B CA           0050        JNZ   DONE            *NO,CHECK IF DONE
CA09 36 1C              0060        MVI   M,1CH           *YES,REPLACE WITH NEW ADDRES
CA0B 23                 0070 DONE   INX   H               *MOVE AHEAD
CA0C 7C                 0080        MOV   A,H
CA0D FE 02              0090        CPI   2               *DONE 0-1FF?
CA0F C2 03 CA           0100        JNZ   TEST            *NO,CONTINUE
CA12 C3 01 C0           0110        JMP   EXIT            *EXIT TO O.S.
CA15                    0120 *
CA15                    0130 *OTHER PATCHES TO TINMAN
CA15                    0140 *
CA15                    0150 PATCH  ORG   0
000C 31 FC 1B           0160        LXI   SP,STACK+2      *NEW STACK LOCATION
0003 AF                 0170        XRA   A
0004 D3 FE              0180        OUT   DSTAT           *ZERO VDM SCROLL REG
0006 CD EF 02           0190        CALL  INIT            *STARTUP ROUTINES
0009                    0200        ORG   24H
0024 31 FA 1B           0210 DIREC  LXI   SP,STACK        *NEW STACK LOCATION
0027                    0220        ORG   55H
0053 CD D9 02           0230        CALL  KINP            *GET SOL KEYBOARD
0056 00                 0240        NOP
0057                    0250        ORG   70H
0070 21 EF FB           0260        LXI   H,-DANGER       *NEW DANGER LIMIT
0073                    0270        ORG   95H
0095 CD BF 02           0280        CALL  MDUMP           *DUMP MIND TO DISPLAY
0098                    0290 RNDB   EQU   146H
0098 CD 46 01           0300 G2     CALL  RNDB            *GET KEY INPUT
009B CD D9 02           0310        CALL  KINP
009E CA 98 00           0320        JZ    G2
00A1                    0330        ORG   0ABH
00AB C3 34 03           0340        JMP   BACKUP          *TO BACKUP PATCH
00AE                    0350        ORG   0C8H
00C8 14                 0360        DB    20              *NEW TABLE SIZE
00C9                    0370 *
00C9                    0380 *EXTEND COMMAND TABLE
00C9                    0390 *
00C9                    0400        ORG   25CH
025C 43                 0410        DB    'C'             *CLEAR DISP & WAIT
025D 7A 02              0420        DW    CLEAR
025F 1B                 0430        DB    1BH             *ESCAPE TO SOLOS
0260 01 C0              0440        DW    EXIT
0262 48                 0450        DB    'H'             *HOME
0263 6A 02              0460        DW    HOME
0265 4E                 0470        DB    'N'             *NORTH
0266 93 02              0480        DW    NORTH
0268 46                 0490        DB    'F'             *FORWARD
0269 4C 03              0500        DW    FORWD
026B 52                 0510        DB    'R'             *RIGHT TURN 45
026C 9D 02              0520        DW    RIGHT
026E 02                 0530        DB    'B'-40H         * CTL B BRAINWASH
026F 09 00              0540        DW    WASH
0271 42                 0550        DB    'B'             *BLACK TRACE
0272 1B 03              0560        DW    BLK
0274 57                 0570        DB    'W'             *WHITE TRACE
0275 0C 03              0580        DW    WHT
0277 53                 0590        DB    'S'             *SWITCH
0278 2A 03              0600        DW    SWITCH
027A                    0610 *
027A                    0620 *THESE ROUTINES ARE LIKE
027A                    0630 *THOSE OF 'ANOTHER BODY'
```

```
027A                    0640 *
027A                    0650 *
027A                    0660 *CLEAR DISPLAY
027A                    0670 *
027A C8                 0680 CLEAR  RZ
027B E5                 0690        PUSH  H
027C 21 00 CC           0700 CDISP  LXI   H,VDMEM      *GET DISP ADDRESS
027F 3E D0              0710        MVI   A,VDMH+4     *END ADDRESS
0281 36 20              0720 CD1    MVI   M,20H        *BLANK
0283 23                 0730        INX   H
0284 BC                 0740        CMP   H
0285 C2 81 02           0750        JNZ   CD1
0288 E1                 0760        POP   H
0289 C9                 0770        RET
028A                    0780 *
028A                    0790 *HOME TO CENTRE SCREEN
028A                    0800 *
028A C8                 0810 HOME   RZ
028B E5                 0820        PUSH  H
028C D5                 0830        PUSH  D
028D 21 40 48           0840        LXI   H,4840H      *CENTRE BLOCK OF DISPLAY
0290 C3 66 03           0850        JMP   DISP
0293                    0860 *
0293                    0870 *FACE NORTH
0293                    0880 *
0293 C8                 0890 NORTH  RZ
0294 AF                 0900        XRA   A
0295 32 FE 1B           0910        STA   DX
0298 3C                 0920        INR   A
0299 32 FF 1B           0930        STA   DY
029C C9                 0940        RET
029D                    0950 *
029D                    0960 *ROTATE RIGHT 45 DEGREES
029D                    0970 *
029D C8                 0980 RIGHT  RZ
029E E5                 0990        PUSH  H
029F 2A FE 1B           1000        LHLD  DX
02A2 7C                 1010        MOV   A,H
02A3 85                 1020        ADD   L
02A4 CD B4 02           1030        CALL  R1
02A7 4F                 1040        MOV   C,A
02A8 7C                 1050        MOV   A,H
02A9 95                 1060        SUB   L
02AA CD B4 02           1070        CALL  R1
02AD 67                 1080        MOV   H,A
02AE 69                 1090        MOV   L,C
02AF 22 FE 1B           1100        SHLD  DX
02B2 E1                 1110        POP   H
02B3 C9                 1120        RET
02B4 FE 02              1130 R1     CPI   2
02B6 C2 BA 02           1140        JNZ   R2
02B9 3D                 1150        DCR   A
02BA FE FE              1160 R2     CPI   -2
02BC C0                 1170        RNZ
02BD 3C                 1180        INR   A
02BE C9                 1190        RET
02BF                    1200 *
02BF                    1210 *THESE ROUTINES ARE UNIQUE
02BF                    1220 *TO 'YET ANOTHER BODY'
02BF                    1230 *BY KEN ANDERSON
02BF                    1240 *
02BF                    1250 *MIND DUMP
02BF                    1260 *THIS ROUTINE CONTROLS DISPLAY UPDATING
02BF                    1270 *FOR 'MIND MODE'.UPDATING IS BLOCKED
02BF                    1280 *DURING 'BODY MODE'.
02BF                    1290 *
02BF 3A B2 03           1300 MDUMP  LDA   MINDF        *DISPLAY IN MIND MODE?
02C2 B7                 1310        ORA   A
02C3 C8                 1320        RZ                 *SORRY, BODY MODE - DONE
02C4 E5                 1330        PUSH  H
02C5 D5                 1340        PUSH  D            *YES, DUMP MIND
02C6 21 00 1C           1350        LXI   H,SCREEN     *FROM CORTEX
02C9 11 00 CC           1360        LXI   D,VDMEM      *TO DISPLAY
02CC 7E                 1370 MOVE   MOV   A,M
02CD 12                 1380        STAX  D
02CE 13                 1390 MOV    INX   D
02CF 23                 1400        INX   H
02D0 3E 20              1410        MVI   A,SCREH+4    *SCREEN HI ADDRESS LIMIT
02D2 BC                 1420        CMP   H
02D3 C2 CC 02           1430        JNZ   MOVE
02D6 D1                 1440        POP   D
02D7 E1                 1450        POP   H
02D8 C9                 1460        RET
02D9                    1470 *
02D9                    1480 *KEYBOARD INPUT ROUTINE
02D9                    1490 *FIXES ';' TO '+'
02D9                    1500 *
02D9 DB FA              1510 KINP   IN    KSTAT
02DB 2F                 1520        CMA                *FOR SOL ONLY
02DC E6 01              1530        ANI   1
02DE C8                 1540        RZ
02DF DB FC              1550        IN    KDATA
02E1 E6 7F              1560        ANI   7FH
02E3 FE 3B              1570        CPI   ';'          *SEMICOLON?
02E5 C2 EB 02           1580        JNZ   K1
02E8 B7                 1590        ORA   A            *RESET Z FLAG
02E9 3E 2B              1600        MVI   A,'+'        *YES, REPLACE WITH +
02EB 32 B2 03           1610 K1     STA   MINDF        *SET MIND FLAG TO UPDATE
02EE C9                 1620        RET
02EF                    1630 *
02EF                    1640 *INITIALIZATION ROUTINE
02EF                    1650 *IT ASKS IF GRAPHICS ARE PROGRAMMABLE &
02EF                    1660 *ENABLES GRAPHICS IF SO; SUBSEQUENT
02EF                    1670 *RE-ENTRIES BYPASS THE QUESTION.
02EF                    1680 *IF A VALID MEMORY EXISTS IN 'SCREEN'
02EF                    1690 *THEN IT IS RETAINED, OTHERWISE MIND
02EF                    1700 *IS BRAINWASHED. MEMORY CAN BE STORED
02EF                    1710 *AND LOADED INDEPENDENTLY OF TINMAN.
02EF 3A B3 03           1720 INIT   LDA   MSGF         *IS THIS A RE-ENTRY?
02F2 B7                 1730        ORA   A
02F3 CC B4 03           1740        CZ    MSG          *NO,ASK QUESTION
02F6 FE 59              1750        CPI   'Y'          *ANSWER IS YES?
02F8 C2 FF 02           1760        JNZ   INIT1        *NO,SKIP
02FB 3E 10              1770        MVI   A,10H        *YES,SET GRAPHICS
02FD D3 FE              1780        OUT   DSTAT        *GRAPHICS ON!
02FF                    1790 *TEST CORTEX TO SEE IF MEMORY IS VALID
02FF 11 BF C2           1800 INIT1  LXI   D,-'=A'
0302 2A 3A 1C           1810        LHLD  SCREEN+58
0305 19                 1820        DAD   D            *KNOWS 'A='?
0306 7C                 1830        MOV   A,H
0307 B5                 1840        ORA   L
0308 C0                 1850        RNZ                *NO BRAINWASH
0309 C3 24 00           1860        JMP   24H          *YES, RETAIN
030C                    1870 *
030C                    1880 *WHITE TRACE,BLACK BACKGROUND
030C                    1890 *
030C C8                 1900 WHT    RZ
030D 3E C0              1910        MVI   A,0C0H       *SET BACKGROUND
030F 32 82 02           1920        STA   CD1+1        *TO BLACK
0312 E5                 1930        PUSH  H
0313 21 B6 00           1940        LXI   H,0B6H       *ORA M INST
0316 22 A2 03           1950        SHLD  BORW         *TRACE WHITE
0319 E1                 1960        POP   H
031A C9                 1970        RET
031B                    1980 *
031B                    1990 *BLACK TRACE,WHITE BACKGROUND
031B                    2000 *
031B C8                 2010 BLK    RZ
031C 3E FF              2020        MVI   A,0FFH       *WHITE BACKGROUND
031E 32 82 02           2030        STA   CD1+1
0321 E5                 2040        PUSH  H
0322 21 B6 AE           2050        LXI   H,0AEB6H     *ORA M,XRA M INST
0325 22 A2 03           2060        SHLD  BORW         *MAKES TRACE BLACK
0328 E1                 2070        POP   H
0329 C9                 2080        RET
032A                    2090 *
032A                    2100 *SWITCH TRACE STATE
032A                    2110 *
032A C8                 2120 SWITC  RZ
032B E5                 2130        PUSH  H
032C 21 AE 00           2140        LXI   H,0AEH       *XRA M INST
```

```
032F 22 A2 03           2150         SHLD  BORW          *SWITCH CELL
0332 E1                 2160         POP   H
0333 C9                 2170         RET
0334                    2180 *
0334                    2190 *CURSOR BACKUP
0334                    2200 *
0334 FE 01              2210 BACKU   CPI   1             *BACK-ARROW?
0336 C2 49 03           2220         JNZ   SAVE
0339 3E 20              2230         MVI   A,20H
033B 12                 2240         STAX  D             *CLEAR CURSOR
033C 1B                 2250         DCX   D             * & BACKUP
033D 1A                 2260         LDAX  D
033E FE 28              2270         CPI   '('
0340 CA 4A 03           2280         JZ    NO
0343 FE 29              2290         CPI   ')'
0345 CA 4A 03           2300         JZ    NO
0348 C9                 2310         RET
0349 12                 2320 SAVE    STAX  D             *NORMAL XFRE
034A 13                 2330 NO      INX   D             *TO SCREEN
034B C9                 2340         RET                 *DONE
034C C8                 2350 FORWD   RZ
034D E5                 2360         PUSH  H
034E D5                 2370         PUSH  D
034F 2A FC 1B           2380         LHLD  X             *GET PRESENT COORDINATES
0352 EB                 2390         XCHG
0353 2A FE 1B           2400         LHLD  DX            *GET DIRECTION
0356 7D                 2410         MOV   A,L
0357 83                 2420         ADD   E             *MOVE ROBOT
0358 E6 7F              2430         ANI   7FH           *NORMALIZE NEW H COORD
035A 6F                 2440         MOV   L,A
035B 7C                 2450         MOV   A,H
035C 82                 2460         ADD   D
035D 1E 30              2470         MVI   E,30H
035F 93                 2480 SUBT    SUB   E
0360 D2 5F 03           2490         JNC   SUBT          *NORMALIZE VERTICAL COORD
0363 83                 2500         ADD   E
0364 83                 2510         ADD   E             *MOVE IT TO RANGE 30-5F
0365 67                 2520         MOV   H,A
0366 22 FC 1B           2530 DISP    SHLD  X
0369 1E 30              2540         MVI   E,30H         *FOR DISP ENTRY
036B 93                 2550         SUB   E
036C 93                 2560         SUB   E
036D C5                 2570         PUSH  B
036E 01 03 FF           2580         LXI   B,0FF03H      *CLEAR COUNTER,DIVISOR=3
0371 2F                 2590         CMA                 *INVERT ROW 0 TO SCREEN TOP
0372 04                 2600 DIVID   INR   B
0373 91                 2610         SUB   C
0374 D2 72 03           2620         JNC   DIVID         *B=CHAR LINE: 0-15
0377 81                 2630         ADD   C             *C=CELL LEVEL: 0-2
0378 4F                 2640         MOV   C,A
0379 7D                 2650         MOV   A,L           *SAVE H COORD
037A 68                 2660         MOV   L,B           *GET V COORD
037B 26 00              2670         MVI   H,0           *CLEAR ADDEND
037D 29                 2680         DAD   H
037E 29                 2690         DAD   H
037F 29                 2700         DAD   H             *MULT X 64
0380 29                 2710         DAD   H
0381 29                 2720         DAD   H
0382 29                 2730         DAD   H
0383 47                 2740         MOV   B,A           *SAVE H COORD AGAIN
0384 1F                 2750         RAR                 *H COORD/2
0385 16 CC              2760         MVI   D,VDMH        *GET DISPLAY ADDRESS
0387 5F                 2770         MOV   E,A
0388 19                 2780         DAD   D             *ADD HORIZ ADDRESS
0389 EB                 2790         XCHG
038A 78                 2800         MOV   A,B           *GET H COORD
038B 1F                 2810         RAR                 *SAVE LSB
038C 79                 2820         MOV   A,C           *SET UP V
038D 17                 2830         RAL                 *HORIZ CELL BITS
038E 21 AC 03           2840         LXI   H,MTABLE      *MASK TABLE BASE
0391 06 00              2850         MVI   B,0
0393 4F                 2860         MOV   C,A
0394 09                 2870         DAD   B             *GET BIT MASK FROM
0395 1A                 2880         LDAX  D             *GET PRESENT CONTENTS OF DISP
0396 E6 C0              2890         ANI   0C0H          *GRAPHICS BYTE?
0398 FE C0              2900         CPI   0C0H
039A CA A0 03           2910         JZ    GCHAR         *YES, OK
039D 3E C0              2920         MVI   A,0C0H        *SET BACKGROUND
039F 0E                 2930         DB    0EH           *SKIP
03A0 1A                 2940 GCHAR   LDAX  D             *GET GRAPHICS BYTE
03A1 B6                 2950         ORA   M             *MASK CELL WHITE
03A2 00                 2960 BORW    NOP                 *OR XRA M SWITCH
03A3 12                 2970         STAX  D             *PLOT BIT
03A4 AF                 2980         XRA   A
03A5 32 B2 03           2990         STA   MINDF         *SET BODY MODE
03A8 C1                 3000         POP   B
03A9 D1                 3010         POP   D
03AA E1                 3020         POP   H
03AB C9                 3030         RET
03AC                    3040 *THIS IS THE CELL MASK TABLE
03AC                    3050 *FOR GRAPHICADD BIT-MAPPED GRAPHICS
03AC 01                 3060 MTABL   DB    1             *CELL 1
03AD 08                 3070         DB    8             *CELL 4
03AE 02                 3080         DB    2             *CELL 2
03AF 10                 3090         DB    10H           *CELL 5
03B0 04                 3100         DB    4             *CELL 3
03B1 20                 3110         DB    20H           *CELL 6
03B2 01                 3120 MINDF   DB    1             *MIND MODE FLAG
03B3 00                 3130 MSGF    DB    0             *DISPLAY MESSAGE FLAG ON ENTR
03B4                    3140 *
03B4                    3150 *DISPLAY MESSAGE & AWAIT RESPONSE
03B4                    3160 *
03B4 CD 7C 02           3170 MSG     CALL  CDISP         *CLEAR THE SCREEN
03B7 21 00 CD           3180         LXI   H,0CD00H      *START
03BA 11 D1 03           3190         LXI   D,MSGG        *SEND OUT INIT QUESTION
03BD 0E 26              3200         MVI   C,MSGL
03BF 1A                 3210 MSG1    LDAX  D
03C0 77                 3220         MOV   M,A
03C1 23                 3230         INX   H
03C2 13                 3240         INX   D
03C3 0D                 3250         DCR   C
03C4 C2 BF 03           3260         JNZ   MSG1
03C7 CD D9 02           3270 YES     CALL  KINP          *WAIT FOR ANSWER
03CA CA C7 03           3280         JZ    YES
03CD 32 B3 03           3290         STA   MSGF          *SAVE IT FOR RE-ENTRY
03D0 C9                 3300         RET
03D1 41 52              3310 MSGG    DW    'RA'
03D3 45 20              3320         DW    ' E'
03D5 4D 59              3330         DW    'YM'
03D7 20 47              3340         DW    'G '
03D9 52 41              3350         DW    'AR'
03DB 50 48              3360         DW    'HP'
03DD 49 43              3370         DW    'CI'
03DF 53                 3380         DB    'S'
03E0 20 53              3390         DW    'S '
03E2 4F 46              3400         DW    'FO'
03E4 54 57              3410         DW    'WT'
03E6 41 52              3420         DW    'RA'
03E8 45 20              3430         DW    ' E'
03EA 53 45              3440         DW    'ES'
03EC 4C 45              3450         DW    'EL'
03EE 43 54              3460         DW    'TC'
03F0 41 42              3470         DW    'BA'
03F2 4C 45              3480         DW    'EL'
03F4 3F 20              3490         DW    ' ?'
03F6                    3500 STACK   ORG   1BFAH
1BFA                    3510 SEED    DS    2
1BFC 01                 3520 X       DB    1
1BFD 01                 3530 Y       DB    1
1BFE 01                 3540 DX      DB    1
1BFF 01                 3550 DY      DB    1
1C00                    3560 DANGE   EQU   MSGG+64       *MSG IS DISPOSABLE
1C00                    3570 MSGL    EQU   38            *MESSAGE LENGTH
1C00                    3580 WASH    EQU   9             *BRAINWASH ROUTINE
1C00                    3590 EXIT    EQU   0C001H        *SOLOS ENTRY POINT
1C00                    3600 SCREE   EQU   1C00H         *MIND ADDRESS
1C00                    3610 SCREH   EQU   1CH
1C00                    3620 VDMEM   EQU   0CC00H        *DISPLAY ADDRESS
1C00                    3630 VDMH    EQU   0CCH
1C00                    3640 DSTAT   EQU   0FEH          *SOL SCROLL REGISTER
1C00                    3650 KSTAT   EQU   0FAH          *KEYBOARD STATUS REG
1C00                    3660 KDATA   EQU   0FCH          *KEYBOARD DATA PORT
```

FORTH

for Microcomputers

BY JOHN S. JAMES
1090 Miller Ave.
Berkeley, CA 94708

Forth is a unique threaded language ideally suited for microcomputers. Programs are incredibly compact; e.g. in 5K to 6K bytes you can get the interactive Forth compiler, running stand-alone as its own operating system including I/O drivers and other run-time routines, plus an assembler written in Forth (in case you want to optimize time-critical programs), virtual memory software, and a text editor. Not only does all this fit into 5K to 6K bytes (of which 4K are written in Forth), but it runs in the same space (plus 1K buffer), with no additional symbol table area, overlays, swapping, or use of any other software.

And while Forth gives all the convenience of interactive interpreters, it is very fast. For most applications, the run-time overhead is 20 to 30% for minis and 70 to 100% for micros (compared to 1000% or more which is common for interpreters). Number-crunching applications may take longer. And when Forth isn't fast enough, the system's own assembler can be used to re-code inner loops.

But the best news of all is that software development times are cut by half or more over assembly programming. The programming is entirely structured (there is no GOTO), and the resulting code is re-entrant and can be designed for PROM.

Forth has been implemented on many microcomputers. A recommended configuration is 16K bytes and a terminal; a floppy is a good idea for storing source programs (and virtual memory for programs and data is supported). Forth works well with a CRT, so hard copy is less necessary.

Forth has existed for several years, and is used commercially in a number of installations, but until recently it has been priced far out of reach of hobbyists, and even most computer experts have never heard of it. However, the language is in the public domain, and today low-cost versions are just becoming available. Also there is a new Forth Interest Group (787 Old County Rd, San Carlos, CA 94070), dedicated to letting the serious hobbyist and the professional know about Forth and how they can get it.

How it Works—Overview

Forth is very different from any other language in common use. Perhaps the nearest is Lisp, which it resembles in one way only: both are tree-structured languages which implement themselves. Forth is an extensible, interactive language.

Forth starts with about 40 *primitives*, or operations defined in machine language. New operations are defined in terms of these. (So far this is like the virtual-machine systems often used for implementing transportable languages—e.g. Tiny BASIC, or P-code for Pascal. But in Forth, as soon as a new operation is defined, it can be used in further definitions or executed immediately, exactly as if it were a primitive).

There are *no argument lists* in Forth; instead, all routines (operations) communicate through a stack. The programmer is allowed to define variables, but in Forth, variables themselves *are* operations which place addresses (or other values) onto the stack—and which may use the full power of Forth or of machine language to compute these addresses. This means that users can create their own variable types.

Forth comes with about 140 routines already defined (including the 40 primitives mentioned above). Programming consists of defining new routines in terms of previously-defined ones, until finally one of them is the whole program which is desired. It then is executed by typing its name. Incidently, the programmer does not need to know about all 140 operations, because some of them are used mainly by the system. After all, these 140 implement the compiler, text editor, assembler, stand-alone operating system and run-time software, etc.

Execution is fast because at run time all definitions are stored as strings of addresses; and it takes as little as two instruction executions (e.g. on the PDP-11) to go from one primitive to the next. Higher levels of nesting take a little longer, but are done less often.

Debugging is exceptionally convenient because of the modularity of programs. Almost any routine can be executed by typing test-data arguments (which places their addresses or values onto the stack), then typing the name; no driver program is required. In practice, most Forth routines are very short, no more than five lines of source code, often one line. Each routine is checked out independently, then saved in an application-oriented library for future use.

Blanks are the standard delimiters, so *any* string of non-blank characters can be an operation name. Examples of actual names are 'ASSEMBLER', 'ELSE', '!', '(', '*', '1+'. Even numbers can be defined as operations. In one practical joke, '0' was defined as seventeen. Operation names can be encrypted with embedded non-printing characters.

Example: A Small Program

Forth uses postfix notation like the Hewlett-Packard stack calculators. This means that operands are written first, followed by operators. E.g. '(A+B) * C' in regular (infix) notation becomes 'A B+C*' in postfix. This expression is executed as follows: place A onto the stack; place B onto the stack; add the two values on top of the stack, destroying both of them and placing their sum on the stack; place C onto the stack; multiply the top two values on the stack, replacing them by their product. Notice that there are no parentheses in postfix. Also, Forth follows the convention of leaving the operands in the same order as in the infix expression; this makes it easy to remember the order of arguments for non-commutative operations like subtraction. Another Forth convention is that all operations should destroy all of their arguments on the stack.

Now let's look at a complete Forth program. Assume that the system is in the machine, but not any user programming. The task is to write a routine to evaluate the polynomial $5X^2 - 7X - 10$. The user types

```
: POLY DUP DUP * 5 * SWAP 7 * -10 - ;
```

This is the complete program. Let's follow how it works in detail.

The colon ':' is one of the Forth operations called *defining words*. A defining word takes the next word in the input stream (here 'POLY'), and prepares an entry for it in the *dictionary* of Forth definitions. As explained later, the dictionary is the most central concept in Forth. In fact, in the entire Forth system, including compiler, text editor, operating system, etc. and also the user's application software, less than 50 bytes of code are not part of the dictionary.

Besides opening a dictionary entry, the colon also places the system into *compile mode*, so that operations which follow are compiled into the definitions, instead of being executed immediately. It is, however, possible to have special immediate operations within definitions, when compile-time execution is desired. The system stays in compile mode until the definition is terminated by a semicolon. Notice that blanks must be used as delimiters for colon, semicolon, and all other operations.

'DUP' is an operation to duplicate the top of the stack, i.e. to push a copy of the top item onto the stack. 'SWAP' reverses the top two items on the stack, '*' and '–' are multiply and subtract.

For this illustration, assume that 'POLY' is executed with the argument '6' on the stack. First the '6' is duplicated twice, leaving three copies of '6' on the stack. Then multiplication is executed, leaving the stack as follows:

```
36
 6
```

And '5*' pushes '5' and multiplies, leaving $5X^2$ (180) on top of the stack, with the copy of X beneath.

'SWAP 7*-' brings the copy of X to the top of the stack, multiplies by 7, and subtracts. '10 -' completes the polynomial.

To test the program, the user could type:

```
3 POLY .
```

where '.' is the command to print the value of top of the stack (the '.' could have been included in the definition of 'POLY'). The system responds:

```
14 OK
```

where 'OK' is the prompt meaning that the system is finished and waiting for the next input. (The underline means it was typed by the system.)

The user may do some further testing:

```
75 POLY . 27590 OK
90 POLY . -25676 OK
```

Notice the overflow on the last example; the width of the stack is always 16 bits in standard Forth, although double precision or floating point can be implemented, using more than one stack entry to represent a number.

It is important to test that the routine doesn't leave anything extra on the stack, because stack overflow is not checked during execution and can be hard to diagnose. The instruction, 'S, pushes the absolute machine address of the top of the stack onto the stack. This address should be the same after a test of an operation as before (provided that arguments are supplied and results printed by the test). E.g. to test 'POLY':

```
'S . 174 OK
8 POLY . 254 OK
'S . 174 OK
```

Example: Number Bases

Any number base can be used: 'DECIMAL', 'HEX', 'OCTAL', 'BINARY', '5 BASE', '17 BASE', etc. For bases above 16, letters after 'F' in the alphabet will be used as numbers. The base can be changed at any time. Only the terminal I/O is affected—all internal work is in binary, so there is little cost for arbitrary number bases. E.g.

```
DECIMAL 666 23 BASE ! . DECIMAL 15M OK
```

This code means: make sure the system is in decimal; enter decimal 666, then 23, onto the stack; enter the address of the system's variable 'BASE' onto the stack; store (the '!' operation) 23 into 'BASE', which sets the base to 23 and destroys the address of 'BASE' and the number '23' on the stack; print 666 in base 23 (15M); and restore the base to decimal, for the sake of future definitions or data entry. By convention, the standard base is decimal.

Example: Scaling

Software floating point has seldom been implemented in microcomputer Forth until recently, and still is not available on most systems (it is more common on larger Forth versions for minicomputers). Instead, Forth has used some tricks for getting around the need for floating point, such as a special scaling operation '/*MOD' (multiply and then divide, leaving quotient and remainder), which keeps a 24-bit intermediate result (instead of keeping only 16 bits as would happen if regular multiply and divide operations were used). So powerful are these special operations that Forth *without* floating point has been used for such applications as non-linear least squares, Fourier transforms, and numerical integration.

For this example, we want a program to multiply a number by 3.14159. The challenge is that we are working in fixed point, using 16-bit signed numbers with maximum value of 32,767. Also, the division operation in this version of Forth only allows divisors up to 128.

Although it doesn't make much difference in this short example, the Forth programmer is strongly advised to start with a top-down approach, taking the whole task and breaking it into operations which could implement the entire job in a couple lines of code, then breaking down these operations, etc. The alternative is to start out defining operations which seem to be relevant to the applications, but may not turn out to be useful. Don't be tempted down the wrong path by the fact that the lower-level operations must be typed in first.

In this case we can break up the task of scaling by a six-digit number (which would not even fit into a word) into three scalings by two digits each. We notice that divisions are by 100, which is allowed, and that no intermediate results on the stack will be any larger than the answer, avoiding overflow problems. The final program becomes:

```
: FIRSTSCALE 31 10 */ ;
: NEXTSCALE 41 100 */ 10 / ;
: LASTSCALE 59 100 */ 100 / 10 / ;
: PI DUP FIRSTSCALE OVER NEXTSCALE +
  SWAP LASTSCALE + ;
```

Now the user can type:

```
10000 PI . 31415 OK
```

or use PI in other definitions.

The 'OVER' operation copies the second element on the stack without destroying it and pushes the copy onto the stack. The scaling operation '*/' is like '*/MOD', except that only the quotient, not the remainder, is returned.

The reader can follow through this example. Typing, e.g., '10000 PI .' first pushes 10000 onto the stack, then executes 'PI'. 'PI' duplicats the stack, then executes 'FIRSTSCALE', which pushes 31, then 10, then executes the scaling operation which multiplies 10000 by 31 (keeping 24 bits for the result), then divides by 10, returning a 16-bit result. Now the stack is:

31000
10000

and execution continues with 'OVER', 'NEXTSCALE', '+', etc;

For efficiency, this example could be improved by first translating 3.14159 to octal or hex, then using Forth's assembler to write a divide-by-shifting routine, as the standard divide software is not smart enough to recognize that a divisor is a power of two. However, run-time speed would *not* be much improved by defining this program in one operation instead of four, because linkage between routines is very fast. The overhead in number-crunching applications is due more to stack manipulation than to linkage. And when time is critical, the Forth assembler is convenient.

Example: Complete Session

The 'PI' example illustrated some practical advantages of Forth for software development. It is no surprise that this program worked on the first try. Testing was immediate. The operation is now available for any further use, simply by writing its name. This routine takes about 75 bytes of object code; the compactness would be more impressive it it were part of an applications package in which references to 'PI', which take two bytes, were used a few times.

The following listing shows the same example, but in a complete terminal session including use of the editor. This session starts after the Forth system has been loaded. Loading the system takes about 30 seconds on the 6502; this includes compilation of all but the first 2K bytes, which must be precompiled because it includes operations necessary for the system to perform compilation. The rest of the system code is left in source form, to allow user modifications. Because compilation is fast, there is no need to store object code on disk, hence no need for relocation. While overlays can be defined, Forth is so compact that they are seldom needed.

```
42 LIST
        SCR # 42

  0
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
OK

EDITOR OK
0 P       ( MULTIPLY BY 3.14159 ) OK
1 P : FIRSTSCALE 31 10 */ ; OK
2 P : NEXTSCALE 41 100 */ 10 / ; OK
3 P : LASTSCALE 59 100 */ 100 / 10 / ; OK
4 P : PI DUP FIRSTSCALE OVER NEXTSCALE + SWAP LASTSCALE + ; OK
42 LOAD OK
10000 PI . 31415 OK
'S . 176 OK
45 PI . 140 OK
'S . 176 OK
```

The user listed a disk screen (here number 42) to confirm that it was empty. Then the word 'EDITOR' invoked the editor vocabulary. This does not load the editor, which is compact enough to normally reside in RAM (though it can be loaded when used, if memory is tight). Instead, 'EDITOR' caused the editor operations to be recognized.

The program is typed using the editor's 'P' command, which places the new line being typed into the line number given, destroying whatever may have been in that line before. There are several other editor commands not shown here. This is a line editor, and cannot modify within a line. A more sophisticated string editor has been developed.

'42 LOAD' causes the disk screen to be input just as if the information had been typed; this causes the compilation to be performed. Some tests of 'PI' are shown. The answer is correct (though not rounded in the last place); and the operation does not leave anything extra on the stack, or take anything away except its argument.

Example: Using the Assembler

To invoke the assembler, begin a definition with 'CODE' instead of ':'. The manufacturer's mnemonics are available, plus 'IF,', 'ELSE,', and 'THEN,' for conditional branching, and 'BEGIN,' 'END,' for conditional looping. These structured primitives can be nested to at least 16 levels, because a stack is used at assembly time. The assembly instructions must be entered in postfix: operand, mode indicator if any, then op code. Incidently this entire assembler as implemented in Forth takes only a few hundred bytes, mostly for storing the manufacturer's mnemonics.

An example of a code definition (for the 6502) is:

```
CODE DIV256 0 ,X LDA, 1 ,X STA,
    0< IF, FF # LDA, ELSE, 00 # LDA, THEN,
    0 ,X STA, NEXT JMP,
```

The operation 'DIV256' is here defined as a fast divide by 256. It is used to same way as any other operation; it is transparent to the user whether a routine is implemented in Forth or in code. For examples of use, see below.

Note that the op codes and some other names end in commas. In this version of Forth, it is conventional that operation names which actually place information into the code stream end with commas; these operations serve as the end of a phrase. The other names and numbers place values onto the stack, which the comman operations will use. For example, '0<' places a value (an op code) that sets up what 'IF,' will test for.

Though not used in this example, the full power of Forth is available at assembly time for computing address displacements. Use of the assembler does *not* enter into any special assembly or compile mode (it does invoke the assembler vocabulary so that assembly mnemonics are recognized). Therefore, all Forth operations are available for execution as normal.

The assembler's conditionals (e.g. 'IF,') are simply Forth operations which are executed when typed, like any others. They are like macros, placing one or more machine-language instructions into the dictionary where the definition is being entered. Sometimes a forward branch is required (e.g. by 'IF,'), and of course the address to branch to is unknown at the time. The macro simply leaves an empty space for an address or a byte count, and also puts the address of that empty space onto the stack. Later, the operation which marks the branch-to point simply takes that address off the stack, and patches in the final offset or address required. Incidently, the advanced programmer can deliberately produce non-structured control, by manipulating the stack during assembly.

Though not shown here, the assembler also has a 'BEGIN,' ... 'END,' looping structure, which repeats until the value on top of the stack becomes non-zero (true).

The following session entered and tested the definition of 'DIV256'.

```
EDITOR OK
0 P        ( ASSEMBLER EXAMPLE ) OK
1 P HEX OK
2 P CODE DIV256 0 ,X LDA, 1  ,X STA, OK
3 P       0< IF, FF # LDA, ELSE, 00 # LDA, THEN, OK
4 P       0 ,X STA NEXT JMP, OK
FLUSH OK
43 LOAD STA ?
EDITOR OK
SCR ? 43 OK
4 P 0 ,X STA, NEXT JMP, OK
FLUSH OK
43 LOAD OK
77 DIV256 . 0 OK
300 DIV256 . 3 OK
DECIMAL OK
77 DIV 256 . DIV ?
77 DIV256 . 0 OK
400 DIV256 . 1 OK
10000 DIV256 . 39 OK
-300 DIV256 . -2 OK
-3 DIV256 . -1 OK
-256 DIV256 . -1 OK
-257 DIV256 . -2 OK
```

This session has some errors; e.g. in the sixth line, 'STA' has the comma omitted. The error message 'STA ?' was printed in the eighth line. In the seventeenth line, mistyping the operation name caused 'DIV' to be unrecognized. Notice that 'HEX' stayed in effect after the code entry, until the 'DECIMAL' operation in line 16. Also, all quotients are truncated downward, even negative quotients, e.g. '-3 DIV256 . -1 OK' 'FLUSH' writes the updated buffers to disk, making sure that the copy of the program is actually saved. 'SCR .' places the address of the system's variable 'SCR', which holds the number of the disk screen being worked on, onto the stack, then '?' takes the address off the top of the stack, and prints the quantity at that address.

Example, LIFE

The following Life program was contributed by Dave Boulton, of Quesnoy Corp. Object code is 475 bytes; Forth works better for larger programs where application-oriented libraries can be developed. This program took an hour and a half to write and five minutes to debug. It runs slowly, and could be speeded up by re-coding the two levels of inner loop in assembly.

```
100 LIST 101 LIST
  0 ( THE GAME OF LIFE )  FORTH DEFINITIONS FORGET TASK : TASK ;
  1 81 CONSTANT XLEN  25 CONSTANT YLEN
  2 XLEN 1 -  YLEN 1 -  ARRAY UNIVERSE
  3 : CHECK  DUP 3 = IF  DROP 2+
  4                  ELSE  2 = NOT
  5                     IF  4 +  THEN  THEN  ;
  6
  7 : CLEAR   YLEN 0 DO  XLEN 0 DO 0 I J UNIVERSE C! LOOP LOOP ;
  8 : SET    SWAP UNIVERSE  1 SWAP C! ;
  9 : RESET  SWAP UNIVERSE  0 SWAP C! ;
 10
 11 : DISPLAY  HOMEUP  XLEN 0 DO [ _] LOOP CR
 12            YLEN 0 DO   XLEN 0 DO
 13               I J UNIVERSE C@ IF [ *] ELSE SPACE  THEN
 14            LOOP CR  LOOP  ;
 15   101 LOAD

  0  ( LIFE CONTINUED )
  1 : GENERATE  YLEN 0 DO   XLEN 0 DO   0
  2                J 2 + YLEN MIN  J 1 - 0 MAX  DO
  3                J 2 + XLEN MIN  J 1 - 0 MAX  DO
  4                   I J UNIVERSE C@  1 AND +
  5                LOOP LOOP
  6             I J UNIVERSE C@  1 AND  SWAP OVER -
  7             CHECK I J UNIVERSE C!  LOOP  LOOP  ;
  8
  9 : NORMALIZE YLEN 0 DO   XLEN 0 DO
 10                I J UNIVERSE DUP C@   DUP 3 >
 11                IF  DROP 0  ELSE  DUP 1 >
 12                                  IF  DROP 1 THEN  THEN  SWAP C!
 13             LOOP  LOOP  ;
 14  : SIZE  1+ 81 MIN  ['] XLEN !  1+ 25 MIN  ['] YLEN ! ;
 15 : GENERATIONS  0 DO DISPLAY GENERATE NORMALIZE  LOOP  DISPLAY ;
 OK
```

Instead of giving a total explanation of this program, which would expand this article into a language manual, we will sketch how it works and browse some of the mysteries.

The program is on two screens, numbers 100 and 101. The first line lists them both. Each screen has line numbers 0 through 15. By convention, line 0 contains a comment describing the screen.

Forth comments are enclosed in parentheses. The left parenthesis must be surrounded by blanks. The right parenthesis need not be preceded by a blank, though by convention it usually is. The reason for this asymmetry is that the left parenthesis is a Forth operation which causes an arbitrary character string to be skipped until a right parenthesis, which is a delimiter, if found.

The 'GENERATIONS' definition, screen 101 line 15, is the main loop of the program. It contains a 'DO' . . . 'LOOP'. 'DO' takes two arguments from the stack, the initial value from the top, and a termination value below it. Looping stops one short of the termination value; e.g. '15 0 DO . . . LOOP' loops 15 times, with the index going from 0 through 14. Arbitrary computation of looping parameters is of course allowed, since 'DO' doesn't care how its arguments got on the stack.

It may seem strange to have the loop termination value before the initialization, the reverse order from BASIC and others. But this way you can say, e.g., '15 GENERATIONS', which puts 15 on the stack and executes 'GENERATIONS', which puts 0 on the stack and executes 'DO'. We more often want to pass a termination value than an initialization value to a routine.

The Forth operation 'I' places onto the stack the current value of the index of the innermost 'DO' loop. This leads to a very strange fact in lines 2 and 3 of screen 101: 'J' refers to different 'DO' loops in the two lines, because the 'DO' at the end of line 2 has changed the level of nesting.

'XLEN 1-YLEN 1-ARRAY UNIVERSE' creates an array named 'UNIVERSE', or dimensions 80 by 24 because XLEN and YLEN were created with values 81 and 25 in the preceding line. Notice that these statements are outside of any colon definition; they are executed immediately when typed, at "compile time" (although it is "run time" for the 'CONSTANT' and 'ARRAY' defining words). In Forth, compile time and run time are definitely separated yet very much interspersed).

'I J UNIVERSE C@' places the values of 'I' and 'J' on the stack, then executes 'UNIVERSE', which is an 80 by 24 byte array, but is also an operation. 'UNIVERSE' takes two index values from the stack, and returns the address of the selected byte. Then 'C@' takes whatever address is on top of the stack, and replaces it by the value of the byte at that address (in the low-order eight bits of the 16-bit top of the stack). Hence constructions like:

```
I J UNIVERSE C@ 1 AND +
```

which takes the I, J byte, logically ANDs it's value with 1, and adds the result to the top of the stack. That is, it increments a counter (on the stack) if the addressed byte has a '1' in its low-order bit. This statement is counting how many are alive in the three by three square surrounding the character in question. The character itself is counted, though it should not be, so it is subtracted out by line six of screen 101.

Line 14, screen 101, is an optional operation to allow the size of the array to be changed—but never to go over 80 by 24. The operation '[']' gets the address of whatever follows it (in this case the constant 'XLEN' or 'YLEN'). Later, '!' stores the second item on the stack into the address on top of the stack. In other words, this line changes the values of "constants" 'XLEN' and 'YLEN'. Operations like '[']' are mainly intended for use in implementing the Forth system, but they are available to the applications programmer.

Incidently, the 'ARRAY' operation (which created 'UNIVERSE') was itself defined by the programmer in two lines of code not shown here. 'ARRAY' is an example of a user-defined variabletype. The full power of Forth is available to specify what happens at compile time (e.g. allocate space for the variable, if a static variable is wanted), and at run time (e.g. compute an address in the array, given subscripts).

One more operation which should be explained is 'IF' . . . 'ELSE' . . . 'THEN', e.g. in line 13 of screen 100,

```
I J UNIVERSE C@ IF [ *] ELSE SPACE THEN
```

The 'IF' operation takes one argument off the stack. If it is non-zero (true), the phrase after the 'IF' is executed; it it is zero (false), the phrase after the 'ELSE' is executed, if there is an 'ELSE' phrase. In either case, control resumes after the 'THEN'. This line takes a character from the array, and if it is non-zero prints an asterisk (left bracket introduces a literal string to be printed, and seeks a right bracket as delimiter), or if the character is zero prints a space.

Forth isn't difficult—as this condensed description may make it seem. Just unusual.

How it Works—Internals

We don't know of any implementation documents. This section is a sketch in that direction.

The basic structure of Forth is simple. The difficult part of the language design is the selection of operations such that the language is able to implement itself gracefully. And the coding of the Forth system operations can be very dense. At this point we are concerned with the structure of the system, not the semantics of the particular operations. For a fuller description of the language itself, see the Decus manual (reference in bibliography), which to our knowledge is the most complete documentation readily available today. For a fuller description of the internals and implementation, check future publications of the Forth Interest Group, 787 Old County Rd., San Carlos, CA 94070.

The central structure of Forth is the *dictionary*. Every operation, primitive or defined, has an entry in the dictionary. All applications software, and virtually all of the Forth system, consist of dictionary entries. Of the 5K to 6K bytes of a typical microcomputer Forth system, only about 50 bytes are not part of a dictionary entry.

All dictionary entries have the same format. In most Forth systems, the first four bytes identify the word; the total length of the word, and the first three characrers, are saved. Operation names may have any length; but if they have the first three characters and length the same, they will not be distinguished. This system works quite well, and confusion is rare.

Besides the length, the first byte also has a *precedence bit*, which tells whether the operation is immediate or not. Immediate operations are executed even when the system is in compile mode, and usually act as compiler directives or macros. Compile mode is set by the colon, and cleared by the semicolon.

The next two bytes have the *link address*. This is simply a pointer to the previous dictionary entry. Its purpose is to allow the dictionary to be searched, at compile time.

The next two bytes are the *code address*. This address is fundamental. It *always* points to machine code. In a code operation (either a primitive, or a user-defined assembler routine) it points to the code which implements the operation; this code begins immediately after the code pointer. In a colon definition (i.e. a standard Forth statement), it points to code which controls nesting at run time, described below. In a variable, the situation is more complicated. In this case, the code address points to code in the dictionary entry of a

defining word—a kind of operation which (a) when used at compile time makes a new entry in the dictionary, and (b) contains code which is unique to that particular class or type of variable, and which is executed at run time.

The eight bytes described above form the *header* of the dictionary entry which defines any Forth operation—code routine, colon definition, or variable. The body of the dictionary entry consists either of machine language, a list of addresses of previously-defined operations (which may have literals interspersed, each preceded by the address of a literal-handling routine), or the value or address of the variable, array, etc., respectively.

Now we know what the dictionary looks like, for it is nothing more than a string of these dictionary entries. And the dictionary makes up virtually all of Forth—the compiler, assembler, operating system, and all application software written by the user. Only about 50 bytes of code are outside of the dictionary. This little bit of code, known as the *inner interpreter*, is the heart of the system.

At run time, all operation names have been replaced by the addresses of their dictionary entries, so the inner interpreter works with addresses only, and never has to search the dictionary. The inner interpreter uses a pointer, 'I', to the next address in a definition to be executed. If possible, 'I' will be kept in a register, because the inner interpreter must be ultra fast.

Consider a particular dictionary entry for a Forth definition, and assume that one of the operation addresses in its body is just finished being executed. 'I' now points to that address in the dictionary entry.

If the routine which is finishing is a code routine (primitive or user-defined), it will end with a machine-language branch to the location 'NEXT' in the inner interpreter. There, code will increment the 'I' pointer by two, leaving it pointing to the dictionary entry of the next operation to be executed. By an indirect fetch from 'I', the code address of this new entry is moved into another pointer called 'W'. Then the inner interpreter branches indirectly to 'W'; i.e., it branches to the code address of the next routine to be executed; the code address always points to machine code. This linkage process takes as little as two instructions on minis and eight or so instructions on eight-bit micros.

This is the linkage from one code routine to another. If the new routine is not code but is a Forth definition instead, then the code address in its dictionary header points to code which handles the nesting by pushing 'I' onto the *return stack*, then setting 'I' from 'W' to point to the address just before the body of the Forth definition, then branching to 'NEXT'. Similarly, when the Forth definition ends, the semicolon operation points to code which pops the return stack into 'I' (removing the last level of nesting), then branches to 'NEXT'. There is also some auxiliary code for performing stack operations and then falling directly through into 'NEXT', for efficiency.

This completes the description of the inner interpreter and of the nested execution of Forth operations. This linkage happens at all times—compilation, assembly, editing, or execution of applications software. In fact there is no fundamental distinction between these times, as the whole Forth system, including the user's application software, is one unit.

Miscellaneous Features

Multitasking

Forth is well suited to multitasking, which has been implemented on minis (though not as of now on micros). E.g. a Nova 2 (35K words) has handled up to 32 terminal users simultaneously. To switch users, only four pointers have to be changed. Each user has a private vocabulary, in addition to the common vocabulary. Communication between users is accomplished by simply leaving messages in agreed-upon places.

Also on minis, a data base management system has been implemented as a Forth superset.

Disk Management

Virtual-memory software is standard for both micros and minis. Data is read and written in 128-byte *blocks*. The editor addresses 8-block (1K) units, which are called *screens* because they fill 16 lines of 64 characters each on a CRT. (A CRT used on a Forth system should be able to display 16 lines of 64 characters.)

The system should have memory room for at least 8 block buffers, so that a whole screen can be displayed; the same buffers are also used for applications and disk I/O. Programs request blocks by number, without regard to whether they are in memory or must be brought in from disk; the system checks whether the block is in a buffer already, reads it from the disk if not, and returns the memory address of the block on the stack. Programs can use the 'UPDATE' command to mark a block as changed; it will be written to disk before that buffer is re-used. 'FLUSH' will write all updated buffers which are currently in memory.

The system treats input identically, whether it comes from the disk or is typed in. This allows use of 'load screens', which contain commands to load other screens, so that the system and/or applications programs can be loaded automatically.

Dictionary Management

Forth is hard to categorize as interpreter, assembler or compiler, and some have even questioned whether it is a language. Whenever a character string is typed into Forth, it is first looked up in the dictionary. Only the first three characters (five in some Forth versions) and total name length are used for the match. If the name is not in the dictionary, the system tries to make a number out of it, using the number base which is currently in effect; if that cannot be done, an error message is issued. Forth operations must be defined before they are used.

It is OK to re-define a name which is already in the dictionary. Later uses of the name refer to the new definition, but earlier uses continue to refer to the old one (these earlier references have already been compiled). This means that there is no worry about accidently re-using a name, unless the old meaning must be used again the same program. Therefore the programmer can freely make use of temporary names in building up the definitions of new operations, without worrying about later conflicts.

For convenience, it is possible to 'JOIN' non-adjacent definitions, taking all intermediate definitions out of the dictionary search path.

There is also a 'FORGET' command, to wipe out the most recent dictionary entries. It is used to clear away recent scratch work. Care is advised. Forth allows you to 'FORGET' the entire system.

But the most powerful form of dictionary management is the use of *vocabularies*. Each vocabulary is a separate linked list of dictionary entries. Vocabularies are used to control the order of dictionary search. There is an 'ASSEMBLER' vocabulary which contains the op codes, etc; and 'EDITOR' vocabulary contains text editor commands. Also, users may define their own application vocabularies. All vocabularies link into the 'FORTH' vocabulary, which is the root of the vocabulary tree. In other words, if an operation name is not found in whichever special vocabulary is in effect, the 'FORTH' vocabulary will be searched. Users can define sub-vocabularies which similarly link into each other.

Vocabularies are implemented by the link pointers in the dictionary entries. In addition, each vocabulary has a diction-

ary entry in its root vocabulary; this entry associates the vocabulary's names with a pointer to its last dictionary entry.

At any time, two vocabularies are in effect: 'CONTEXT' and 'CURRENT'. 'CONTEXT' works as described above, determining the path of dictionary search. 'CURRENT' tells which vocabulary new definitions will be linked into. If all this gets confusing, keep in mind that in any Forth system there is only one dictionary, but there may be many vocabularies. Further discussion of vocabularies is beyond the scope of this article.

Microcode

There is much interest in microcode implementation of Forth. One approach is to microcode only the inner interpreter. At most this could double the speed on micros and less on minis, so it would probably be more practical to use a faster chip instead.

The other approach is to microcode all 40 or so of the primitives, plus a few other operations for completeness (as there would be no "machine language" to drop out of "Forth" into). Such a machine would be many times faster. While work is proceeding, we don't know of anyone who has done it yet.

Transportability

Forth is easy to move to new machines, because only 1K bytes must be re-written. An experienced implementer can do it in a few weeks. Some care must be used when transporting higher-level definitions to machines of a different architecture —e.g. when going between high-order vs. low-order byte first, character arrays may need attention.

Cross-compilation is possible, but complicated because Forth operations need two definitions, a separate one for the target machine.

Complex vocabulary manipulations are used to keep the two sets of definitions separate. Cross-compilation is commonly used when the target machine is the same chip, but to run without a terminal and perhaps from ROM, e.g. for control applications. In this case the purpose is to compress the program by eliminating those primitives and definitions which are only used by the compiler, and to shorten the remaining definitions by removing the name characters and the link pointer (which are only needed during compilation).

History and Availability

Forth was developed by Charles H. Moore around 1970, at the National Radio Astronomy Observatory, and to our knowledge was first described in publication in 1973[1]. Later Mr. Moore left the NRAO and is now marketing the language through Forth, Inc., 815 Manhattan Ave., Manhattan Beach, CA 90266. The price is $10,000 for the larger, minicomputer timesharing version, including on-site training. The microcomputer version is $2,500, sent by mail and including telephone consulting. A "primer" is available separately at nominal cost. As of May 1977, Forth Inc. offered implementations on the COSMAC, 8080, 6800, Z-80, and LSI-11.

A version of Forth for the PDP-11 is available through DEC Users' Society, 126 Parker St., Maynard, MA 01754. Distribution charge is $60 for tape, or $23 for floppy. It runs under RT-11 or stand-along, and needs 8K. PDP-11 Forth is DECUS No. 11-232.

The DECUS manual is sold separately, for $2. This is the only manual we know which is readily available to the public at this time.

Also on the 11, Doug Faunt, PO 60116, Sunnyvale, CA 94088, or phone (415) 494-7030, will soon be publishing the *LSI-11 Newsletter*, can supply a DECUS floppy version in exchange for a floppy and return envelope.

For CP/M, a Forth version called STOIC is available from the CP/M Users Group, 164 W. 83rd St., New York, NY 10024. STOIC is Vol. #23, and the distribution charge is $8. Documentation is on the floppy.

The STOIC system takes about 13K for its basic definition and assembler; why is it so much larger than other systems? Several K are used for buffers to interface CP/M; and the first five characters of each name, instead of three, are saved in each dictionary definition. Also there are many more operations than in the smaller system, such as a more complete signed and unsigned arithmetic. STOIC also has additional packages for vector and matrix operations, double precision, and floating point.

Forth versions are running at various universities and research institutions; Rather and Moore[7] list three where it is available. Also, University of Rochester and University of Arizona have implementations on Varian 620-I, CDC 6600, PDP-11, and IBM370. We don't know about distribution arrangements.

For the Z-80, the Digital Group, PO 6528, Denver, CO 80206, has announced CONVERS, its version of Forth, to sell for $26, with the manual selling separately for $10, creditable toward purchase. However, the product is not yet out.

For the Alpha Micro, a version is being developed by Alpha Micro Systems, 17875 N. Sky Park, Suite N, Irvine, CA 92714.

Programma Consultants, 3400 Wilshire Blvd., Los Angeles, CA 90010, is developing Forth versions for the 6800, 6502 machines (Apple, PET, OSI), and other micros. These are intended to run without floppies.

A newly formed Forth Interest Group, 787 Old County Rd., San Carlos, CA 94070, is an information clearinghouse on the language; it does not have a version for distribution at this time.

Summary

Only recently have sophisticated programming languages become available for home computers and small development systems. There are others besides Forth, such as Pascal, or C. Probably each has certain applications which it can do best. Forth is very different from anything else. It is remarkable that so rich a language, which is also a stand-alone operating system, can be implemented with 6K bytes.

Software development times can be greatly reduced, especially for large projects. Rather and Moore[7] give as example a radiotelescope control and data analysis system, including on-line interactive graphics, which they claim would typically be estimated at six man years and with 64K of memory required. It was implemented in Forth in 24 man weeks and 16K.

1. **Moore, C.H., and E.D. Rather, "The FORTH program for spectral line observing," *Proc. IEEE*, vol. 61, September, 1973, pp. 1346-1349.**
2. **——, "FORTH: A new way to program a minicomputer," *Astron. Astrophys. Suppl.* 15 1974, pp. 497-511.**
3. **DECUS, *Forth Programming System for the PDP-11*, DECUS No. 11-232, September 1975. (Note also manual for the PDP-10.)**
4. **Stein, P., "The FORTH dimension: Mini language has many faces," *Computer Decisions*, November, 1975, pp. 10.**
5. **Rather, E.D., and C.H. Moore, "FORTH high-level programming technique on microprocessors." Presented at Electro/76 Professional Program (paper 23/4), Boston, MA, 11-14 May 1976.**
6. **Sinclair, W.S., "Forth: A Stack Oriented Language," *Interface Age*, September 1976.**
7. **——, "The FORTH approach to operating systems," *ACM '76 Proc.*, pp. 233-240, October 1976.**
8. **Forth, Inc., *Microforth Primer*, December 1976.**
9. **——, "The use of 'FORTH' in process control." Presented at International '77 Mini-Micro Computer Conference, Geneva, 26 May 1977.**

RESIDENT MEMORY TEST SYSTEMS: With An Example For The 6800

BY TERRY RITTER
Motorola, Inc.
3501 Ed Bluestein Blvd.
Austin, TX 78721
© 1978, Motorola, Inc.
Reprinted with permission

A memory test package can be a very useful utility for a microcomputer system. Not only can it provide periodic checks of memory functionality, but it can also give day-to-day checks of the system after re-configuration. Getting a moderately-complex program to actually execute says a great deal about the correctness of the entire computer. Testing the area which is supposedly RAM isolates address-decode and board-contact problems which could otherwise inhibit operations. A small and fast memory test package can confirm the correctness of a dynamically-changing system to a high degree of probability.

A resident memory test system must usually *assume* some degree of RAM correctness in order to execute the program (even if executed from ROM, the program will probably need temporary storage, and will inevitably use subroutines). Further, since RAM is being tested under the control of an instruction sequence also derived from memory, the test program cannot check access times. Dynamic RAM refresh problems are especially difficult to check, since all memory accesses should be halted while stored charge is given the opportunity to leak from inside dynamic nodes. At room temperature, this may need to be a minute or so, making the resulting test very long. Nor do most systems even consider temperature or operating voltages which will define the limits of valid memory operation. A resident memory test is functional only, and valid only at one point in time.

Nevertheless, it is possible to make a reasonably fast and effective memory test system. At the very least, each memory bit must be written to a '1' and recovered as a '1', and written to a '0' then recovered as a '0'. This is possible using a simple "write $FF, check, write $00, check" sequence, but this sequence will not detect changes in other address locations which are caused by these operations. A better method is to store a pattern over the entire test area then recheck every location after each write to a particular cell, but this is clearly an N^2 algorithm, in terms of time.

Another test method is to place a known sequence of bytes in the test area, then recover this data and compare it to the internally-generated sequence. Two passes of a known pattern can check each bit in memory. But if the pattern has a length divisible by 2—and is shorter than the tested area—the same byte values could appear in similar areas within, say, two different pages. Now, if an address-decoding fault causes a single memory block to be enabled for multiple address blocks, and similar data is stored in each address block, the error will not be apparent (since the second write pass in the enabled block writes the same data as the first pass, it may be considered not to have occurred). A testing improvement results by generating a sequence whose length is relatively-prime to the minimum enable-block size (say, 1K). Now, the same byte value cannot appear at a similar location on different multiply-decoded pages, and the error of multiple address decoding will be detected.

Once an acceptable test algorithm has been found it is necessary to consider the operator interface. It is clearly necessary to be able to define the test area, and any error displays should include both the error address and indicate the bit-in-error. The program must be operator interruptable, both to halt a long error listing, and to re-specify the test area. The operator should be able to command a return to the overall calling system. The program could be made to verify its integrity before beginning. And, if the package is position-independent, the program could be made to move to a new location to test the area where it was originally located.

Most of these principles are included in the program MTEST9. Written as an example of a standard-product ROM, it can be used with different hardware designs having ROM, RAM, and I/O at arbitrary locations. The sample output shows execution of various options, as well as the ramifications of some different types of inappropriate commands.

```
AUSTIN, TEXAS -- MICROCOMPUTER CAPITAL OF THE WORLD
M6800-M6809 CROSS-ASSEMBLER  2.0
PAGE  001     MTEST9  PSEUDO-RANDOM MEMORY TEST

00001                            NAM   MTEST9
00002                    *
00004                    * COPYRIGHT (C) 1978 MOTOROLA INC, AUSTIN, TX
00005                    *  MPU SYSTEMS DESIGN, T. F. RITTER
00006                    *  3.0/01/27/78/TFR
00007                    *  3.1/02/08/78/TFR+RMK


00009                    *     MTEST9 IS A FAST MEMORY TEST SYSTEM.  IT HAS
00010                    *  SELF-CONTAINED I/O, IS COMPLETELY POSITION-
00011                    *  INDEPENDENT, AND USES NO ABSOLUTE RAM.  IT MAY
00012                    *  BE PLACED IN UNDER 1K OF ROM.
00013                    *
00014                    *     MTEST9 IS ENTERED AT ITS FIRST LOCATION,
00015                    *  AND ASKS FOR START/STOP ADDRESSES FOR THE
00016                    *  TEST.  THE LAST FOUR HEX CHARS BEFORE (CR)
00017                    *  ARE ACCUMULATED; A NULL ENTRY PRESERVES THE
00018                    *  ORIGINAL ADDRESSES.  IF AN 'M' IS ENTERED,
00019                    *  MTEST9 WILL COPY ITSELF INTO A NEW LOCATION
00020                    *  BEGINNING AT THE CURRENT START ADDRESS, AND
00021                    *  RESTART AT THAT ADDRESS.
00022                    *
00023                    *     MTEST9 STORES A SEQUENCE OF BYTES THROUGH-
00024                    *  OUT THE MEMORY TEST AREA, THEN COMPARES THE
00025                    *  RECOVERED SEQUENCE TO THE INTERNALLY-GENERATED
00026                    *  SEQUENCE.  ANY ERRORS CAUSE DISPLAY OF THE
00027                    *  ERROR ADDRESS AND THE BITS IN ERROR; ALL STUCK
00028                    *  BITS AND IMPROPER ADDRESS-DECODE ERRORS CAN
00029                    *  BE FOUND, AND SOME PATTERN-SENSITIVITIES ARE
00030                    *  ALSO EXERCISED.  AN 'X' IS PRINTED FOR EACH
00031                    *  PASS THROUGH MEMORY;  EIGHT X'S IS A FUNCTIONAL
00032                    *  TEST,  AND 'ALL DONE!' WILL PRINT AFTER THE
00033                    *  FULL SEQUENCE OF 211 PASSES; THEN MTEST9 WILL
00034                    *  START OVER.  AN (ESC) ALWAYS RESTARTS MTEST9;
00035                    *  (CONTROL X) ALWAYS RETURNS TO THE CALLING
00036                    *  SYSTEM (MAID, IN THE EXORCISOR).
00037                    *
00038                    *     A SHORT INITIALIZATION ROUTINE IS USED TO
00039                    *  CONFIGURE MTEST9 FOR THE EXORCISOR; CONTROL
00040                    *  THEN FALLS INTO M0, WHICH IS THE GENERAL TEST
00041                    *  SYSTEM.  DIFFERENT HARDWARE CONFIGURATIONS
00042                    *  NEED ONLY SET UP THE STACK, PUSH A ZERO MODE
00043                    *  BYTE, PUSH THE ABSOLUTE ADDRESSES OF THE ACIA
00044                    *  CONTROL AND DATA PORTS, THEN CALL TVM0 AT
00045                    *  MTEST+3.  ALTERNATELY, PUSHING A NON-ZERO
00046                    *  MODE BYTE AND ABSOLUTE ADDRESSES OF INPUT AND
00047                    *  OUTPUT ROUTINES WILL ALLOW ALL I/O TO BE DONE
00048                    *  EXTERNALLY (NOTICE THE SPECIAL PARAMETER
00049                    *  REQUIREMENTS OF INCH: ACCA IS SENT TO INCH AS
00050                    *  A PARAMETER.  IFF B7 OF ACCA IS 0, INCH WILL
00051                    *  WAIT FOR A NEW CHAR.  IFF ACCA=0, INCH WILL
00052                    *  ECHO CHAR TO OUTCH.  INCH RETURNS THE RECOVERED
00053                    *  CHAR IN ACCA.)
00056          F11E      MAID   EQU   $F11E     REENTRY ADDRESS
00057          000D      CR     EQU   $0D       ASCII CR
00058          0D0A      CRLF   EQU   $0D0A     ASCII CRLF
00059          0018      CTLX   EQU   $18       ASCII CANCEL (CONTROL X)
00060          001B      ESC    EQU   $1B       ASCII ESCAPE
00061          0020      SPACE  EQU   $20       ASCII SPACE
00062          0024      STACKS EQU   $24       STACK AREA (MAX SIZE)


00064          FCF4      ACIAC  EQU   $FCF4     ACIA CONTROL REGISTER
00065          FCF5      ACIAD  EQU   $FCF5     ACIA DATA REGISTER


00067                    *  CONDITION CODE BITS
00068          0080      E      EQU   $80
00069          0040      F      EQU   $40
00070          0020      H      EQU   $20
00071          0010      I      EQU   $10
00072          0008      N      EQU   $08
00073          0004      Z      EQU   $04
00074          0002      V      EQU   $02
00075          0001      C      EQU   $01
00077                    *  CONDITION CODE BITS (NOT)
00078          00FF      NE     EQU   $7F
00079          00BF      NF     EQU   $BF
00080          00DF      NH     EQU   $DF
00081          00EF      NI     EQU   $EF
00082          00F7      NN     EQU   $F7
00083          00FB      NZ     EQU   $FB
00084          00FD      NV     EQU   $FD
00085          00FE      NC     EQU   $FE
00088 0400               ORG   $0400    POSITION INDEPENDENT
00089 0400 16  003F    6 MTEST  LBRA  M2

00091                    *
00092                    *  TRANSFER VECTORS
00093                    *
00094 0403 16  004D    6 TVM0   LBRA  M0       GENERAL PURPOSE ENTRY
00095 0406 16  028D    6 TVINIA LBRA  INITAC   INIT. ACIA
00096 0409 16  0281    6 TVGCH  LBRA  GCH      GET PRESENT CHAR IN ACCA
00097 040C 16  0300    6 TVINCH LBRA  INCH     A=0 FOR ECHO, BIT7=0 FOR WAIT
00098 040F 16  02F7    6 TVINNP LBRA  INCHNP   CHAR W/O PARITY IN ACCA
00099 0412 16  031D    6 TVINIH LBRA  INIH     CHR IN A, HEX IN B, NEG IF BAD
00100 0415 16  0207    6 TVINAD LBRA  INADDR   GET CHARS UNTIL NON-HEX
00101 0418 16  01AB    6 TVBEGE LBRA  BEGEND   GET ADDRESSES IN 0,X - 3,X

00103 041B 16  0241    6 TVOUT  LBRA  OUT      SEND CHAR FROM ACCA NOW
00104 041E 16  0283    6 TVOUTC LBRA  OUTCH    SEND CHAR WHEN READY
00105 0421 16  029A    6 TVHEXL LBRA  CHEXL    CONVERT ACCA MSN TO HEX (ASCII
00106 0424 16  029B    6 TVHEXR LBRA  CHEXR    CONVERT RIGHT NYBBLE
00107 0427 16  02A5    6 TVOUT2 LBRA  OUT2H    SEND 2 HEX (IX)
00108 042A 16  02A0    6 TVOUT4 LBRA  OUT4H    SEND 4 HEX (IX)
00109 042D 16  02AF    6 TVPDAT LBRA  PDATA    SEND CRLF, DATA ...
00110 0430 16  02AE    6 TVPDA1 LBRA  PDATA1   SEND DATA ...THRU MSB=1
00111 0433 16  02B8    6 TVPCRL LBRA  PCRLF    SEND CRLF NULLS
00112 0436 16  02C9    6 TVREP  LBRA  REPEAT   SEND ACCA, B TIMES
00113 0439 16  02C4    6 TVRSR  LBRA  RSPACE   SEND SPACE, B TIMES

00115 043C 16  0125    6 TVPRIN LBRA  PRINBI   SEND ACCB AS BINARY
00116 043F 16  0116    6 TVRAND LBRA  RAND     PSEUDO-RANDOM ACCA


00118                    *
00119                    *  M2 CONFIGURES FOR EXORCISOR
00120                    *
00121                    *     (ANOTHER SYSTEM MIGHT INITIALIZE
00122                    *     THE STACK, STACK I/O ADDRESSES,
00123                    *     THEN CALL TVM0 IN A SMALL PROM).
00124                    *
00125 0442 32   8C BB   6 M2       LEAS  MTEST,PCR  STACK BELOW PROGRAM
00126 0445 6F   E2       8        CLR   ,-S        INTERNAL I/O MODE
00127 0447 CE   FCF4     3        LDU   #ACIAC
00128 044A 108E FCF5     4        LDY   #ACIAD
00129 044E 8E   F11E     3        LDX   #MAID
00130 0451 34   70       10       PSHS  U,Y,X      ABSOLUTES ON STACK
00131                    *  FALL INTO THE GENERAL-PURPOSE PACKAGE M0
00134 0453 1F   43       6 M0     TFR   S,U        MARK STACK
00135 0455 32   77       5        LEAS  -9,S       AREA FOR TEMP GLOBALS


00137                    *
00138                    *  EQUATES ARE RELATIVE POSITION
00139                    *     FROM USER STACK POINTER.
00140                    *
00141          0006      MODE   EQU   6          I/O SELECT (0 MEANS ACIA ADDR)
00142          0004      CIAC   EQU   4          ACIA CONTROL
00143          0004      INSUB  EQU   4          GE. CHAR IN A
00144          0002      CIAD   EQU   2          ACIA DATA
00145          0002      OUTSUB EQU   2          SEND CHAR FROM A
00146          0000      MAI    EQU   0          MAID RETURN
00147          FFFF      SEED   EQU   -1         STARTING PSEUDO-RANDOM VALUE
00148          FFFE      FLAG   EQU   -2         ERROR HEADING PRINTED? FLAG
00149          FFFD      XCOUNT EQU   -3         NO. OF X'S ON LINE
00150          FFFB      ENDAD  EQU   -5         END ADDRESS
00151          FFF9      BEGAD  EQU   -7         BEGIN ADDRESS
00152          FFF8      NUCH   EQU   -8         NEW CHAR (ESCAPE TEST)
00153          FFF7      OLCH   EQU   -9         OLD CHAR
```

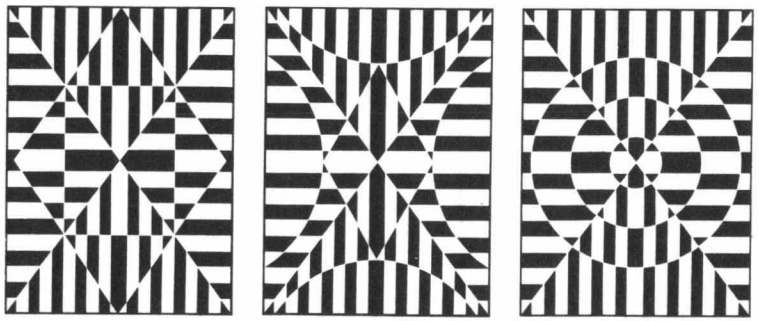
```
00155                          *
00156                          *  VERIFY PROGRAM CORRECTNESS
00157 0457 17   0300    13 M1      LBSR  VERPGM
00158                          *
00159                          *  INITIALIZE ACIA
00160 045A 17   FFA9    13         LBSR  TVINIA
00161                          *
00162                          *  PRINT PROGRAM ID
00163 045D 30   8D 0068  9         LEAX  MSG1,PCR POINT AT MSG1
00164 0461 17   027B    13         LBSR  PDATA    PRINT IT
00165                          *
00166                          *  PRINT PROGRAM LOCATION
00167 0464 30   8C 99     6         LEAX  MTEST,PCR
00168 0467 17   02E7    13         LBSR  PRNTIX
00169 046A 86   2D       2         LDA   #'-
00170 046C 17   0235    13         LBSR  OUTCH
00171 046F 30   8D 0319  9         LEAX  PGMEND,PCR
00172 0473 17   02DB    13         LBSR  PRNTIX
00173                          *
00174                          *  GET ADDRESSES
00175 0476 17   010F    13         LBSR  GETAD
00176                          *
00177                          *  INITIALIZE
00178 0479 86   01       2 TST5    LDA   #1
00179 047B A7   5F       5         STA   SEED,U   SEED VALUE
00180 047D A7   5D       5         STA   XCOUNT,U CRLF ON NEXT X
00181 047F 6F   5E       7         CLR   FLAG,U   NO HEADING YET
00182                          *
00183                          *  STORE PSEUDO-RANDOM SEQUENCE
00184 0481 A6   5F       5 TESTM   LDA   SEED,U   GET SEED
00185 0483 AE   59       6         LDX   BEGAD,U
00186 0485 30   1F       5         LEAX  -1,X     DEX
00187 0487 30   01       5 TST1    LEAX  1,X      NEXT LOCATION
00188 0489 17   00CC    13         LBSR  RAND     DO PSEUDO-RANDOM IN A
00189 048C A7   84       4         STA   0,X
00190 048E AC   5B       7         CMPX  ENDAD,U  ALL DONE ENTRY SWEEP?
00191 0490 2D   F5       3         BLT   TST1     NO, GO AGAIN
00192                          *
00193                          *  CHECK RECOVERED SEQUENCE
00194 0492 A6   5F       5         LDA   SEED,U   GET SEED AGAIN
00195 0494 AE   59       6         LDX   BEGAD,U
00196 0496 30   1F       5         LEAX  -1,X
00197 0498 30   01       5 TST2    LEAX  1,X
00198 049A 17   00BB    13         LBSR  RAND
00199 049D E6   84       4         LDB   0,X      SAVE CHAR FROM MEM
00200 049F 34   04       5         PSHS  B        :  CBA
00201 04A1 A1   E0       6         CMPA  ,S+      :
00202 04A3 27   03       3         BEQ   TST3
00203 04A5 17   005F    13         LBSR  ERR
00204 04A8 AC   5B       7 TST3    CMPX  ENDAD,U  ALL DONE CHECK SWEEP?
00205 04AA 2D   EC       3         BLT   TST2     NO, GO AGAIN
00206                          *
00207                          *  DO PASSES UNTIL END OF SEQUENCE
00208 04AC 17   0048    13         LBSR  PRNTX    SINGLE PASS DONE
00209 04AF 1027 FFA0     7         LBEQ  M0       OUT IFF ESC
00210 04B3 A6   5F       5         LDA   SEED,U   :
00211 04B5 17   00A0    13         LBSR  RAND     :  UPDATE SEED
00212 04B8 A7   5F       5         STA   SEED,U   :
00213 04BA 81   01       2         CMPA  #1       END OF PSEUDO-RANDOM SEQUENCE
00214 04BC 26   C3       3         BNE   TESTM    DO ANOTHER PASS
00215                          *
00216                          *  PRINT DONE, THEN START OVER
00217 04BE 30   8D 002B  9         LEAX  ENDM,PCR POINT AT END MESSAGE
00218 04C2 17   021C    13         LBSR  PDATA1
00219 04C5 30   59       5         LEAX  BEGAD,U
00220 04C7 20   B0       3         BRA   TST5     ANOTHER COMPLETE TEST
00221                          *
00222                          *
00223 04C9      0D0A           MSG1  FDB   CRLF
00224 04CB 50                  PDATA1 FCC  /PSEUDO-RANDOM MEMORY TEST 3.1 AT /
00225 04EC A4                         FCB  $A4      $ W/MSB=1
00226 04ED 41                  ENDM  FCC   /ALL DONE!/
00227 04F6 A0                        FCB   $A0
00230                          *
00231                          *  PRINT AN X FOR EACH PATTERN-TEST
00232                          *
00233                          *  BLOWS A
00234                          *
00235 04F7 6A   5D       7 PRNTX   DEC   XCOUNT,U LINE FULL?
00236 04F9 26   07       3         BNE   PR1      NO, NEED NO CRLF
00237 04FB 86   40       2         LDA   #64      CRLF IMPLIES NEW CHAR CNT
00238 04FD A7   5D       5         STA   XCOUNT,U
00239 04FF 17   01EC    13         LBSR  PCRLF
00240 0502 86   58       2 PR1     LDA   #'X
00241 0504 16   019D      6        LBRA  OUTCH    PRINT X
00242
00243                          *
00244                          *
00245                          *  ERR PRINTS DATA FOUND IN ERROR
00246                          *     IX (2,S) = LOCATION OF ERROR
00247                          *  ACCB (1,S) = VALUE READ FROM MEMORY
00248                          *  ACCA (0,S) = PSEUDO-RANDOM VALUE
00249                          *
00250 0507 34   16       8 ERR     PSHS  X,B,A
00251 0509 86   01       2         LDA   #1       :  CRLF ON NEXT X
00252 050B A7   5D       5         STA   XCOUNT,U :
00253 050D 6D   5E       7         TST   FLAG,U   ERROR HEADING PRINTED?
00254 050F 26   09       3         BNE   E1       YES, DON'T PRINT AGAIN!
00255 0511 6C   5E       7         INC   FLAG,U   REMEMBER, 'IT'S PRINTED!'
00256 0513 30   8D 001A  9         LEAX  HDR,PCR  POINT AT HEADER MSG
00257 0517 17   01C5    13         LBSR  PDATA
00258 051A 17   01D1    13 E1      LBSR  PCRLF
00259 051D 30   62       5         LEAX  2,S      POINT AT SAVED X
00260 051F 17   01AB    13         LBSR  OUT4H    PRINT ADDRESS
00261 0522 C6   03       2         LDB   #3
00262 0524 17   01D9    13         LBSR  KSPACE
00263 0527 E6   61       5         LDB   1,S      MEMORY VALUE
00264 0529 8D   39       7         BSR   PRINBI   PRINT MEMORY VALUE
00265 052B E8   E4       4         EORB  0,S      DESIRED VALUE
00266 052D 8D   35       7         BSR   PRINBI   PRINT ERRORS AS 1'S
00267 052F 35   96      10         PULS  A,B,X,PC GET SAVED REGS, RET
00268                          *
00269                          *
00270 0531      0D0A           HDR   FDB   CRLF
00271 0533 41                        FCC   /ADDRESS    READS        /
00272 054B 42                        FCC   /BIT-IN-ERROR/
00273 0557 A0                        FCB   $A0
00276                          *
00277                          *  RAND GENERATES A 211 BYTE
00278                          *     SEQUENCE IN ACCA
00279                          *
00280 0558 34   02       5 RAND    PSHS  A        SAVE CURRENT #
00281 055A 46            2         RORA
00282 055B 46            2         RORA
00283 055C 46            2         RORA
00284 055D A8   E4       4         EORA  0,S
00285 055F 46            2         RORA
00286 0560 35   02       5         PULS  A        GET SAVED #, CLEAN STACK
00287 0562 46            2         RORA           ROTATED, 1 BIT CHANGED
00288 0563 39            4         RTS
00289
00290                          *
00291                          *  PRINBI OUTPUTS THE VALUE IN B
00292                          *     AS BINARY ASCII, MSB FIRST,
00293                          *     THROUGH ACCA, TO SUBROUTINE OUTCH
00294                          *
00295 0564 34   06       6 PRINBI PSHS  B,A      SAVE STATE
00296 0566 86   08       2         LDA   #$08
00297 0568 34   02       5         PSHS  A        SAVE BIT COUNTER
00298 056A 58            2 T1      LSLB           GET NEXT BIT
00299                          *  BRANCH IF CARRY A ONE
00300 056B 25   04       3         BCS   T2
00301 056D 86   30       2         LDA   #'0
00302 056F 20   02       3         BRA   T3
00303 0571 86   31       2 T2      LDA   #'1
00304 0573 17   012E    13 T3      LBSR  OUTCH    SEND IT
00305 0576 86   20       2         LDA   #SPACE
00306 0578 17   0129    13         LBSR  OUTCH
00307 057B 6A   E4       6         DEC   0,S      COUNT IT
00308                          *  BRANCH IF NOT A WHOLE BYTE DONE
00309 057D 26   EB       3         BNE   T1
00310 057F 35   02       5         PULS  A        CLEAN UP COUNTER
```

```
00311 0581 C6   02       2        LDB   #2       2 SPACES
00312 0583 17   017A    13        LBSR  RSPACE
00313 0586 35   86       8        PULS  A,B,PC   RECOVER STATE, RET
00316                             *
00317                             *  GETAD GETS ADDRESSES INTO 0,X - 3,X.
00318                             *     GOES AGAIN IF TEST WOULD OVERWRITE
00319                             *     MTEST9.
00320                             *
00321 0588 30   59       5 GETAD  LEAX  BEGAD,U
00322 058A 17   0039    13        LBSR  BEGEND
00323 058D 30   8D 01FB  9        LEAX  PGMEND,PCR  END OF MTEST
00324 0591 AC   53       7        CMPX  BEGAD,U
00325 0593 25   18       3        BLO   OK         TESTING AFTER MTEST

00327 0595 30   8D FE67  9        LEAX  MTEST,PCR  START OF MTEST
00328 0599 30   88 DC    5        LEAX  -STACK5,X  ENCLOSE THE STACK
00329 059C AC   59       7        CMPX  BEGAD,U
00330 059E 23   04       3        BLS   NOPE      TESTING INSIDE MTEST!

00332 05A0 AC   5B       7        CMPX  ENDAD,U
00333 05A2 22   09       3        BHI   OK
00334 05A4 30   8D 0006  9 NOPE   LEAX  DANMSG,PCR  DANGER MESSAGE!
00335 05A8 17   0134    13        LBSR  PDATA
00336 05AB 20   DB       3        BRA   GETAD

00338 05AD 39            4 OK     RTS


00340 05AE 44                     DANMSG FCC   /DON'T OVERWRITE MTEST9!/
00341 05C5 A0                            FCB   $A0




00343                             *
00344                             *  BEGEND GETS BEGIN AND END
00345                             *     ADDRESSES FROM KEYBOARD
00346                             *     AND PUTS THEM IN RAM (IX).
00347                             *     BEGIN .LE. END OR TRYS AGAIN.
00348                             *
00349                             *       (X):(X+1) = BEGIN
00350                             *     (X+2):(X+3) = END
00351                             *
00352 05C6 34   06       6 BEGEND PSHS  B,A       SAVE STATE
00353 05C8 8D   1F       7        BSR   INST      GET BEGIN ADDR
00354 05CA 30   02       5        LEAX  2,X
00355 05CC 8D   32       7        BSR   INFIN     GET END ADDR
00356 05CE EC   84       5        LDD   0,X
00357 05D0 A3   83       9        SUBD  ,--X      BEGIN .LE. END?
00358 05D2 25   F2       3        BLO   BEGEND
00359 05D4 35   86       8        PULS  A,B,PC    RECOVER STATE, RET
00361                             *
00362                             *  ASLM4 SHIFTS TWO BYTES (X):(X+1)
00363                             *     LEFT FOUR PLACES
00364                             *
00365 05D6 34   02       5 ASLM4  PSHS  A         SAVE A
00366 05D8 86   04       2        LDA   #4        SHIFT COUNT
00367 05DA 68   01       7 AS1    LSL   1,X       :    16-BIT SHIFT
00368 05DC 69   84       6        ROL   0,X       :
00369 05DE 4A            2        DECA             DONE?
00370 05DF 26   F9       3        BNE   AS1       AGAIN GO IF NO
00371 05E1 35   82       7        PULS  A,PC      RECOVER A, RETURN


00373                             *
00374                             *  INST PRINTS START MESSAGE AND COLLECTS
00375                             *     A HEX ADDRESS (X):(X+1).  ASCII
00376                             *     CR RETURNS, OTHER NON-HEX STARTS OVER.
00377                             *     M TRANSFERS CONTROL TO SUBROUTINE MOVE.
00378                             *
00379                             *     BLOWS A,B
00380                             *
00381 05E3 81   4D       2 INS1   CMPA  #'M
00382 05E5 1027 005D     7        LBEQ  MOVE
00383 05E9 34   10       6 INST   PSHS  X
00384 05EB 30   8D 0022  9        LEAX  STARTM,PCR  ENTER HERE
00385 05EF 17   00ED    13        LBSR  PDATA
00386 05F2 35   10       6        PULS  X
00387 05F4 17   0028    13        LBSR  INADDR
00388 05F7 26   EA       3        BNE   INS1      : AGAIN IFF BAD
00389 05F9 39            4        RTS             :   HEX .NE. CR
00390                             *
00391                             *  INFIN PRINTS END MESSAGE AND
00392                             *     COLLECTS A HEX ADDRESS (X):(X+1).
00393                             *     ASCII CR RETURNS, OTHER NON-HEX STARTS OVER.
00394                             *     M TRANSFERS CONTROL TO SR MOVE.
00395                             *
00396                             *     BLOWS A,B
00397                             *
00398 05FA 81   4D       2 INF1   CMPA  #'M
00399 05FC 1027 0046     7        LBEQ  MOVE

00401 0600 34   10       6 INFIN  PSHS  X
00402 0602 30   8D 0013  9        LEAX  FINM,PCR ENTER HERE
00403 0606 17   00D6    13        LBSR  PDATA
00404 0609 35   10       6        PULS  X
00405 060B 17   0011    13        LBSR  INADDR    : AGAIN IFF BAD
00406 060E 26   EA       3        BNE   INF1      :   HEX .NE. CR
00407 0610 39            4        RTS

00409 0611 20                     STARTM FCC   / BEGIN:/
00410 0618 A0                            FCB   $A0
00411 0619 20                     FINM   FCC   / END:/
00412 061E A0                            FCB   $A0
00415                             *
00416                             *  INADDR INPUTS HEX ADDRESS FROM KEYBOARD
00417                             *     UNTIL NON-HEX.  RETURNS NON-HEX IN
00418                             *     ACCA AND ALSO Z=1 IFF CR.
00419                             *     LAST 4 CHARS ARE COLLECTED
00420                             *     IN BINARY AT 0,X AND 1,X.
00421                             *
00422                             *     BLOWS A,B
00423                             *
00424 061F 34   10       6 INADDR PSHS  X
00425 0621 17   00A9    13        LBSR  OUT4H     PRESENT ADDRESS
00426 0624 C6   02       2        LDB   #2
00427 0626 17   00D7    13        LBSR  RSPACE
00428 0629 35   10       6        PULS  X
00429 062B 17   0104    13        LBSR  IN1H      GET CHAR IN A, HEX IN B
00430 062E 2B   13       3        BMI   INA2      RETURN IFF NOT HEX
00431 0630 6F   84       6        CLR   0,X       INITIALIZE ADDR=0
00432 0632 6F   01       7        CLR   1,X
00433 0634 20   05       3        BRA   INA3      IS HEX, SO ACCUMULATE
00434 0636 17   00F9    13 INA1   LBSR  IN1H      GET CHAR IN A, HEX IN B
00435 0639 2B   08       3        BMI   INA2      RETURN IFF NOT HEX
00436 063B 8D   99       7 INA3   BSR   ASLM4     MAKE A PLACE
00437 063D EA   01       5        ORB   1,X       CATENATE HEX
00438 063F E7   01       5        STB   1,X
00439 0641 20   F3       3        BRA   INA1
00440 0643 81   0D       2 INA2   CMPA  #CR       RETURN Z=1 IFF CR
00441 0645 39            4        RTS


00443                             *
00444                             *  MOVE RE-POSITIONS MTEST9 TO BEGIN ADDRESS!
00445                             *
00446 0646 30   8D FDB6  9 MOVE   LEAX  MTEST,PCR  START OF MTEST
00447 064A 34   40       6        PSHS  U
00448 064C EE   59       6        LDU   BEGAD,U   MOVE-TO LOCATION
00449 064E 108E 038D     4        LDY   #PGMEND+1-MTEST  LENGTH OF MTEST
00450 0652 A6   80       6 MO1    LDA   ,X+       GET A BYTE
00451 0654 A7   C0       6        STA   ,U+       MOVE IT
00452 0656 31   3F       5        LEAY  -1,Y      COUNT IT
00453 0658 26   F8       3        BNE   MO1       BRANCH IF NOT DONE
00454 065A 35   40       6        PULS  U
00455 065C 6E   D8 F9    6        JMP   [BEGAD,U]  RE-INITIALIZE EVERYTHING


00457                             *
00458                             *  OUT SENDS CHAR NOW
00459                             *
00460 065F A7   D8 02    7 OUT    STA   [CIAD,U]
00461 0662 39            4        RTS
00464                             *
00465                             *  CKESC CHECKS FOR ESCAPE OR CONTROL X
00466                             *     ESC RETURNS Z=1 (RESTART MEMTEST)
00467                             *     CTLX = RETURN (JUMP TO MAID)
00468                             *
```

```
00469 0663 34   04       5 CKESC  PSHS  B
00470 0665 E6   58       5        LDB   NUCH,U    SET UP...
00471 0667 E7   57       5        STB   OLCH,U    SOFTWARE EDGE-DETECTOR
00472 0669 1E   89       7        EXG   A,B
00473 066B 86   80       2        LDA   #$80      GET CHAR NOW
00474 066D 17   009F    13        LBSR  INCH      (NO ECHO)
00475 0670 1E   89       7        EXG   A,B
00476 0672 C4   7F       2        ANDB  #$7F
00477 0674 E7   58       5        STB   NUCH,U
00478 0676 C1   18       2        CMPB  #CTLX
00479 0678 26   03       3        BNE   CK1       ABORT MTEST9 PACKAGE?
00480 067A 32   C4       4        LEAS  0,U       FUNNY TFR U,S
00481 067C 39            4        RTS             RETURN TO CALLING SYSTEM

00483 067D C1   1B       2 CK1    CMPB  #ESC      IS THE NEW CHAR ESC?
00484 067F 26   0A       3        BNE   CK2
00485                      *  HERE ACCB = NUCH = ESC
00486 0681 E1   57       5        CMPB  OLCH,U    THE OLD CHAR ALSO ESC?
00487 0683 27   06       3        BEQ   CK2       RESTART IFF FIRST ESC CHAR
00488 0685 1F   34       6        TFR   U,S       ABSOLUTES REMAIN
00489 0687 6E   8D FDC8  8        JMP   M0,PCR    FUNNY LBRA M0

00491 068B 35   84       7 CK2    PULS  B,PC      RETURN NO ACTION


00493                      *
00494                      *  GCH GETS PRESENT CHAR INTO ACCA
00495                      *
00496 068D A6   D8 02    7 GCH    LDA   [CIAD,U] GET THE CHAR NOW!
00497 0690 A1   D8 02    7        CMPA  [CIAD,U] STILL THE SAME?
00498 0693 26   F8       3        BNE   GCH       IF NOT, GO AGAIN
00499 0695 39            4        RTS


00501                      *
00502                      *  INITIALIZE ACIA
00503                      *
00504 0696 34   02       5 INITAC PSHS  A
00505 0698 86   03       2        LDA   #3       RESET ACIA
00506 069A A7   D8 04    7        STA   [CIAC,U]
00507                      *
00508                      *     X X X  X X X  0 1   DIVIDE BY 16
00509                      *     X X X  1 0 1  X X   8 DATA + 1 STOP
00510                      *     0 0 0  X X X  X X   READER OFF, BOTH
00511                      *                           INTERRUPTS DISABLED
00512 069D 86   15       2        LDA   #%00010101
00513 069F A7   D8 04    7        STA   [CIAC,U]
00514 06A2 35   82       7        PULS  A,PC
00517                      *
00518                      *  OUTCH WAITS TILL ACIA IS READY
00519                      *     THEN SENDS A TO ACIA
00520                      *
00521                      *     CHANGES A
00522                      *
00523 06A4 8A   80       2 OUTCH  ORA   #$80      DONT SEND MSB
00524 06A6 6D   46       7        TST   MODE,U
00525 06A8 27   03       3        BEQ   OUTC2
00526 06AA 6E   D8 02    6        JMP   [OUTSUB,U]  ALTERNATE I/O
00527 06AD 34   04       5 OUTC2  PSHS  B         SAVE B IN STACK
00528 06AF E6   D8 04    7 OUTC1  LDB   [CIAC,U] ACIA CONTROL
00529 06B2 C4   02       2        ANDB  #2        CHECK XMIT STATUS
00530 06B4 27   F9       3        BEQ   OUTC1     LOOP IF XMIT NOT READY
00531 06B6 A7   D8 02    7        STA   [CIAD,U] ACIA DATA
00532 06B9 17   FFA7    13        LBSR  CKESC
00533 06BC 35   84       7        PULS  B,PC      RECOVER B, RETURN


00535                      *
00536                      *  CHEXL MAKES LEFT NYBBLE ACCA
00537                      *     ASCII HEX
00538                      *  CHEXR MAKES RIGHT NYBBLE ACCA
00539                      *     ASCII HEX
00540                      *
00541                      *     BLOWS A
00542                      *
00543 06BE 44            2 CHEXL  LSRA            LEFT NYBBLE BECOMES RIGHT
00544 06BF 44            2        LSRA
00545 06C0 44            2        LSRA
00546 06C1 44            2        LSRA

00547 06C2 84   0F       2 CHEXR  ANDA  #$F       RIGHT NYBBLE ONLY
00548 06C4 8B   30       2        ADDA  #$30      OFFSET TO ASCII 0
00549 06C6 81   39       2        CMPA  #$'9      LARGER THAN ASCII 9?
00550 06C8 23   02       3        BLS   CHEX1
00551 06CA 8B   07       2        ADDA  #7        ADDITIONAL OFFSET TO ASCII A-F
00552 06CC 39            4 CHEX1  RTS


00554                      *
00555                      *  OUT4H DOES OUT2H TWICE
00556                      *  OUT2H SENDS (IX) AS
00557                      *     2 ASCII HEX CHARS.
00558                      *
00559                      *     BLOWS A, MOVES X
00560                      *
00561 06CD 8D   00       7 OUT4H  BSR   OUT2H     2H X 2 = 4H
00562 06CF A6   80       6 OUT2H  LDA   ,X+
00563 06D1 34   02       5        PSHS  A         SAVE A
00564 06D3 8D   E9       7        BSR   CHEXL     GET MS BYTE
00565 06D5 17   FFCC    13        LBSR  OUTCH     SEND IT
00566 06D8 35   02       5        PULS  A
00567 06DA 8D   E6       7        BSR   CHEXR     GET LS BYTE
00568 06DC 16   FFC5     6        LBRA  OUTCH     SEND IT
00571                      *
00572                      *  PDATA PRINTS CRLF, TEXT STRING
00573                      *     (IX) IS START
00574                      *     B7 = 1 IS LAST CHAR PRINTED
00575 06DF 8D   0D       7 PDATA  BSR   PCRLF
00576                      *  FALL INTO PDATA1


00578                      *
00579                      *  PDATA1 PRINTS TEXT STRING
00580 06E1 34   12       7 PDATA1 PSHS  A,X       SAVE STATE
00581 06E3 A6   84       4 PD1    LDA   ,X        GET A CHAR
00582 06E5 17   FFBC    13        LBSR  OUTCH     SEND IT
00583 06E8 6D   80       8        TST   ,X+       TEST MSB
00584 06EA 2A   F7       3        BPL   PD1       ANOTHER CHAR IF B7=0
00585 06EC 35   92       9        PULS  A,X,PC    RECOVER STATE, RETURN


00587                      *
00588                      *  PRINT CRLF
00589 06EE 34   10       6 PCRLF  PSHS  X         SAVE PRESENT X
00590 06F0 30   8D 0004  9        LEAX  TCRLF,PCR  POINT AT CRLF TEXT
00591 06F4 8D   EB       7        BSR   PDATA1    PRINT IT
00592 06F6 35   90       8        PULS  X,PC      RECOVER STATE, RETURN

00594 06F8      0D0A       TCRLF  FDB   CRLF
00595 06FA 00                     FCB   0,,,$80


00597                      *
00598                      *  PSPACE PRINTS ONE SPACE
00599                      *  RSPACE PRINTS B SPACES
00600                      *  REPEAT PRINTS ACCA, B TIMES
00601                      *
00602                      *     THESE ALL BLOW A,B
00603                      *
00604 06FE C6   01       2 PSPACE LDB   #1        SET COUNT TO 1
00605 0700 86   20       2 RSPACE LDA   #SPACE    LOAD A WITH ASCII SPACE
00606 0702 17   FF9F    13 REPEAT LBSR  OUTCH     PRINT ACCA
00607 0705 5A            2        DECB            DONE?
00608 0706 26   FA       3        BNE   REPEAT    LOOP TILL DONE
00609 0708 39            4        RTS             RETURN
00610                      *
00611                      *  INCHNP GETS A CHAR (NO PARITY)
00612                      *
00613 0709 4F            2 INCHNP CLRA            SET UP ECHO, WAIT FOR CHAR
00614 070A 8D   03       7        BSR   INCH      GET BYTE FROM ACIA
00615 070C 84   7F       2        ANDA  #$7F      CLEAR BIT7
00616 070E 39            4        RTS
00619                      *
00620                      *  INCH RETURNS CHAR IN ACCA.
00621                      *     ECHOS IFF OLD ACCA=0.
00622                      *     WAITS FOR CHAR IFF OLD ACCA B7=0
00623                      *
00624 070F 6D   46       7 INCH   TST   MODE,U
00625 0711 27   03       3        BEQ   INCH3
```

```
00626 0713 6E    D8 04     6          JMP    [INSUB,U]
00627 0716 4D               2 INCH3    TSTA
00628 0717 2A    03         3          BPL    INCH4
00629 0719 16    FF71       6          LBRA   GCH
00630 071C 34    02         5 INCH4    PSHA
00631 071E A6    D8 04      7 INCH1    LDA    [CIAC,U] ACIA STATUS
00632                         *  IFF DATA READY, B0=1
00633 0721 44               2          LSRA                B0 INTO CARRY
00634 0722 24    FA         3          BCC    INCH1
00635 0724 A6    E0         6          LDA    ,S+          SNEAKY PULL
00636 0726 26    06         3          BNE    INCH2        SHALL WE ECHO?
00637 0728 A6    D8 02      7          LDA    [CIAD,U]
00638 072B 16    FF76       6          LBRA   OUTCH

00640 072E A6    D8 02      7 INCH2    LDA    [CIAD,U] DATA INTO ACCA
00641 0731 39               4          RTS


00643                         *
00644                         *  IN1H WAITS FOR NEW CHAR FROM ACIA IN ACCA,
00645                         *     THEN TRANSLATES CHAR TO HEX IN ACCB.
00646                         *     IN1H RETURNS NEG IFF NOT HEX.
00647                         *
00648 0732 8D    D5         7 IN1H     BSR    INCHNP       WAIT FOR CHAR AND ECHO
00649 0734 1F    89         6          TFR    A,B
00650 0736 17    FF2A      13          LBSR   CKESC
00651                         *
00652                         *  CHECK AND CONVERT FOR VALID HEX CHAR
00653 0739 C1    30         2 CMPINP   CMPB   #'0
00654 073B 25    11         3          BLO    INBAD        BAD IF UNDER ASCII 0
00655 073D C1    39         2          CMPB   #'9
00656 073F 23    0A         3          BLS    INGD         GOOD IF 0-9
00657 0741 C1    41         2          CMPB   #'A
00658 0743 25    09         3          BLO    INBAD        BAD IF BETWEEN 9,A
00659 0745 C1    46         2          CMPB   #'F
00660 0747 22    05         3          BHI    INBAD        BAD IF OVER F
00661 0749 C0    07         2          SUBB   #7           LETTERS TO BINARY
00662 074B C4    0F         2 INGD     ANDB   #$F          RETURN POS IFF GOOD
00663 074D 39               4          RTS
00664 074E 1A    08         2 INBAD    ORCC   #N           RETURN NEG IFF BAD
00665 0750 39               4          RTS
00668                         *
00669                         *  PRINTIX PRINTS THE VALUE IN X
00670                         *     AS 4 HEX DIGITS
00671                         *
00672 0751 34    10         6 PRNTIX   PSHS   X            SAVE X
00673 0753 1F    41         6          TFR    S,X          POINT AT SAVED X
00674 0755 17    FF75      13          LBSR   OUT4H        PRINT IT
00675 0758 35    90         8          PULS   X,PC         RECOVER X, RETURN.
00677                         *
00678                         *  VERPGM VERIFYS PROGRAM CORRECTNESS BY
00679                         *     COMPUTING PARITY OVER ENTIRE PGM
00680                         *     (PGM HAS BEEN MADE ODD PARITY)
00681                         *
00682 075A 30    8D 002E    9 VERPGM   LEAX   PGMEND,PCR  LAST ADDRESS
00683 075E 34    10         6          PSHS   X            (PARITY BYTE)
00684 0760 30    8D FC9C    9          LEAX   MTEST,PCR
00685 0764 4F               2          CLRA
00686 0765 A8    80         6 VER1     EORA   ,X+
00687 0767 AC    E4         6          CMPX   0,S          DONE?
00688 0769 23    FA         3          BLS    VER1
00689 076B 32    62         5          LEAS   2,S          CLEAN UP STACK
00690 076D 4C               2          INCA                ODD PARITY NOW 0'S
00691 076E 27    09         3          BEQ    VER2         NORMAL RETURN
00692 0770 30    8D 0006    9          LEAX   VERMSG,PCR
00693 0774 17    FF68      13          LBSR   PDATA
00694 0777 1F    34         6          TFR    U,S          RETURN TO MAIN SYSTEM
00695 0779 39               4 VER2     RTS

00697 077A 49                 VERMSG   FCC    /INVALID PGM LOAD!/
00698 078B A0                          FCB    $A0


00700 078C 93                 PGMEND   FCB    $93          ODD PARITY BYTE
00702              0000                END

TOTAL ERRORS   00000
TOTAL WARNINGS 00000
```

```
TEXBUG $30.
TEXBUG MAID
>E800;G
MDOS 02.01
=LOAD MTEST9;MV

>400;G

PSEUDO-RANDOM MEMORY TEST 3.1 AT $0400-078C
 BEGIN: 0D8D  400 (CR)
 END: 07B6  2000 (CR)
DON'T OVERWRITE MTEST9!
 BEGIN: 0400  2000 (CR)
 END: 2000 (CR)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXALL DONE!
XXXXXXXXXXXXXXXXXX (ESC)

PSEUDO-RANDOM MEMORY TEST 3.1 AT $0400-078C
 BEGIN: 2000  M

PSEUDO-RANDOM MEMORY TEST 3.1 AT $2000-238C
 BEGIN: 0000  AFFEBFFE (CR)
 END: 0000
 END: 0000  J
 END: 0000  P
 END: 0000  C002 (CR)

ADDRESS     READS                 BIT-IN-ERROR
C000    0 0 0 0 0 0 0 0    0 0 1 0 0 0 0 0
C001    0 0 0 0 0 0 0 1    0 0 0 1 0 0 0 1
C002    0 0 0 0 0 0 1 0    0 0 0 0 1 0 1 0
X
C000    0 0 0 0 0 0 0 0    0 0 0 1 0 0 0 0
C001    0 0 0 0 0 0 0 1    0 0 0 0 1 0 0 1
C002    0 0 0 0 0 0 1 0    1 0 0 0 0 1 1 0
X
C000    0 0 0 0 0 0 0 0    0 0 0 0 1 0 0 0
C001    0 0 0 0 0 0 0 0 (control X)
>78C/93 00
>400;G
INVALID PGM LOAD!
>78C/00 93
>400;G

PSEUDO-RANDOM MEMORY TEST 3.1 AT $0400-078C
 BEGIN: 2000
 END: 2000  400
 BEGIN: 2000
 END: 0400  2000
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
>
```



EDITHA

A REVISION OF THE FYLSTRA KIM-1 EDITOR PROGRAM

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

Dear Jim, Tom, et al., *Received: 78 April 3*

I am herewith submitting an extensive recoding and enhancement of the editor section of Dan Fylstra's SWEETS program for the KIM-1, published in (and copyrighted by) BYTE. It poses several dilemmas of general interest. First, the problem of program names. SWEETS is Fylstra's acronym for 2 main programs, that share some subroutines. The entry label for the editor (the only one I have worked on) is CMD. I have given the more descriptive name EDITHA to my revision, on the grounds that its coding is mostly quite different and that it includes some major innovations (address information and modification capability, greater error-protection, and a richer command structure with modes not found in CMD). It is clearly an enhancement of the kind that Fylstra himself anticipated would grow out of CMD. I hesitated before adopting the name EDITHA, since renaming can easily become an idiotic game, involving personal vanity, causing confusion, and even trying to disguise software theft. I have restained many of Fylstra's names in EDITHA whenever my recoding did not involve a major functional alteration, both as a courtesy to the originator of the basic concept and as an aid to comprehension by users. One of his subroutines (DETLN1) is an inefficient way of determining the bytecount of a 6502 opcode. In EDITHA I have replaced DTELN1 logic by the decoding logic of my subroutine BYTCNT (published in DDJ *# 22), with minor retailoring to fit the special context. I re-use the name BYTCNT for this trivial modification, because I think it is better to extend the meaning of a subroutine name to include minor variations, instead of endlessly coining new names. There are, alas, no rules defining how great a dissimilarity is needed to justify a new name.*

The second dilemma is the perennial one of software ownership. I assert copyright (qualified by a special non-monopolistic free-diffusion clause) for all components of EDITHA not either previously in the public domain or copyrighted by someone else. Exactly what these are is a moot question. Very little of the original CMD coding is left intact in EDITHA. None of the alterations is anything but functional (i.e., there is no intent to evade existing copyright by mere recoding). None is copied from any work I have had access to (although since there are relatively few ways of doing something efficiently, it is not impossible that some coding is an independent recreation of already existing logic). Here is the qualifying clause: The intent of my copyright is to remove all impediments to free diffusion, including (to whatever extent my copyright gives me leverage to do so) impediments that others may wish to impose on enhanced or different implementations, including systems other than the KIM-1. I therfore authorize any and all uses (including copying in any form, publication in whole or in part, and commercialization) if (and only if) all software associated with the EDITHA coding is declared to be equally available for totally unrestricted use. What I am trying to ensure is not only that my copyright will be no bar to progress but that it shall serve as an aid to it. Perhaps this is Utopian, but there's no harm in trying! It would be a tragicomedy if micro software had to be diffused underground, in the style of Russian samizdats, *as a response to the censorship aspects of copyrighting.*

The third dilemma is: where should revision of a program published in BYTE *be published? First, let me say that I consider* BYTE *a valuable journal. I have had a note (on 6502 opcodes) published in it, and a second one (on a pseudorandom number generator, 8080-coded) is accepted and will (eventually!) get published. There are some advantages (a small remuneration, glossy paper, and a very large readership) and some disadvantages (among them are: loss of all rights to your work, and often long publication delay, and no indication of the date of receipt of articles). On the whole, my relationship with* DDJ *has been a more humanly satisfying one. Friends have told me its title is zany (but to me it's whimsical – I reserve zany [and its synonyms] for a world with tens of thousands of megatons of hydrogen bombs poised for launching). But publication is prompt, always includes date of receipt (a fine ethical touch), and the author retains the rights to his work. These small-is-beautiful qualities compensate for inexpensive paper and no remuneration!*

Sincerely,
H. T. Gordon

The SWEETS machine-language editor program CMD, created by Dan Fylstra (Feb. 1978 *BYTE*) is a major enhancement of the KIM-1 SBC. However, it has some coding inefficiencies, logic errors, and other limitations. The major ones are its inability to allow manipulation of addresses (only program instructions can be seen in the KIM display, and the lack of adequate safeguards against user errors. It is all too easy to ignore the CMD error-warning and cause disastrous upmoves or downmoves of immense blocks of RAM, including the CMD program itself. My revision gradually became a major overhaul, involving drastic alteration of both the main program (whose entry point I have renamed EDITHA) and the subroutines.

Insofar as possible, EDITHA (version 1.0, intended for the basic KIM-1) is located in the same memory areas used by CMD. This conforms with Fylstra's intent of leaving a large block of RAM free for new program construction under editor control. However, the enhancements provided by EDITHA increase its length to 478 bytes (71 more than CMD), and its subroutine READIN had to be located in 0200-0234. This leaves only the 459 locations from 0235-03FF free.

The revision relocates the original CMD pointers in zero-page: MOVAD (to DE-DF), BEGAD (to E0-E1), CURAD (to E2-E3), ENDAD (to E4-E5), and adds a new one, LIMAD (at E6-E7). This allows operation of a new guard subroutine, INLIMT, that protects memory locations outside preset program limits from being moved or overwritten. BEGAD is redefined as one location *below* the program start (lowest) address. LIMAD must be set 2 locations below the highest address that the program can enter. BEGAD and all lower memory locations, and LIMAD+3 and all higher memory locations, cannot be altered or moved by EDITHA. The guard subroutine blocks the action of the DA (delete instruction) key unless BEGAD < CURAD < ENDAD. In addition, it blocks the AD (add instruction) key unless ENDAD < LIMAD. When these conditions are not met, the only effect of pressing either key is to display CURAD in the usual KIM address display, and EE in the data display. CURAD is the address of the instruction currently displayed, that EDITHA is refusing to add to or delete. The same message will be displayed if any hex key is pressed when EDITHA is in the command-entry mode, but in this case the EE does not signify that CURAD is out-of-bounds but that hex keys cannot be interpreted and are being rejected.

Although EDITHA has instructions that allow the user to alter BEGAD or LIMAD, they are normally fixed and define the program limits. As in CMD, CURAD is advanced to the next higher instruction when the + key is pressed. However, while the + key is depressed, EDITHA displays the address of the next instruction (that appears when the key is released). If this address is within limits, 03 appears in the data display, but if it is out-of-bounds, EE is displayed.

During program writing, CURAD does not change, but ENDAD moves upward when AD is used to insert an instruction and downward when DA is used to delete the instruction currently in CURAD. If deletions exceed insertions, ENDAD may move below CURAD, and thereby block the action of the AD and DA keys. This is no problem, since the user can press a hex key to see where CURAD is and then use the pointer-resetting instruction to set ENDAD to a higher location. In fact, it is sometimes useful to set ENDAD to the program start address, so that the user can explore anywhere in memory with no risk of altering programs by accidental pressing of the AD or DA keys.

The operational changes in the + and DA key actions are achieved by quite trivial modification of the original (much less informative) CMD error-message, together with continual supervision by the guard subroutine. Users may notice one more trivial change: one-byte displays by EDITHA are of normal brightness, while those by CMD are extremely bright. CMD refreshes one-byte displays 3 times as often as 3-byte ones; this may or may not shorten the life of the LEDs, but I found it unpleasant. Therefore EDITHA inserts a very brief dark period in every refresh of a one-byte display. Both CMD and EDITHA display two-bytes at about 1.5 times normal brightness.

Implementation of Complex Commands in the AD and GO Operations.

Unlike the + and DA operations, which are simple commands executed by pressing one command key, the AD and GO operations require complex commands. Both call subroutine READIN, since the user is expected to key in more information. Although much more complex than its equivalent in CMD (subroutine RDBYTE), READIN also calls subroutine SCAN1 twice. Unlike RDBYTE (that always error-exits if a command key is pressed), the first call to SCAN1 by READIN will accept either a hex or a command key, the latter causing a return for interpretation by the calling routine. If the first key is hex, the second call to SCAN1 *ignores* all command keys but returns when a hex key is pressed, inserting the completed byte in the leftmost display (all others blank). ADKEY logic interprets this as an opcode; if it is a one-byte opcode, it is inserted in the program and the operation is over. If it is a two- or three-byte opcode, READIN is again called so the user can complete the instruction. At this point, it is possible to press a command key and cause a return. In version 1.0, EDITHA interprets such command-hex-command instructions as errors, simply wiping them out as if they had never been keyed-in at all. In an expanded version, however, more complex interpreting logic could open a cornucopia (or a Pandora's box?) of special commands. Very few of these possibilities are realized in version 1.0. One reason is the memory limitations of the basic KIM-1. Another is that I am not sure how powerful a machine-language editor ought to become, or what operations will prove to be the most useful.

The AD Key Operations

Pressing the AD key has no detectable effect if CURAD is within limits and the instruction currently in the display is one-byte. If it is two- or three-byte, it is left-shifted so that only the previously rightmost byte appears in the leftmost display (all others blank). When the opcode is keyed-in, it replaces the displayed byte. If it is a one-byte opcode the instruction is complete and written into the program (and being displayed by the program-scanning subroutines). Pressing a hex key will cause the usual error-display of CURAD. If the opcode is incorrect, it can be deleted by pressing the DA key.

If the opcode requires 1 or 2 operand bytes, pressing any command key will cause an error wipe-out (a quick way to erase a wrong opcode). If a hex key is pressed, nothing will happen until a second hex key is pressed. If the opcode was two-byte, the second keyed-in byte will appear in the *middle* display (*normal* order, an indication that the whole instruction has been written into the program). It can be deleted by the DA key. However, keying in the first operand of a three-byte opcode will cause the opcode to shift to the middle display and the operand to appear in the leftmost one (*reverse* order, instruction not yet completed). Again, pressing a command key will cause a wipeout. Keying in the third byte (second operand) will cause display in normal order (instruction written-in, deletable by DA).

A peculiar "double-command" operation is also possible, executed if the AD key is pressed *twice* (pressing AD, then any other command key, has the same effect but the AD-AD is safer). The effect is to insert the single-byte in the display (following the first AD) in the program, "completing" the instruction. The AD-AD operation is useful *only* if the instruction in the display before the first AD was a one-byte opcode. It allows *duplication* of this opcode, a convenient way to enter a string of one-byte opcodes (00, 0A, 4A, EA, etc.). The AD-AD was not a planned feature of EDITHA, but an accidental consequence of the planned GO-AD (to be discussed in the next section), retained because its implementation requires very little interpreting logic.

The GO Key Operations

These represent by far the most complex EDITHA enhancements, that still are only a minute fraction of a legion of possibilities. The only "double-command" is the GO-AD (the other 3 possible ones are interpreted as errors). This sequence of 2 command keyings resets CURAD to the program start (one location higher than BEGAD). (Note that, although the command keys are read by the KIM monitor subroutine

GETKEY as hex numbers from $10 to $13, the READIN decoder subtracts 4 from these values and returns command codes from $0C to $0F to the GOKEY routine.)

All other GO operations require keying-in 4 hex bytes for completion, the first 3 appearing in the display one after another, in reverse order. Wipeout can be caused by pressing any command key following entry of any byte into the display. The terminator byte (not displayed) is the command code that will cause execution of a special operation. Since all non-interpreted command codes cause an error-message, it is useful to keep the final key of this code depressed so that it can be seen displayed. If there is no error-message, the command has been executed. Error-messages are always XX 04 EE, where XX is the command code that could not be executed (except for codes 04 and 08, where XX is 00). The following is a list of the 5 valid command modes, with their codes:

Mode I, code 00
This is the ERASE command, that restores the original status before the GO key was pressed. Although the "wipeout" feature of command keys (a later development) makes it unnecessary, it might be useful in later versions.

Mode II, codes 01 to 03
These are IDPR (insert data into program) commands, 01 inserting only the first keyed-in byte (rightmost in the display), 02 the first and second, 03 all three. Before completion, display is in *reverse* order. When the command is completed, the inserted bytes will be displayed in *normal* order (as with the AD operation) but interpreted as data. Thus the sequence of keyings: GO-08-18-28-03, that is displayed as 28 18 08 before the command code is entered, will be displayed as 08 18 28 afterward. This sequence would need 3 separate entries in the AD operation, since all the entries are one-byte opcodes. Also, the *entire* sequence can be deleted by pressing the DA key once (if there has been an error).

In both CMD and EDITHA, to add a new program or data sequence *following* the one in the display, one must first press the + key to advance CURAD. This is necessary to allow the user to see in the display exactly what he has entered. In the EDITHA IDPR Mode II, all bytes that will be entered (and with codes 02 and 01, some that will *not* be) are visible in the display before the command is keyed-in, although in reverse order. Some users may wish to implement an automatic advance, so that more data can be entered by the Mode II GO without having to press the + key. This can be done by replacing the BEQ DACMD at 017B by a BEQ STPKEY (F0 37). The disadvantage is that one will not see displayed what has been entered, and not have a second chance to delete an erroneous entry.

Mode III, codes 05 to 07 and 09 to 0B
These are program SEARCH commands, with some kinship to the GO operation in CMD. There is unnecessary redundancy here (05 = 09, 06 = 0A, 07 = 0B) that saves a little GOKEY interpreting logic, but could be eliminated in expanded versions if codes 09 to 0B can be given another useful interpretation. One major difference from the search GO in CMD is that searching is done in the range from the current CURAD (the address of the instruction that was in the display before the GO) upward in memory to ENDAD. To search the *entire* program, one must first reset CURAD to the program start by a GO-AD. The reason for this modification is that the same instruction may recur several times in a program, and the GO logic in CMD can only find the *first* occurrence. When using the EDITHA search GO, the user should keep the final command key depressed. If a match is found, the display will show the address of this instruction that will be displayed when the key is released (while the data display will show the number of bytes that were searched for). If this is not the one wanted, press the + key and redo the search GO to find the *next* identical instruction, higher in the program. If there is no match, the operation will display the ENDAD address,with EE in the data display.

The second major difference from CMD is that EDITHA can search for only the opcode (code 05), for the opcode and one following byte (code 06, identical with the CMD search GO), or for the opcode and *two* following bytes (code 07). E.g., the key sequence GO-85-FB-60-07 requires that an 85 FB 60 sequence exist in the program. This happens to be the terminal program sequence in the KIM monitor ROM. An interesting test of searching speed is possible if one sets BEGAD to 03FF, CURAD to 1FD4, and ENDAD and LIMAD both to 1FD5. When EDITHA is entered (set address to 019A and press GO) it will display the RTS ($60 at 1FD4. A GO-AD sends it to 0400, and 0404 is displayed (although of course there is nothing there in the basic KIM). The GO-85-FB-60-70 keying sequence will darken the display for less than a second, then (with the final key still depressed) display 1F D2 03. This is the addresss of the last 85 FB instruction in the ROM, that appears when the 7 key is released. Nearly 7K of (mostly non-existent!) memory has been subjected to a complex search in a reasonably short time. If one uses the 06 command code to search for only 85 FB, one can find 5 recurrences of it lower in the KIM ROM than the terminal one.

Mode IV, code OC
This works exactly like the GO-AD double-command. It was the awkwardness of resetting CURAD to the program start in this way (or in another way possible with Mode V) that led me to develop the GO-AD. In the original CMD design (largely retained in this pioneer version of EDITHA), keyings are recognized in 2 different ways. CMD calls the program-scanning routines that maintain display of the instruction at CURAD and detect a depressed key, which causes a return for interpretation by CMD logic. Hex keys are rejected (an EE EE EE error-message being displayed while they are depressed), while each of the 4 command keys causes control to shift to a special routine. With more elaborate logic, one could interpret double-commands here (or even, if one were ambitious, triple-commands). It would be a cleaner way than that in version 1.0, which re-uses the scaffolding of a primitive approach to implement a more sophisticated one (just to avoid one more reconstruction job).

The second kind of key-recognition in CMD is done by the GOKEY and ADKEY routines, via subroutine RDBYTE, which error-exits if a command key is pressed but accepts keyed-in hex bytes. My decision to implement double commands here was based on the fact that I had already included a lot of command-code interpreting logic and it was very simple just to modify READIN slightly. When I started out to revise CMD, I did not have a clear idea of how it worked and never dreamed that it would go so far! The structural flaws in EDITHA (1.0) are the result of its just growing like Topsy, with its growth halted when it started getting too big for the basic KIM-1.

Mode V, even-numbered codes from $10 to $8E
These are IDZP (insert data into zero-page) commands (64 in all), that write the first 2 keyed-in data bytes into contiguous zero-page locations specified by the command code. E.g., the keying sequence GO-12-34-10-10 will cause $34 to be written in location 0010 and $12 in location 0011. Before the keying-in of the command code 10, the first 3 bytes are displayed as 10 34 12 (reverse order); this is wiped out by the command code, since nothing is entered into the program. Since the effect is not visible and one has to be extra careful with zero-page, the third keyed-in (and so displayed) byte

must be *identical* with the final command code that causes execution. If it is not, an error message is displayed while the final command key is depressed to indicate a total wipeout.

Because important subroutines are located in zero page (0080 to 00D4), command codes 80 to 8E skip these locations and write into locations 00D8 to 00E7. E.g., the sequence GO-1C-00-8A-8A resets CURAD to address 1C00 in the KIM ROM, causing display of the instruction there. This program can then be scanned by the + key, since EDITHA does not object to its user going anywhere in memory just to *look* at it. Before wandering around in memory, ENDAD should be reset to the program-start address (easily checked by a GO-AD), using the 8C (reset ENDAD) command code, to "disconnect" the AD and DA keys. Codes 8E (reset LIMAD), 88 (reset BEGAD), and 86 (reset MOVAD) will be less often needed, and the 3 lower pointers (80, 82, 84) are available for future enhancements.

Some General Commands on the Listing

Since EDITHA interacts with molasses-slow human operators, it is not designed for speed but emphasizes coding efficiency, error-protection, and convenience. One major subroutine (ASCNIT, 0100 to 011E) is *not* listed because its coding is basically identical with that of the Fylstra-BYTE copyrighted CMD (located in the original at 0103 to 0121) except that the downward relocation in EDITHA requires that the 3JSRs to SCAN3 be coded as 20 1F 01 instead of the original 20 22 01. The relocation is functional, since the JSR DETLEN that is the first instruction of the CMD subroutine SCAN is now the second instruction in EDITHA (at 019B), so that the third one (JSR SCANIT, labeled DACMD) can be branched to to cause a display *not* interpreted as an instruction. My omission of SCANIT may strike some readers as ridiculous, but the intent is to avert the wrath of whatever gods EDITHA may offend! It also underscores the reasons for the free-diffusion conditions (spelled out in my covering letter) attached to my own copyright.

Listings therefore start at 011F (SCAN3), that is significantly modified in order to reduce the brightness of single-byte displays (the JSR ONE just kills a little time). To be punctilious, I note that the terminal coding of the EDITHA subroutine MVDOWN (00C9-00D4) is identical with that of the CMD subroutine ADVAND. It seems to me that the logic of adding an 8-bit number to a 16-bit number is so commonly used that it is in the public domain.

Since keying in 478 program bytes into a KIM is a headache, I am willing to tape EDITHA for KIM users who would like to try it out if (1) they mail a blank cassette to me, with a stamped self-addressed envelope that I can return it in, and (2) they agree to inform me of any flaws they may discover in operation (*after* checking the RAM to make sure the listing is *correct*). Regrettably, I can not be responsible for any failures of the U.S. Postal Service, and this approach will not be usable by users outside the USA. The taping will be in the standard KIM format (slow but sure) in 2 sections: 0080-0235 (index 01) and 1780-17E3 (index 02). I am not eager to do this hundreds of times, but fortunately this is an unlikely event!

Comment on Enhancements

As I have indicated, EDITHA has many possibilities for restructurings and new operations, and everyone is free to do whatever he or she likes. It is quite possible that someone has already written something as good as or better than EDITHA. More likely, every further revision will have some fine ideas of its own that ought to be in the ultimate version(s). Perhaps *DDJ* can serve as a clearinghouse for pooling these ideas, and thus hasten development of a super-version (that will merit a super-name!).

```
                SCANIT
                SCAN1
0100
010C
011F A4 E8      SCAN3  LDY BYTES
0121 C0 01             CPY # 1 (test for 1-byte)
0123 D0 03             BNE NOTONE (skip if ≠ 1)
0125 20 9E 19          JSR ONE  (time delay)
0128 A2 09      NOTONE LDX # 9
012A A9 7F             LDA # 7F
012C 8D 41 17          STA PADD
012F 20 28 1F          JSR SCAND1
0132 60                RTS
0133 A6 EB      SEARCH LDX XINDEX (# bytes sought)
0135 A0 00             LDY # 0
0137 B1 E2      TESBYT LDA (CURAD),Y (get byte)
0139 D9 F9 00          CMP INH,Y  (match display?)
013C D0 06             BNE GONEXT (≠, new opcode)
013E C8                INY        (=, next byte)
013F CA                DEX        (matching completed?)
0140 D0 F5             BNE TESBYT (no, continue)
0142 F0 73             BEQ ADVAL  (match found!)
0144 20 80 00   GONEXT JSR DETLEN
0147 20 8D 17          JSR ADVANC
014A B0 E7             BCS SEARCH  (more program)
014C 90 69             BCC ADVAL (no match found)
014E A2 04      GOKEY  LDX # 4 (user input 4 bytes)
0150 86 EA             STX COUNT (stored in COUNT)
0152 20 00 02          JSR READIN (go get key input)
0155 A8                TAY (move input byte into Y)
0156 F0 5F             BEQ ADVAL  (an ERASE exit)
0158 C9 10             CMP #$ 10 (hi or lo command?)
015A B0 25             BCS HICMD (hi, go to IDZP op)
015C C9 0C             CMP #$ 0C (double-command?)
015E 90 0A             BCC ANDOUT (no, go to ANDOUT)
0160 D0 5B             BNE ERROR (exit, only 0C legal)
0162 8A                TXA  (X is 00 for true double)
0163 D0 58             BNE ERROR (false, wipe-out)
0165 20 80 17          JSR BEGIN (set CURAD to start)
0168 B0 4D             BCS ADVAL (return to editor)
016A 29 03      ANDOUT AND # 3 (retain bits 0 and 1)
016C F0 4F             BEQ ERROR (04, 08, not legal)
016E 85 EB             STA XINDEX (store for SEARCH)
0170 C0 04             CPY # 4  (SEARCH command?)
0172 B0 BF             BCS SEARCH  (yes, go do it)
```

```
Ø174  85 E8          STA BYTES (store for ops)
Ø176  2Ø DF 17       JSR LIMITS (OK to write?)
Ø179  9Ø 3C          BCC ADVAL  (no, exit)
Ø17B  2Ø EØ Ø1       JSR WRITIN (write data)
Ø17E  FØ 1E          BEQ DACMD (display data)

Ø18Ø  EA             NOP  (just a leftover!)

Ø181  C9 8F    HICMD CMP #$ 8F (command too hi?)
Ø183  BØ 38          BCS ERROR (yes, exit)
Ø185  C5 FB          CMP POINTH  (confirmed?)
Ø187  DØ 34          BNE ERROR (no, exit)
Ø189  4A             LSR A  (even or odd #?)
Ø18A  BØ 31          BCS ERROR (odd, exit)
Ø18C  ØA             ASL A (restore command)
Ø18D  1Ø Ø2          BPL NOADD (< $8Ø, use as is)
Ø18F  69 58          ADC #$ 58 (raise to $D8-up)
Ø191  AA       NOADD TAX (move command to X index)
Ø192  A5 F9          LDA INH (pickup first byte)
Ø194  95 Ø1          STA HIBASE+1,X (IDZP write)
Ø196  A5 FA          LDA POINTL (pickup second)
Ø198  95 ØØ          STA HIBASE,X (IDZP write)

Ø19A  D8       EDITHA CLD  (clear decimal mode)
Ø19B  2Ø 8Ø ØØ       JSR DETLEN (get opcode bytes)
Ø19E  2Ø ØØ Ø1 DACMD  JSR SCANIT

Ø1A1  C9 12     GCMD CMP #$ 12
Ø1A3  FØ ØF          BEQ STPKEY (+ key = $12)
Ø1A5  BØ A7          BCS GOKEY (GO key = $13)
Ø1A7  C9 1Ø          CMP #$ 1Ø
Ø1A9  9Ø ØC          BCC ADVAL (any hex key)
Ø1AB  FØ 1F          BEQ ADKEY (AD key = $1Ø)

Ø1AD  2Ø B2 17       JSR MOVEUP (DAKEY deletion)
Ø1BØ  BØ E8          BCS EDITHA (OK, done)
Ø1B2  9Ø Ø3          BCC ADVAL (can't be done)

Ø1B4  2Ø 8D 17 STPKEY JSR ADVANC (to next opcode)
Ø1B7  A6 E2     ADVAL LDX CURAD (to be displayed)
Ø1B9  A5 E3           LDA CURAD+1  ( "    "   )
Ø1BB  BØ Ø2           BCS INFORM  (not an error)

Ø1BD  AØ EE     ERROR LDY #$ EE  (display EE)
Ø1BF  84 F9    INFORM STY INH  (display Y in data)
Ø1C1  86 FA           STX POINTL (address low)
Ø1C3  85 FB           STA POINTH (address high)
Ø1C5  2Ø 1F 1F  ERR1  JSR SCANDS (display message)
Ø1C8  DØ FB           BNE ERR1 (while key is held)
Ø1CA  FØ CE           BEQ EDITHA (back to editor)
Ø1CC  2Ø DF 17 ADKEY  JSR LIMITS (write OK?)
Ø1CF  9Ø E6           BCC ADVAL (no, exit)
Ø1D1  86 EA           STX COUNT (sets to ØØ)

Ø1D3  2Ø ØØ Ø2        JSR READIN (get key input)
Ø1D6  FØ Ø3           BEQ ACCEPT (all-hex, OK)
Ø1D8  CA              DEX   (if AD-AD, X was ØØ)
Ø1D9  1Ø E2           BPL ERROR (not AD-AD, exit)
Ø1DB  2Ø EØ Ø1 ACCEPT JSR WRITIN (write input)
Ø1DE  FØ BA           BEQ EDITHA (back to editor)

Ø1EØ  2Ø A4 ØØ WRITIN JSR MVDOWN (open up space)
Ø1E3  A6 E8           LDX BYTES (# bytes to write)
Ø1E5  AØ ØØ           LDY # Ø
Ø1E7  B9 F9 ØØ STORIT LDA INH,Y  (pickup byte)
Ø1EA  91 E2           STA (CURAD),Y (write it in)
Ø1EC  C8              INY   (for next pickup)
Ø1ED  CA              DEX  (if = Ø, all written)
Ø1EE  DØ F7           BNE STORIT (≠ Ø, next byte)
Ø1FØ  6Ø              RTS  (return to caller)

(coding for subroutine READIN)

ØØØØ  A2 Ø1    READIN LDX # 1  (set X to 1)
ØØØ2  86 E8           STX BYTES (display 1 byte)
ØØØ4  CA              DEX   (X now = ØØ)
ØØØ5  86 EB    NEXKEY STX XINDEX  (saves X)
ØØØ7  2Ø ØC Ø1 READHI JSR SCAN1 (read first key)
ØØØA  C9 1Ø           CMP #$ 1Ø  (a hex key?)
ØØØC  9Ø Ø5           BCC SHIFT (yes, use it)
ØØØE  A6 EB           LDX XINDEX (no, restore X)
ØØ1Ø  E9 Ø4           SBC # 4  (subtract, Acc - 4)
ØØ12  6Ø              RTS  (return to caller)
ØØ13  ØA       SHIFT  ASL A  (hex into hi nybble)
ØØ14  ØA              ASL A
ØØ15  ØA              ASL A
ØØ16  ØA              ASL A
ØØ17  85 E9           STA RDTEMP (save hi nybble)
ØØ19  2Ø ØC Ø1 READLO JSR SCAN1 (read second key)
ØØ1C  C9 1Ø           CMP #$ 1Ø (a hex key?)
ØØ1E  BØ F9           BCS READLO (no, try again)
ØØ2Ø  Ø5 E9           ORA RDTEMP (add hi nybble)
ØØ22  A6 EB           LDX XINDEX (restore X)
ØØ24  95 F9           STA INH,X (byte into display)
ØØ26  E8              INX (increment X for next op)
ØØ27  A4 EA           LDY COUNT (=Ø only first AD)
ØØ29  DØ Ø3           BNE OPSKIP (not an opcode)
ØØ2B  2Ø 84 ØØ        JSR BYTCNT (get bytecount)
ØØ2E  86 E8    OPSKIP STX BYTES (BYTES must = X)
ØØ3Ø  E4 EA           CPX COUNT (is X = COUNT?)
ØØ32  DØ D1           BNE NEXKEY (no, continue)
ØØ34  6Ø              RTS (return to caller)
1780  A4 E1     BEGIN LDY BEGAD+1
1782  A6 EØ           LDX BEGAD
1784  E8               INX (true program start)
1785  86 E2           STX CURAD (resets low)
1787  DØ Ø1           BNE NOINY (high is OK)
```

```
1789  C8          INY  (increment high)
178A  84 E3  NOINY STY CURAD+1 (reset high)
178C  60          RTS (return to caller)

178D  18     ADVANC CLC
178E  A5 E2         LDA CURAD
1790  65 E8         ADC BYTES
1792  85 E2         STA CURAD
1794  90 02         BCC PRELIM
1796  E6 E3         INC CURAD+1
1798  A2 04  PRELIM LDX # 4 (for 2 adchecks)
179A  B5 E1  INLIMT LDA BEGAD+1,X
179C  A8            TAY   (save hi in Y)
179D  D5 DF         CMP MOVAD+1,X
179F  90 0E         BCC LIMRET (error exit)
17A1  D0 08         BNE TESTEX (hi ad is safe)
17A3  B5 E0         LDA BEGAD,X (test lo ad)
17A5  D5 DE         CMP MOVAD,X
17A7  90 06         BCC LIMRET (error exit)
17A9  F0 05         BEQ ERREX (if ads=, error)
17AB  CA     TESTEX DEX
17AC  CA            DEX
17AD  D0 EB         BNE INLIMT (≠0, continue)
17AF  60     LIMRET RTS  (return to caller)
17B0  18      ERREX CLC (clear carry, error)
17B1  60            RTS

17B2  20 98 17 MOVEUP JSR PRELIM (delete OK?)
17B5  90 27         BCC MVURET (no, exit)
17B7  84 DF         STY MOVAD+1
17B9  A6 E2         LDX CURAD
17BB  86 DE  UPLOOP STX MOVAD
17BD  A4 E8         LDY BYTES
17BF  B1 DE         LDA (MOVAD),Y
17C1  A0 00         LDY # 0
17C3  91 DE         STA (MOVAD),Y
17C5  E8            INX
17C6  D0 02         BNE CHKEND
17C8  E6 DF         INC MOVAD+1
17CA  E4 E4  CHKEND CPX ENDAD
17CC  D0 ED         BNE UPLOOP
17CE  A4 DF         LDY MOVAD+1
17D0  C4 E5         CPY ENDAD+1
17D2  90 E7         BCC UPLOOP
17D4  8A            TXA (CURAD in X now=ENDAD)
17D5  E5 E8         SBC BYTES
17D7  85 E4         STA ENDAD
17D9  B0 03         BCS MVURET
17DB  C6 E5         DEC ENDAD+1
17DD  38            SEC
17DE  60     MVURET RTS  (return to caller)
17DF  A2 06  LIMITS LDX # 6 (for 3 adchecks)
17E1  D0 B7         BNE INLIMT

80  A0 00   DETLEN LDY # 0
82  B1 E2          LDA (CURAD),Y
84  A0 01   BYTCNT LDY # 1
86  2C 81 19       BIT TES3 ($08 in 1981)
89  D0 08          BNE HALFOP (bit 3 = 1)
8B  C9 20          CMP #$ 20 (is it JSR?)
8D  F0 0E          BEQ THREE (length is 3)
8F  29 9F          AND #$ 9F (test bits 74321 0)
91  D0 0B          BNE TWO (if ≠0, length is 2)
93  29 15  HALFOP AND #$ 15 (retain bits 420)
95  C9 01          CMP # 1 (is only bit 0 = 1?)
97  F0 05          BEQ TWO (if yes, length 2)
99  29 05          AND # 5  (retain bits 2,0)
9B  F0 02          BEQ ONE1 (if =0, length one)
9D  C8       THREE INY   (length 3 exit)
9E  C8         TWO INY   (length 2 exit)
9F  84 E8     ONE1 STY BYTES (save bytecount)
A1  84 EA          STY COUNT (save it again)
A3  60             RTS  (return to caller)

A4  A5 E5  MVDOWN LDA ENDAD+1  (this will set
A6  85 DF         STA MOVAD+1   MOVAD=ENDAD)
A8  A6 E4         LDX ENDAD
AA  E8            INX
AB  F0 05         BEQ DECEX
AD  8A     MVLOOP TXA            (this will
AE  D0 02         BNE DECEX      decrement
B0  C6 DF         DEC MOVAD+1     MOVAD)
B2  CA      DECEX DEX
B3  86 DE         STX MOVAD
B5  A0 00         LDY # 0         (this will
B7  B1 DE         LDA (MOVAD),Y     move
B9  A4 E8         LDY BYTES       program)
BB  91 DE         STA (MOVAD),Y
BD  E4 E2         CPX CURAD      (this will
BF  D0 EC         BNE MVLOOP      exit when
C1  A4 DF         LDY MOVAD+1     MOVAD=CURAD)
C3  C4 E3         CPY CURAD+1
C5  90 EF         BCC BRKERR (to 00 at 01B6)
C7  D0 E4         BNE MVLOOP
C9  18            CLC  (readjust ENDAD)
CA  A5 E4         LDA ENDAD
CC  65 E8         ADC BYTES
CE  85 E4         STA ENDAD
D0  90 02         BCC MVDRET
D2  E6 E5         INC ENDAD+1
D4  60      MVDRET RTS
```

IEEE COMPUTER SOCIETY MICROPROCESSOR STANDARDS COMMITTEE

PROPOSED STANDARD for the S-100 BUS

PRELIMINARY SPECIFICATION

BY GEORGE MORROW
Thinker Toys
1201-10th Street
Berkeley, CA 94707
(415) 527-7548

and

HOWARD FULLMER
Parasitic Engineering Inc.
P.O. Box 6314
Albany, CA 94706
(415) 547-6612

The following is a preliminary specification for the computer bus commonly known as the S-100 Bus. This bus was first introduced by MITS Inc. with their Altair kit. It has since spread throughout the electronics industry and beyond. Today over a hundred manufacturers make products which claim to be compatible with the S-100 bus even though, until now, no complete specification has been available.

This document is a specification for both timing and signal discipline. Signal discipline is described using the Bus Master/Bus Slave language long associated with Digital Equipment's PDP 11. This point of view facilitated the development of a simple and highly realizable DMA protocol, the extended addressing capabilities, and the 16 bit wide data path proposals. These extensions to the original Altair bus represent a significant advance in the state of the art of small computers. These advances are a direct result of a continuing dialogue with a large number of interested people who have contributed their thoughts and ideas to the standards committee.

S-100 Bus Signal Definitions

Bus Signal Notes

1. There are three types of signals on the S-100 bus. M stands for Bus Master. Signals designated by M are those which a Bus Master must generate. The Bus Master currently controlling the bus has the responsibility for faithfully generating *all* signals of type M during the duration of its control of the bus.

   S stands for Bus Slave. A Bus Slave need generate only that subset of type S signals which are necessary to communicate with Bus Masters which have the ability to address the slave.

   B stands for Bus. Any bus signal which is not of type M or S is by default type B. This is not to say that some Bus Master is not in fact generating one or more type B signals. Rather a type B signal is one that (a) not all Bus Masters are required to generate, and (b) not any Bus Slave is required to generate.

   A Bus Master is, by definition, a bus device which generates at least all of the type M signals. A Bus Slave is a bus device which generates some subset of type S signals. A Bus Master may also be a bus slave and vice-versa. Memory devices are almost always Bus Slaves while DMA devices are usually both a Bus Master (data transfers) and a Bus Slave (accepting commands). Central Processing Units are usually Bus Masters.
2. The 8 status signals are: SMEMR, SINP, SM1, SOUT, SHLTA, SSTACK (not specified), $\overline{\text{SWO}}$, and SINTA.
3. The 6 command/control signals are: PHLDA, PSYNC, PDBIN, PINTE (not specified), $\overline{\text{PWR}}$, and PWAIT.
4. The 16 address signals are: A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, and A0.
5. Data output is specified relative to a Bus Master. By definition, data which is transmitted by a Bus Master is always data output and occurs on the DO bus.
6. Data input is specified relative to a Bus Master. By definition, data which is received by a Bus Master is always data input and occurs on the DI bus.
7. A bus cycle is a collection of bus states (BS_α). A bus cycle always starts with a BS_1 state which is followed by a BS_2 state. After BS_2 comes an indeterminate number of BS_W states. A bus cycle may have zero BS_W states or it may have an arbitrarily large number of BS_W states. BS_3 is the bus state which follows BS_W (or BS_2 if there are no BS_W states present). BS_3 is followed by zero to three BS_i states. A BS_3 or BS_i state terminates a bus cycle.
8. The DO bus is the following set of signals: DO7, DO6, DO5, DO4, DO3, DO2, DO1, DO0.
9. The DI bus is the following set of signals: D17, D16, D15, D14, D13, D12, D11, D10.
10. These signals should be generated by an open collector bus driver capable of sinking at least 20 ma. at no more than .5 volts.
11. During the last half of bus state 1 and the first half of bus state 2, the DO lines are used to furnish extended addressing to slaves that can utilize this information.

For 16 bit masters and slaves, the DI and DO lines gang together to form a 16 bit bidirectional data bus with the DO lines forming the low order byte while the DI lines accomodate the high order byte. The configuration of the DI and DO buses is governed by the signals $\overline{\text{SXTRQ}}$ and $\overline{\text{SXTN}}$. When both these signals are low, the DI and DO lines become bidirectional. Otherwise, DI carries the data to the current master while DO carries data to the addressed slave.

Signal Characteristics

Bus drivers must sink at least 24 ma. at no more than .5 volts. Except for open collector drivers, they must source at least 2 ma. at no less than 2.4 volts.

Bus receivers must have diode clamps to prevent excessive negative excursions. Any bus signal less than 0.8 volts must be recognized as a logic zero and any signal more than 2 volts must be interpreted as a logic one.

Receivers are to source no more than 0.8 ma. at .5 volts and are to sink no more than 80 μa. at 2.4 volts. The capacitance load of an input from the bus must not exceed 25 pf.

Bus State Comments

BS_1 is the initial bus state of a bus cycle. The address lines are in a state of flux during through most of BS_1 and PSYNC is active starting with the second half of BS_1.

BS_2 is the second bus state when address data, status, and ready signals become stable.

BS_W states occur as needed to synchronize a Bus Master with a Bus Slave which has brought one of the ready lines false.

BS_3 is the data transfer state when a Bus Master transfers data to a slave or vice-versa.

BS_i is a state during which the bus is idle.

Timing Notes

All timing references in the timing diagrams are specified at the midpoint of the rising or falling edge of the signal. Rise and fall times are not to exceed 50 ns.

All signals referred to in the timing diagrams are S-100 Bus signals with the exceptions in note 6.

1. The falling edge of $\overline{PWR}$ must occur within the area shown. The rising edge must occur within a similar area of the next Bus State.
2. BS_α is either BS_2 or BS_W.
3. Addresses, Data, and Status signals must remain stable during BS_W.
4. The interrupt lines must be stable for the period shown in the Bus State preceding BS_1 of an op-code fetch. The proposal is that when an interrupt line is true, it remains true until the CPU responds. Normally, this response would be an I/O instruction that addresses the interrupting device.
5. The rising edge of PWAIT must occur within the area shown. The falling edge must occur within a similar area following the rising edge of the logical AND of PRDY and XRDY.
6. Signals prefixed by "DMA" refer to internal logic of the new Bus Master. These signals control the buffers of this Bus Master which drive the Command and Control, Status, Address, and Data output Bus lines. The timing diagram depicts logic levels which are high when these buffers are disabled.
7. BS_β is either BS_3 or BS_i.
8. BS_γ is either BS_i or BS_1.
9. Extended address bits A16 through A23 are multiplexed on the DO lines. They must occur as shown within 250 ns after the rising edge of ϕ_2 during BS_1. During a write, data must appear on these lines 100 ns before PWR is active. The PWR signal can be asserted as soon as 5 ns after the rising edge of ϕ_2 during BS_2. Therefore, under worst case conditions, the extended addresses are on the bus for 155 ns. Masters can generate PWR later as shown. This would allow the extended addresses to be on the bus longer. However, slaves should be designed to correctly respond in the worst case situation.
10. To avoid conflict on the DO lines during a 16 bit read, as in the situation described in note 9 above, the extended addresses must be removed from the DO lines before the rising edge of PDBIN.

Direct Memory Access (DMA) Requirements

A DMA cycle is a special case of a Bus Master taking over the Bus to execute a memory read or write cycle. A DMA device is required to generate *all* type M (Bus Master) signals on the Bus.

The Bus Exchange

$\overline{PHOLD}$ is the signal used by one Bus Master to request that another Bus Master give up control of the Bus. $\overline{PHOLD}$ must not be asserted true unless PHLDA is false *and* PRIORITY (if implemented) is true.

One Bus Master (CPU) relinquishes control of the Bus to another (DMA) as shown in the Bus Exchange Timing diagram. The DMA device must control the CPU's bus drivers through the use of $\overline{ADD\ DSB}$, $\overline{DO\ DSB}$, $\overline{STAT\ DSB}$ and $\overline{C/C\ DSB}$. It must also control its own bus drivers through the use of signals similar to those shown in note 6.

The CPU (current master) and the DMA device (new Master) must *both* drive the Command and Control signals for at least 200 ns at two different periods as shown in the Bus Exchange timing diagram. During these two times, the Command and Control signals for at least 200 ns at two different periods as shown in the Bus Exchange timing diagram. During these two times, the Command and Control signals are required to have the following levels:

1. PSYNC = 0
2. PWAIT = 0
3. PHLDA = 1
4. PDBIN = 0
5. PWR = 1

The DMA cycle timing sequence which follows is a suggested implementation that meets all the requirements of the generalized Bus Exchange timing. The sequence is controlled by the edges of ϕ_2. At some previous time, $\overline{PHOLD}$ was asserted according to the limitations described in the first paragraph of this section. PHLDA is asserted true by the CPU during BS_3 of the last CPU bus cycle. The bus exchange begins on the falling edge of ϕ_2 while PHLDA is true (labeled 1 on the timing diagram). The DMA bus cycle then proceeds as described in the following section. At edge 8 of ϕ_2, $\overline{PHOLD}$ is driven false by the DMA device and henceforth the CPU is again in control of the bus.

Proposed DMA Cycle Sequence

ϕ_2 edge:

1. CPU Address and Data bus drivers turned off. DMA Command and Control drivers turned on. The CPU and DMA Command and Control signals must match values described in the previous section.
2. CPU Status and Command and Control drivers turned off. DMA Address, Data output and Status drivers turned on. PSYNC = 1.
3. No change.
4. PSYNC = 0. PDBIN = 1 if memory read, or $\overline{PWR}$ = 0 if memory write.
5. No change.
6. PDBIN = 0 and $\overline{PWR}$ = 1.
7. CPU Command and control drivers turned on. DMA Address and Data output drivers turned off.
8. CPU Address, Data output, and Status drivers turned on. DMA Status and Command and Control drivers turned off. $\overline{PHOLD}$ = 1.

Acknowledgement

We would like to thank the other members of the Microprocessor Standards Committee for their support and invaluable comments. Special thanks to Robert Stewart who is chairman of the Committee. His leadership and organizational skills have been a great aid to all of us.

We would also like to extend our thanks to Kells Elmquist and Steve Edleman of Ithaca Audio. Both of these gentlemen flew out to the Second Computer Faire to discuss their ideas about S-100 bus extensions with us. Several of their ideas were truly inspirational. Their intellectual prodding was the force which created the extended addressing and the bi-directional data bus proposals. Kells Elmquist is responsible for another proposal which we will term "logical daisy chaining". This will be incorporated into the standard soon and will make possible a significant advance in interrupt structures within the present S-100 bus.

IEEE Computer Society Microprocessor Standards Committee

Proposed Standard for the S-100 Bus

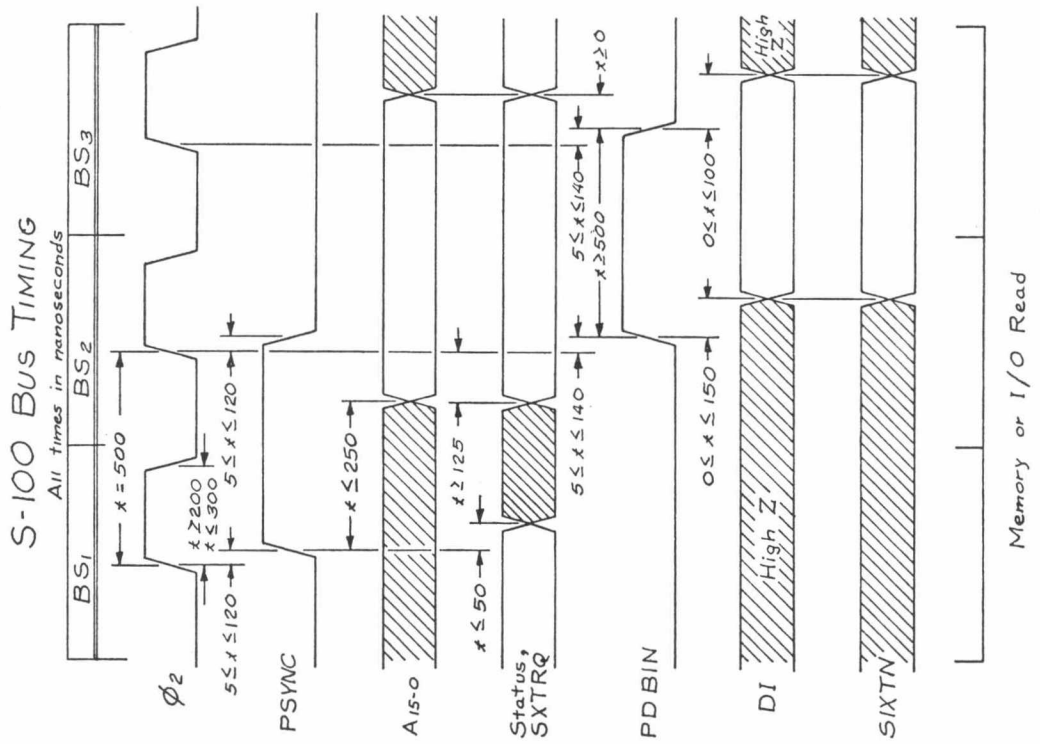
S-100 BUS SIGNAL DEFINITIONS:

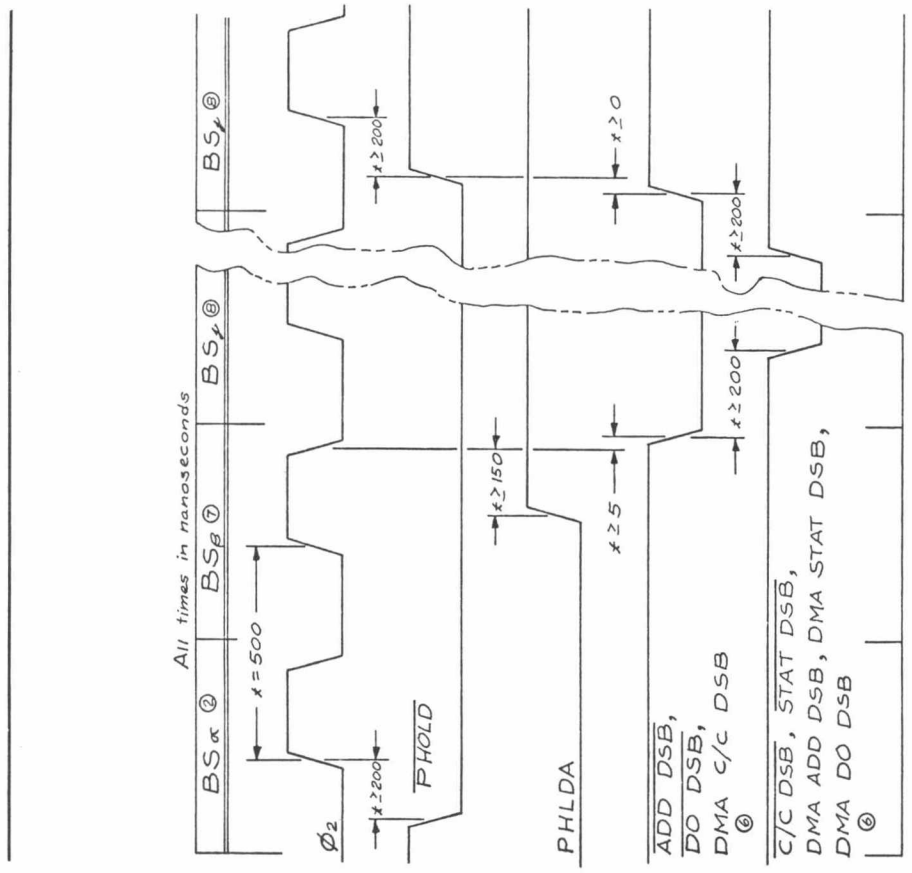
Pin No.	Signal Name & Type	Polarity	
1	+8 volts (B)[1]		Instantaneous minimum greater than 7 volts, instantaneous maximum less than volts, average maximum less than volts.
2	+16 volts (B)		Instantaneous minimum greater than 14 volts, instantaneous maximum less than 35 volts, average maximum less than 20 volts.
3	XRDY (S)[1,10]	positive	One of two ready inputs to the current Bus Master. The bus is ready when both these ready inputs are true.
4	$\overline{VI_0}$ (S)[10]	negative	Vectored interrupt line Ø.
5	$\overline{VI_1}$ (S)[10]	"	" 1.
6	$\overline{VI_2}$ (S)[10]	"	" 2.
7	$\overline{VI_3}$ (S)[10]	"	" 3.
8	$\overline{VI_4}$ (S)[10]	"	" 4.
9	$\overline{VI_5}$ (S)[10]	"	" 5.
10	$\overline{VI_6}$ (S)[10]	"	" 6.
11	$\overline{VI_7}$ (S)[10]	"	" 7.
12	-	-	Not specified.
13	-	-	"
14	-	-	"
15	-	-	"
16	-	-	"
17	-	-	"
18	$\overline{\text{STAT DSB}}$ (M)[1,10]	negative	The control signal to disable the 8 status signals[2].
19	$\overline{\text{C/C DSB}}$ (M)[10]	"	The control signal to disable the 6 command/control signal[3].
20	UNPROT	-	Not specified.
21	SS		Not specified.
22	$\overline{\text{ADD DSB}}$ (M)[10]	negative	The control signal to disable the 16 address signals[4].
23	$\overline{\text{DO DSB}}$ (M)[10]	"	The control signal to disable the 8 data output[5] signals.
24	ϕ_2 (B)	positive	The master timing signal for the bus.
25	ϕ_1	-	Not specified.
26	PHLDA (M)	positive	A command/control signal used in conjunction with PHOLD to coordinate Bus Master transfer operations.
27	PWAIT (M)	"	The acknowledge signal to either of the bus ready signals XRDY, PRDY or to a HLT instruction.
28	PINTE	"	Not specified.
29	A5 (M)	"	Address bit 5.
30	A4 (M)	"	Address bit 4.
31	A3 (M)	"	Address bit 3.
32	A15 (M)	"	Address bit 15 (most significant).
33	A12 (M)	"	Address bit 12.
34	A9 (M)	"	Address bit 9.
35	DO1 (M)/EA1 (M)/DATA1 (M/S)	"	Data out bit 1, Extended address bit 1, Bidirectional data bit 1.
36	DOØ (M)/EAØ (M)/DATAØ (M/S)	"	Data out bit Ø, Extended address with bit Ø, Bidirectional data bit Ø (least significant)
37	A10 (M)	"	Address bit 10.
38	DO4 (M)	positive	Data out bit 4.
39	DO5 (M)	"	" 5.
40	DO6 (M)	"	" 6.
41	D12 (M)	"	Data in[6] bit 2.
42	D13 (M)	"	Data in bit 3.
43	D17 (M)	"	Data in bit 7 (most significant).
44	SM1 (M)	"	The status signal which indicates that the current bus cycle[7] is an op-code fetch.
45	SOUT (M)	"	The status signal identifying the data transfer bus cycle of an OUT instruction.
46	SINP (M)	"	The status signal identifying the data transfer bus cycle of an IN instruction.
47	SMEMR (M)	"	The status signal identifying bus cycles which transfer data from memory to a Bus Master which are not interrupt acknowledge instruction fetch cycle(s).
48	SHLTA (M)	"	The status signals which acknowledges that a HLT instruction has been executed.
49	$\overline{\text{CLOCK}}$	-	Not specified.
50	GND		Signal and pwer ground.

PRELIMINARY SUBJECT TO REVISION

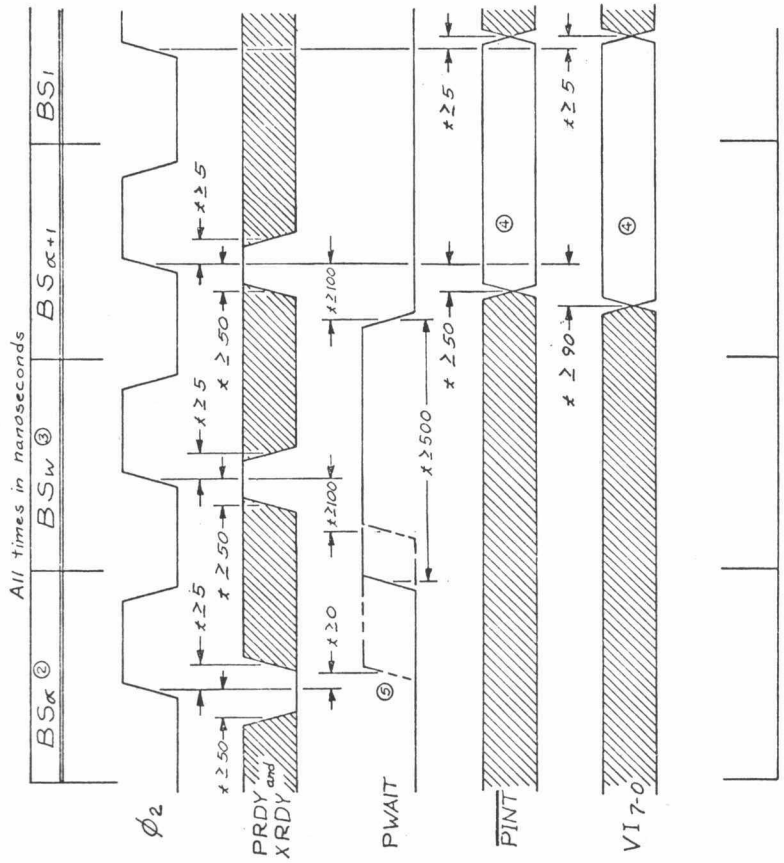
51	+8 volts (B)		See comments above for pin #1.
52	-16 volts (B)		Instantaneous maximum less than -14 volts, instantaneous maximum greater than -35 volts, average minimum greater than -20 volts.
53	$\overline{\text{SSWI}}$	-	Not specified.
54	$\overline{\text{EXT CLR}}$	negative	A reset signal to reset Bus Slaves. When this signal goes low, it must stay low for at least 3 bus states.
55	-	-	Not specified.
56	-	-	"
57	-	-	"
58	-	-	Not specified.
59	$\overline{\text{SXTRQ}}$ (M)	negative	Master signal which requests 16 bit wide slaves to respond by asserting SXTN.
60	-	-	Not specified.
61	$\overline{\text{SXTN}}$ (S)	negative	The signal generated by 16 bit slaves in response to the 16 bit request signal SXTRQ.
62	-	-	Not specified.
63	-	-	"
64	-	-	"
65	-	-	"
66	-	-	"
67	$\overline{\text{PHANTOM}}$ (B)	negative	A bus signal which disables normal slave devices and enables phantom slaves -- primarily used for bootstrapping systems without hardware front panels.
68	MWRITE (B)	positive	The logical negation of PWR and SOUT; this signal must follow PWR by not more than 30 ns.
69	$\overline{\text{PS}}$	-	Not specified.
70	PORT	-	"
71	RUN	-	"
72	PRDY (S)[10]	positive	See comments above for pin #3.
73	$\overline{\text{PINT}}$ (S)[10]	negative	The pirmary interrupt request bus signal.
74	$\overline{\text{PHOLD}}$ (M)[10]	"	The command/control signal used in conjunction with PHLDA to coordinate Bus Master transfer operations.
75	$\overline{\text{PRESET}}$ (B)[10]	"	The reset signal to reset Bus Master devices. When this signal goes low, it must stay low for at least 3 bus states.
76	PSYNC (M)	positive	The command/control signal identifying BS_1. (See bus states comments.)
77	$\overline{\text{PWR}}$ (M)	negative	The command/control signal signifying the presence of valid data on the DO bus[8].
78	PDBIN (M)	positive	The command/control signal that requests data on the DI bus[9] from the currently addressed slave.
79	AØ (M)	"	Address bit Ø (least significant).
80	A1 (M)	"	Address bit 1.
81	A2 (M)	"	" 2.
82	A6 (M)	"	" 6.
83	A7 (M)	"	" 7.
84	A8 (M)	"	" 8.
85	A13 (M)	"	" 13.
86	A14 (M)	"	" 14.
87	A11 (M)	"	" 11.
88	DO2 (M)/A18 (M)/DATA2(M/S)	positive	Data out bit 2, extended address bit 2, & bidirectional data bit 2.
89	DO3 (M)/A19 (M)/DATA3 (M/S)	"	Data out bit 3, extended address bit 3, & bidirectional data bit 3.
90	DO7 (M)/A23 (M)/DATA7 (M/S)	"	Data out bit 7, extended address bit 7, & bidirectional data bit 7 (most significant).
91	DI4 (S)/DATA12 (M/S)	positive	Data in bit 4 & bidirectional data bit 12.
92	DI5 (S)/DATA13 (M/S)	"	Data in bit 5 & bidirectional data bit 13.
93	DI6 (S)/DATA14 (M/S)	"	Data in bit 6 & bidirectional data bit 14.
94	DI1 (S)/DATA9 (M/S)	"	Data in bit 1 & bidirectional data bit 9.
95	DIØ (S)/DATA8 (M/S)	"	Data in bit Ø & bidirectional data bit 8.
96	SINTA (M)	"	The status signal identifying the instruction fetch cycle(s) that immediately follow an accepted interrupt request presented on PINT.
97	$\overline{\text{SWO}}$ (M)	negative	The status signal identifying a bus cycle which transfers data from a Bus Master to a slave.
98	SSTACK	-	Not specified.
99	$\overline{\text{POC}}$ (B)	negative	The power-on clear signal for all bus devices; when this signal goes low, it must stay low for at least 3 bus states.
100	GND		Signal and power ground.

PRELIMINARY SUBJECT TO REVISION

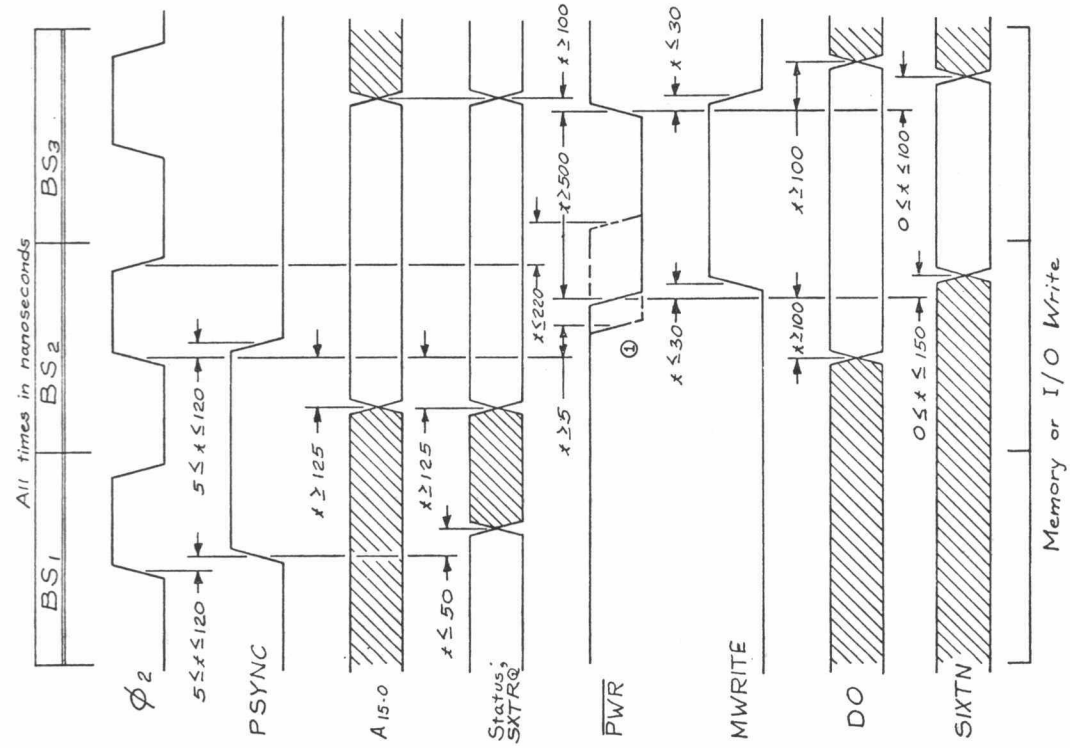




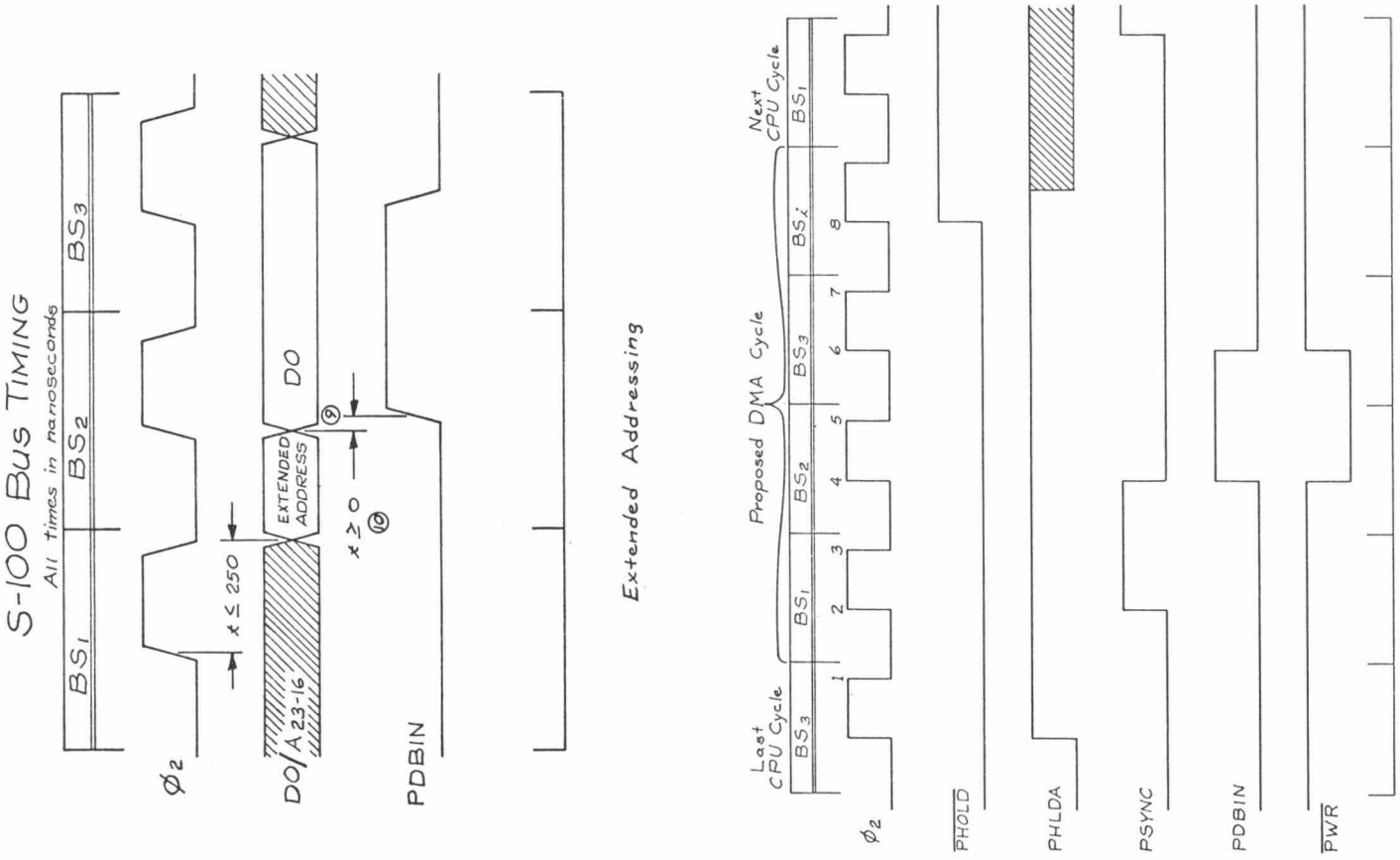
Bus Exchange Timing



Interrupt and Wait Timing



Memory or I/O Write



PRELIMINARY SUBJECT TO REVISION

TSA APPLICATIONS AND DEVELOPMENT TOOLS

News Release *Received: 78 March 10*

Now available, a full complement of Z80/8080/8085 software devlopment tools. These are disk based, and oriented towards the popular CP/M Operating System (TM Digital Research). System development tools include a relocatable linking macro assembler with linking loader, cross-reference generator, and full library of modules. With the assembler is included a symbolic debugger allowing user defined symbols. Higher level language support is provided by interface with Micro-Soft FORTRAN.

For advanced systems, TSA/OS is an upward compatable CP/M-like operating system providing CRT screen control, automatic library search, an extended batch mode with turn-key system capability, as well as an advanced configuration scheme.

TSA Software has available a set of applications packages. The TSA Database System uses a mixture of assembly code and Fortran. The system uses table driven screen and record formats; and has a mini-compiler to optimize record search capability. The TSA Word Processor uses a normal terminal to provide natural text editing with advanced formatting features, including proportional printing. Full use of disk files is provided, as well as file merging for mailing list and similar uses.

The product line is S-100 oriented, but is flexible for all other main frames, as all special functions are handled through the operating system. Package prices: $100 and up, dealer and OEM inquiries invited. TSA Software, 5 North Salem Rd., Ridgefield, CT 06877; (203) 438-3954.

THE SOFTWARE EXCHANGE

New Publication *Received: 78 April 4*

The Software Exchange is a new publication devoted to the exchange of ready to use software for business and the home. *The Software Exchange* provides classified listings for computer software. You will be able to match your application and computer to those listed. Each program will have a description of its operation, hardware requirements, and where the provided materials can be obtained. If you have a special program you need, you can place a Software Search in *The Software Exchange*. It is very likely that someone has the software you are looking for.

In addition, all advertising will be devoted exclusively to software services available to micro and minicomputer users. *The Software Exchange* is a bi-monthly publication available soon at computer stores for $1.50 per issue, and by subscription for $8 per year (six issues). Box 55056, Valencia, CA 91355.

CORRECTION

Gentlemen: *Received: 78 March 28*

With regard to your reproduction of the Stanford University News Release, I wish to correct one of your references. The "memorandum" from CMU about the robot was written by Mark *Fox* (not Mark Gos) and Brian Reid.

Sincerely,
Mark Sherman

Graduate Student
CMU Dept. of Computer Science

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 26 | June/July 1978 | Volume 3, Issue 6

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

Intelligent Terminal Implementation on the S-100 Bus *Tim Pugh* 4
Disk file transfer under CP/M using the IDS 88-Modem Board

Autodump *Richard Emerson* 18
A memory dumper for Poly-88 tape system

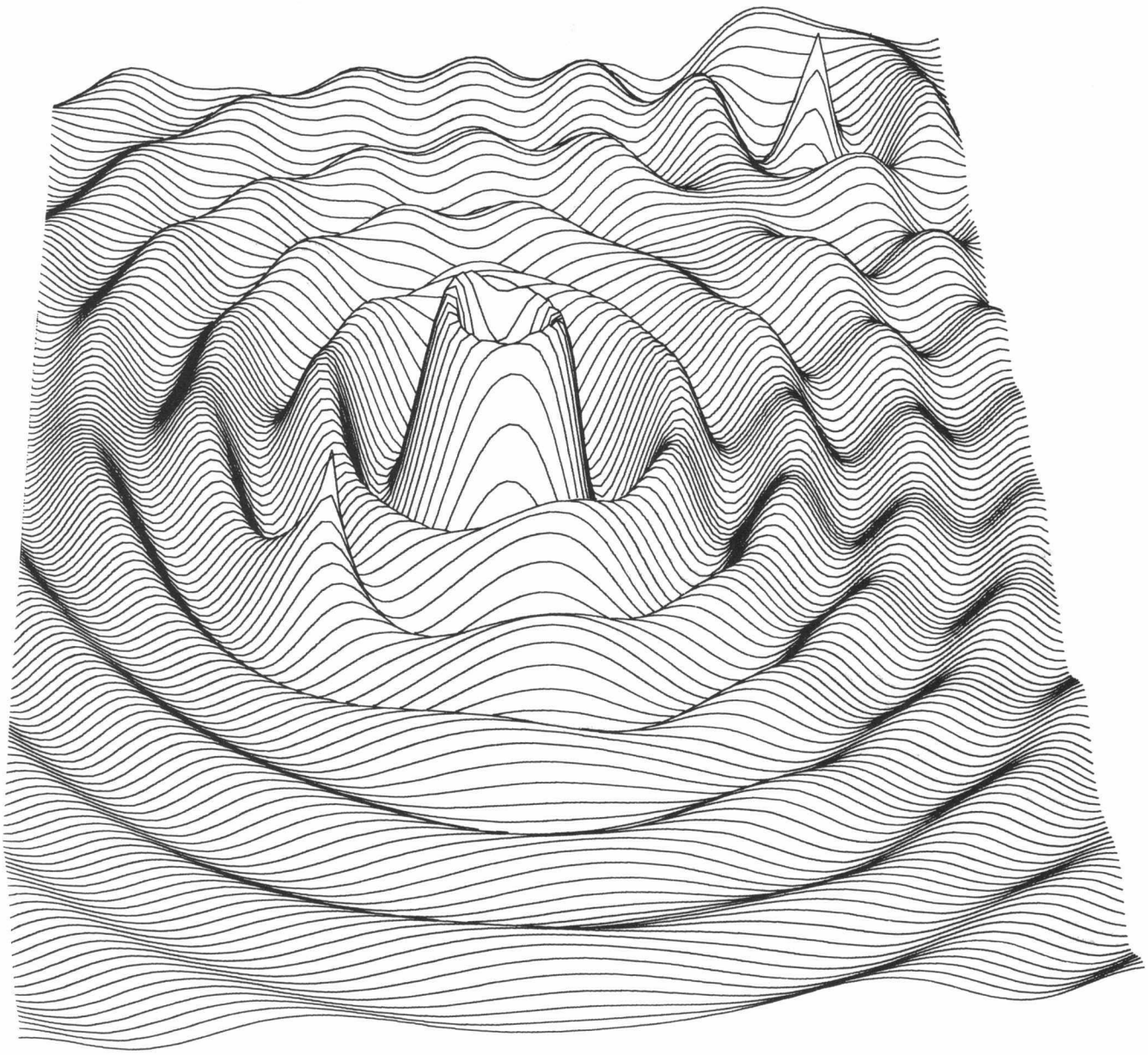
High Speed Cassette I/O for the KIM-1 *Theodore E. Bridge* 24

LAZARUS *Andrew Hertzfeld* 31
A program to resurrect BASIC programs on the Apple-II

A Tiny BASIC Extensions Package *Leor Zolman* 34

HARDWARE

The Zilog Z-8 *U.R. Source* 28
A PEEK at the new single-chip microcomputer from Zilog



© David Dameron 1979

MORE ON UNIX

Dear Dr. Dobb, *Received: 78 May 22*

In the April 1978 issue of *DDJ*, page 3, Pierre duPont wrote that he is planning to write an operating system similar to UNIX for his 8080. Enclosed is a copy of a letter I sent to Mr. duPont.

Jim Howell
555 Candlestick Way
San Jose, CA 95127

Pierre duPont
Salisbury School
Salisbury, CT 06068

Dear Pierre,

Your letter in *DDJ* attracted my interest. I have used UNIX for about 2 months and I really like it. Before you start writing your "UNIX", you may want to read "The UNIX Time-Sharing System" in the July 1974 issue of Communications of the ACM (vol 17, #7), pages 365-375. This article, which is part of the documentation for users of UNIX, gives a description of the structure of UNIX and describes some of the capabilities of the "shell" program. I can send you a copy of this article if you need it.

As for some of your questions: (1) 8080 interrupts. To get more than 8 interrupts for the 8080, the hardware must arrange to supply a CALL instruction, rather than a restart, for at least some devices when acknowledging an interrupt. Intel has a chip in the 8080 family (I don't know the number) which can do most of this for you. Each I/O device can generate a call to a different location, thus allowing as many interrupts as you want. The Intel chip handles 8 devices, I think.

(2) One way to store relocatable programs is to have, as part of the program, one or more (or maybe none?) records in the file, which are a list of the locations within the program which need relocating. These records must be distinguishable, of course, from program records. One problem with using an unused opcode is that any code may appear as an operand of an immediate instruction or in an area containing constants or tables.

(4) 8080 vs Z-80. Advantage of using the Z-80: the program(s) comprising your "UNIX" may be smaller than with the 8080. Advantages of using the 8080: the program would be distributable to more people, (assuming you plan to distribute it): and the 8080 is what you have now. The disk would appear to be the limiting factor in the speed of the system, so that the faster speed of the Z-80 relative to the 8080 is probably not an important consideration. The lack of unused opcodes for the Z-80 is not a problem if you use a relocation scheme such as in (2). These are the major considerations as I see them. I hope this helps you make your decision.

(6) I am interested in your "UNIX" (or whatever you end up calling it), but mainly from the standpoint of keeping up with developments in the computer hobbyist area. My system is the COSMAC ELF II (using the RCA 1802) and my choice for a bigger system would be one using the 6800, such as SWTPC or MSI.

Good luck and keep us informed (thru *DDJ*).

Sincerely,
Jim Howell

AND MORE. . .

Dear Sirs: *Received: 78 May 19*

The purpose of this letter is to introduce UNIX for the 8080 microprocessor. The system is called CHAOS, an acronym for *C*lairemont *H*igh *A*LTAIR *O*perating *S*ystem.

CHAOS is currently in operation at Clairemont High, and offers the user the following capabilities:

* A maximum of nine timeshared processes. A process may be an interactive user, lineprinter controller, and executing command, minicomputer translator, etc.
* A choice of three computer programming languages:
  BASIC—MITS Disk Extended Version 4.1 (modified for CHAOS).
  8080 Assembly—relocation table allows patching of relative addresses, and placement of the program anywhere in memory.
  Shell – operating system command lines that are contained in a file and may reference parameters in the calling command line via a macro structure.
  PASCAL—hopefully will soon become the fourth language supported by CHAOS.
* User-maintained system. All updating and expanding is easily accomplished due to the power of CHAOS.
* A complete accounting system that allows program/file sharing, access rights control and unlimited sub-accounts. CHAOS supports six file types: 8080 Machine Language, BASIC program, sequential data, Shell command line, random data and directory. Access rights include read, write, ownership and execute.
* Interfacing with the Hewlett-Packard 2000 Access System BASIC minicomputer and a text editor, assembler and loader on that system.
* Storage of programs and data on a maximum of eight floppy disk drives and one cassette tape drive.
* Mechanisms within the Kernel (timeshare controller) to handle semaphore requests and job queuing.
* Program nesting, recursion and inter-language calls via the CHAOS file module and stack schemes.

Our current hardware configuration includes:
IMSAI 8080 microprocessor
64K RAM
Memory Switching Device for CHAOS
2K PROM
6 serial input/output ports
Three parallel plus one serial input/output port
Disk Controller
Cassette Interface
4 Video Terminals
1 Printer/Terminal
Diablo Lineprinter
2 MITS Floppy Disk Drives
Cassette Player/Recorder

I would welcome any inquiries regarding CHAOS, and would be interested in learning about other like operating systems for the 8080 microprocessor.

Sincerely,
Mary Elizabeth Kroening
3632 Governor Drive
San Diego, CA 92122

Aha, do I sense the beginnings of a dialogue? —*TRW*

Intelligent Terminal Implementation for the S-100 Bus

DISK FILE TRANSFER UNDER CP/M USING THE IDS 88-MODEM BOARD

BY TIM PUGH
9655-M Homestead Court
Laurel, MD 20810
(301) 776-5253

There is now available a board for the S-100 BUS which, when coupled with appropriate software, can add a powerful new dimension to your system. The board I refer to is an Originate/Answer Modem board and is being produced by at least two manufacturers: D C Hayes Assoc. of Atlanta, GA and International Data Systems (IDS) of Falls Church, VA. With an Originate/Answer Modem you can implement the following functions:

1. Intelligent Terminal
2. Remote Console
3. PCNET Node

There are undoubtedly other functions which can be implemented but these three are fairly clearly defined.

The first of these functions (Intelligent Terminal) is implemented to a significant extent by the program which accompanies this article. The basic purpose of this function is to allow you to communicate with a large "Time Sharing Computer" (TSC) and effectively use the various components of your "Personal Computer" (PC) in carrying on this communication. For example, you may wish to use your video terminal for some of the communications and then switch to a printing terminal or hard copy device at the appropriate time. Also, you may wish to transfer data between the PC's floppy disk and the TSC.

This last activity is truly significant if you are taking programming courses or are a professional programmer whose company computer allows Demand Mode access; i.e., access via a terminal. It has been my experience that access to TSC systems in either a school or professional environment is seldom available during "prime time"; i.e., the hours of 8:00 AM to 5:00 PM. Thus, one must either enter his programs via cards—not a very exciting prospect—or do most of his work during "odd hours" when the available terminal lines are not heavily utilized. However, given the ability to transfer files between the TSC and the PC one can prepare his source input at his convenience using the text editor of his PC system. The source statements are stored on a floppy disk and later transferred to the TSC. Thus, the TSC is needed only for compilation and execution. Subsequent corrections to the source "deck" can be performed on the PC system. I find that most of the time I spend on a computer is used in editing source statements and comparatively little in compilation and execution. This means that only a small portion of one's work need be performed at "odd hours" and this can be performed in the comfort of one's home.

The second function (Remote Console) is my next project and should not be nearly so difficult as the first. With such a capability a businessman who has a computer system at his place of business need only purchase an additional terminal and acoustic coupler to bring the capabilities of that system into his home. Or the professional; e.g., engineer or programmer, can have access to his home system from his place of work. Or the person who wants to demonstrate the capabilities of a computer system to a potential customer can do so without having to carry the entire system with him.

The third function (PCNET Node) will allow both messages and program files to be transferred between two or more PC's or between a PC and a data facility whose communication protocol is compatible with that of the PCNET. This capability opens up all sorts of possibilities. The correspondence I have received to date indicates that the protocols for this network will be ready for general distribution in the first half of this year, with prototype PCNET software available in the second half of this year. On March 5th of this year a demonstration of existing PCNET software was given when an 8080 system in Sunnyvale, CA sent messages to a PET in San Jose, CA. Noise was injected on the phone lines and the two systems successfully detected bad "packets" and retransmitted good ones. Additional information concerning this can be had by contacting:

PCNET Committee
701 Welch Road, Suite 226
Palo Alto, CA 94304
(415) 328-2411

It is quite probable that given these functions and the ever increasing power of PC systems and software, many professionals will be able to perform a good deal of their work in the convenience of their home augmented by conferences between coworkers at their place of employment.

Modem Selection

In choosing a modem I was very concerned that it have a stable frequency source and good filtering characteristics. About 18 months ago I built the Pennywhistle modem (acoustic coupler) kit. It used a 555 timer IC with a two stage filter and phase locked loop. While this modem would work (after a rather tedious allignment) with one TSC, it was not reliable when interfacing with a second and would not function at all with a third. All three of these TSC's were within a radius of 20 miles of my terminal. I understand that certain design improvements have been made in the Pennywhistle since then.

With regard to choosing between an acoustic coupler and one of the S-100 BUS Originate/Answer Modem cards, the function of "Intelligent Terminal" can be performed by either.

I have used an Anderson Jacobson AD 342A Originate/Answer Acoustic Coupler with great success in this application. However, acoustic couplers can neither provide auto answer nor automatic dialing; this precludes the Remote Console function and places considerable limitations on the PCNET function.

Having decided on an S-100 BUS card implementation over the acoustic coupler approach I then made a comparison between the IDS and D C Hayes boards. The following differences seemed significant to me:

1. Hayes – Uses the Motorola MC14412 Modem chip.
   IDS – Uses the Motorola MC6860L Modem chip.
2. Hayes – Has a 6-pole filter in the receive channel and no filtering in the transmit channel.
   IDS – Has an 8-pole filter in the receive channel and a second 8-pole filter in the transmit channel.
3. Hayes – Has two software selectable BAUD rates, 110 and 300.
   IDS – Has a BAUD rate which is variable under program control from 66 to 600 BAUD.
4. Hayes – Does not provide dial-tone detection.
   IDS – Detects dial tone and sets a corresponding status bit.
5. Hayes – Requires host processor to perform dial pulsing.
   IDS – Provides auto-dialing in hardware.
6. Hayes – Requires CBT be supplied with internal power source.
   IDS – Supplies current source for CBT thus allowing the less expensive model CBT to be used.
7. Hayes – Has no published receiver sensitivity.
   IDS – Provides communication down to −58 dBm.
8. Hayes – Claims to provide for direct connection to telephone lines.
   IDS – Requires a coupler (CBT or CBS).
9. Hayes – Assembled & tested $279.95
   Kit (not available) ----
   IDS – Assembled & tested $395.00
   Kit $245.00

The difference between the two modem chips is considerable. In essence, the MC14412 does not perform the "supervisory" functions that are provided by the MC6860. For example, the 6860 in answer mode detects the ring, causes the phone to go off hook (i.e. answers the phone) places the carrier on the line and detects the remote tone (if present). All this must be done under software control on the 14412. There are many other individual points of difference. The interested individual should get spec sheets on each to make a proper comparison.

A Motorola application note entitled "Low-speed Modem System Design Using the MC6860" states on page 13 that "A complete full duplex system will most likely require operation with input signals down to −50 dBm at the line input. This requires a receive filter network having at least 8 poles. . . and a local transmit filter having 4 poles to provide 25 dB local transmit signal harmonic rejection." Notice that the IDS unit has 8 poles in both transmit and receive. The fact that there is no transmit filter in the Hayes unit probably precludes successful full-duplex operation. While full duplex is not required for the Intelligent Terminal function, I do believe it is necessary for the Remote Console function. The console (LS Dumb Terminal) I am using to generate this article is operating in full duplex. Were it not, I would have difficulty using my TDL Z-TEL text editor. When I enter an ESC character to terminate a string, it echoes a $. When I enter other control characters; e.g., control Y, it echoes Y. In half duplex this would not occur.

The −50 dBm receive level spec mentioned by Motorola agrees with the specs for both the Anderson Jacobson AD 342A Acoustic Coupler mentioned earlier and the BELL 103/113 Modem. On page 3 of a report by VADIC entitled "Telecommunications From The Terminal User's Viewpoint" the statement is made that "to enable 98% of all calls to be received, a modem would have to be capable of demodulating signals as weak as −50 dBm." Note however, that the IDS unit exceeds this by 8 dBm – a further indication of superior design.

The reason for having two 8-pole filters in the IDS unit rather than 8 for TX and 4 for RX is that the filters are not actually identified with TX and RX but with high band and low band. The originator of the call always transmits on the low band–1270 Hz Mark and 1070 Hz space–while the device which answers transmits on the high band–2225 Hz Mark and 2025 Hz space. Thus, each filter is designed for optimum performance for its particular band. In the Hayes unit there is but one filter and it is always in the receive channel. Thus, it must function as a high band filter for originate mode and a low band filter on answerback by component switching. This results in less than optimum performance (even for 6-poles) on either band. To get some quantitative feel for the difference in the two filter designs, the IDS unit employs a total of 32 Op Amps in its filter and detector circuit as compared with 8 Op Amps in the Hayes unit. As you can see, – 58 dBm is not easily achieved.

The two apparent advantages the Hayes unit has over the IDS unit is ability to connect directly into the phone line and price. Needless to say, price is an important consideration. It must be weighed against the probability of success each unit is likely to have in error free communications. This is especially true in long distance communication–meaning any distance requiring more than one telephone switching station to be involved in the connection. Also, considerable variation in local communications can result from differences in line loss and line noise. This variation probably accounts for the mixed results I described earlier for the Pennywhistle when trying to communicate with three different TSCs. In the final analysis, specifications are only indicators and it is the actual performance of a unit that counts. Unfortunately I have not had the opportunity to make any "A B comparisons" between the two units. I can only say that I have been using the IDS unit successfully for the past three months in communicating with the same three TSCs mentioned earlier. I have operated with these TSCs at 110, 150 and 300 BAUD. On April 24 I communicated for the first time with another IDS 88-Modem. The transmission was error free at 300 and 450 BAUD. It was readable but not reliable at 500 BAUD; i.e., about 85% error free. This test was run between Laurel, MD and a location near Dulles Airport in Virginia; a distance of about 35 air miles and undoubtedly involved two switching stations. The direct connection feature of the Hayes unit is discussed in the section on Telco Interface which follows.

If you choose to get the IDS kit as I did, be advised that this is a rather difficult kit to build and should not be undertaken by someone who has not built several others in advance

of it. This does not imply a poor quality board. It is due to the high component density required to put everything on one board. The major difficulty is due to close component placement—raising the possibility of solder bridges—and the fact that most passive components, i.e., resistors, diodes, etc., are mounted vertically. In addition, there are several signal level adjustments which must be made: TX Carrier Level, Dial Tone Detector Gain, Low & High Band Filter Gain, and Threshold Detector Sensitivity. But do not despair, IDS will accept your completed kit and both make the necessary adjustments and check the board out for proper operation. Theoretically, this service is free provided no repairs are necessary. In my case, I was charged just over $30.00 for certain discrepancies. This is still cheaper than the assembled Hayes unit ($275 vs $280). I strongly advise letting IDS do the gain adjustments. When connected up to a data coupler—the gadget that interconnects your modem board to the telephone lines—if your signal level is too high, it is attenuated to the proper level by the coupler. So what, you say. The problem is that the received signal will be attenuated by an equal amount.

For further information on the IDS 88-Modem contact:

International Data Systems, Inc
400 North Washington St.
Suite 200
Falls Church, VA 22046
(703) 536-7373

Telco Interface & Protocol

Once you have the IDS 88-Modem board up and running, it is necessary to interface it to the telephone system. The conventional way to do this is to install a data coupler between the modem board and the telephone line. The data coupler—also known as a Data Access Arrangement (DAA)—comes in two forms: CBT and CBS. The CBT uses DC current loop control while the CBS uses RS232 voltage levels for the same purpose. The CBT is less expensive than the CBS but requires an external power source. This source is supplied by the 88-Modem board. Also, the CBT can be purchased for $125.00 from IDS. Whether you buy the device or rent it from Telco ($4.00 to $5.50 per month) the CBT is usually recommended unless a CBS device is already in place.

The reason the data coupler is necessary is that it provides for ring detection (necessary for auto-answer) and off hook control (the equivalent of picking up the handset) in addition to DC isolation, impedance matching and surge protection. As of June 1, 1977, non-Bell modems may be directly connected to the phone line without a requirement for a Telco-provided DAA. However, in order to do this and still provide for auto answer capability the system must, at a minimum, provide for ring detection (the IDS unit provides this while Hayes does not). Additionally, proper impedance matching and signal level requirements must be met. If this is done then the manufacturer must get his device approved and certified. Any manufacturer who has done so is given an FCC authorization code and REN number. Since D C Hayes does not give this information and since both the telephone company and FCC regulations prohibit direct connection of an unapproved device, I would be very cautious about this. Note also that the DAA not only protects the switched network from the modem, it also provides protection to the user from electrical surges that routinely occur on the telephone lines due to lightning. Direct connection does not offer this protection and resulting risk is high that the modem board and/or the computer of which it is a part will be damaged.

I can't give you much help when it comes to CBT installation as I rent mine and it was installed by Telco. As I had no one to advise me in this regard, I decided to play it safe. Telco will want to know what options you want installed with your CBT. The suggested options are given in the manual which accompanies the IDS modem. Associated with the options list are some symbols which have meaning only to Telco but using these symbols for brevity I would say that options "F-12" and "7.3" are of questionable value and that "G-14" should actually read "G-13". If you are timid, as I was, and want Telco to install the coupler, I believe they will do the installation even if you purchase the CBT from IDS.

In the Automatic Answer mode the combination of software and modem hardware must supply the logic necessary to answer a call. In support of this the following sequence of events take place:

1. Idle — The system has been initialized and awaits call.
2. RI-on — The telephone rings.
3. OH-on — The modem commands the CBT to go "off hook".
4. DA-on — The modem places the CBT in the data mode.
5. Delay — 1 to 3 second delay before CCT. This allows the automatic billing equipment to register the call.
6. CCT-on — Indication that the connection has been established.
7. **** — Normal data communications is performed.
8. OH-off — Terminate the call and break the connection.
9. DA-off — Leave data mode.
10. CCT-off

To originate a call automatically the CBT must be commanded to go "off hook", detect dial tone and generate the dial pulses corresponding to the phone number being dialed. When the dial tone is detected, the DA lead must be turned off and after CCT goes off, the OH lead is pulsed to produce the dialing function. Thus, the following sequence of events occur when in Originate Mode:

1. Idle — The system has been initialized in preparation for this mode.
2. OH-on — Go "off hook".
3. DA-on — Enter data mode.
4. CCT-on — Indicates connection has been established.
5. **** — Wait until dial tone has been detected by modem.
6. DA-off — Data mode must be off to enable dialing.
7. CCT-off
8. OH-pulsed (perform dialing operation).
9. DA-on — Prepare for normal data transfer (DAta mode).
10. CCT-on — Indicates connection for normal data transfer has been established.
11. **** — Normal data interchange.
12. OH-off — Terminate communications.
13. DA-off — Discontinue data mode.
14. CCT-off — (the phone has been hung up and idle state restored).

The normal dial pulse rate is 10pps but the pulse rate is variable under software control. I have been successfully employing a dial pulse rate of 20pps. This is almost as fast as "touch tone" dialing and at this rate the average 7 digit number can be dialed in under three seconds.

88-Modem Board Function & Interface

The 88-Modem board can be broken down into 5 major functions as depicted in the block diagram of the device. The UART function is provided by a General Instruments AY-3-1014/1015 UART chip. The MODEM function is provided by the Motorola MC6860L chip. The DIALER function consists primarily of four 74192 DECADE UP/DOWN COUNTERS. The remainder of the functions employ a variety of devices.

The 88-Modem requires 4 consecutive I/O port addresses. Following the suggestion of the manufacturer, I have assigned them the values of 0C0H thru 0C3H. They may be changed by using the DIP switch provided on the board. Due primarily to the sophistication of the MC6860L most of the TELCO interface activity described earlier is handled automatically by the hardware.

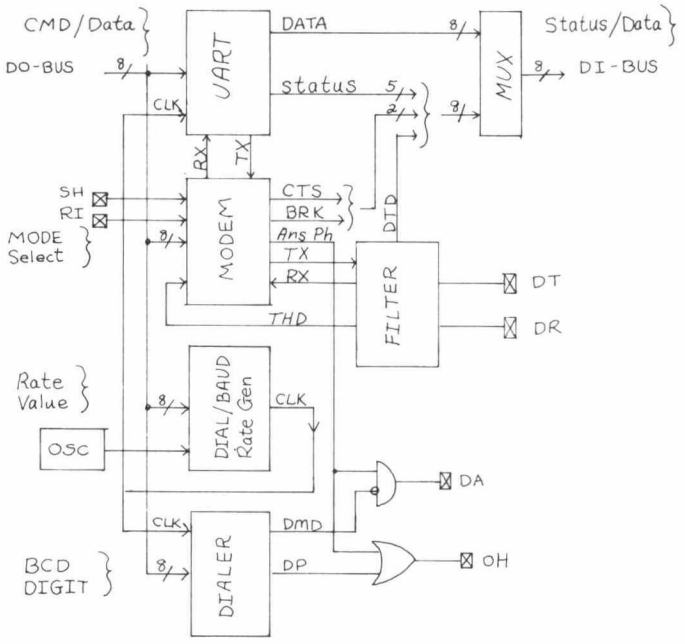
The following sequence illustrates the principal steps required to initiate, perform, and terminate the Answer Mode:

1. Send a mode command to the UART establishing the number of data bits, stop bits (1 or 2), and parity (even or odd).
2. Load the BAUD rate value (8 bits) into the Rate Generator.
3. Send a mode command to the MODEM; i.e., DTR, ≤300 BAUD, BRK-REL, ESD, ELS (mnemonics explained below).
4. Send a second mode command to MODEM as in 3. which removes BRK-REL.
5. Read 88-Modem status and test for Clear to Send (CTS); i.e., the phone rang & the remote carrier was detected.
6. Commence normal communications.
7. Test status for loss of CTS and test data for control-D.
8. If 7. then terminate by resetting DTR in MODEM control byte.

Steps 1. thru 4. above can be considered initialization. Once this is done, the hardware takes over. Referring to the block diagram, the Answer sequence is initiated by the presence of RI (Ring Indicator) at the MODEM input. The MODEM activates the "Ans Ph" (Answer the Phone) which sends OH (Off Hook command) to the CBT. Since we are not in Dial Mode (DMD), the presence of "Ans Ph" also generates DA (DAta mode). The MODEM then places a 2225Hz tone on the TX line. It then awaits a 1270Hz tone from the RX line combined with THD (ThresHold Detect). If this condition continues for the prescribed time, CTS (Clear To Send) is set in the status byte which carries us through step 5. above.

In step 6., data is transmitted to the remote device by loading a data byte into the UART. The byte is shifted serially into the MODEM over the TX line at the prescribed BAUD rate. The MODEM then employs "Frequency Shift Keying" (FSK) to impress the serial data stream onto the carrier; i.e., a binary 1 is transmitted as a 2225Hz tone for the duration of a bit time while a binary 0 appears as a 2025Hz tone for one bit time.

88-Modem Block Diagram



The resetting of DTR (Data Terminal Ready) in step 8. causes "Ans Ph" to go to zero and results in the loss of OH (Off Hook command) thus, "hanging up the phone". A test for control-D is performed in step 7 since this character typed either remotely or at the local console signifies that communications are to be terminated.

In step 3. certain mnemonics were used which will now be defined. "≤300 BAUD" instructs the MODEM that the baud rate will not exceed 300. "BRK-REL" resets the BREAK detect flip-flop. This is a pulsed signal and must therefore be removed in step 4. "ESD" (Enable Space Disconnect) conditions the MODEM to "hang up" if it detects a constant space tone in excess of a predefined period. "ELS" (Enable Long Space) is the opposite of "ESS" (Enable Short Space) and establishes the space duration at which disconnect occurs. SS=0.3 seconds and LS=1.5 seconds.

The Originate Mode is implemented using the following sequence:

1. Load the DIAL rate (20pps) into the Rate Generator.
2. Set the MODEM to Originate mode; i.e., DTR, OH, BRK-REL.
3. Test status for Dial Tone Detect (DTD).
4. If DTD then load first BCD digit to be dialed into DIALER.
5. Test status for Dialer BusY (DBY).
6. If not DBY then load next digit, etc.
7. Load the UART mode—number data bits, etc.
9. Load BAUD rate into Rate Generator.
10. Test status for CTS; i.e., carrier detect.
11. Change MODEM mode from Originate to normal; i.e., remove OH and allow space disconnect.
12. Commence normal communications.
13. Test status for loss of CTS and test data for control-D.
14. If 13. then terminate by resetting DTR.

In step 11. the OH bit in the Modem control word was reset. This does not mean that the connection is broken. Actually,

IDS refers to this bit as SH (switch hook). I call it OH because when set it forces an off hook conditions. However, the hardware will maintain the OH line going to the CBT in the true state provided either that this bit is set or if SH coming from the CBT is true. In the Originate Mode the frequencies used for transmission of data are channeled through the low band filter and are 1270 Hz for a mark (binary 1) and 1070 Hz for a space (binary 0). The received data is channeled through the high band filter with 2225 Hz for a mark and 2025 Hz for a space. This is just the reverse of the Answer Mode.

Program Functions

As previously stated, the program supplied with this article performs the Intelligent Console function. As such, it is designed to establish communications with a large scale computer which allows terminal access. In support of this it provides for both "manual" and "automatic" dialing of the remote facility and accomodates operator selection of one of 6 baud rates. Manual dialing means entering the phone number via the console keyboard while automatic dialing means that certain alpha characters correspond to special phone numbers. When a predefined alpha character is entered in place of the first digit of a phone number, it is replaced by the associated phone number sequence.

Once communication with the TSC has been established the system enters the CHARACTER TX/RX mode. As the name implies, it provides for single character—versus line or block—transfers between the PC and TSC. From this mode one can enter either the FILE TX or the FILE RX modes. These modes provide for the transfer of disk files between the PC and TSC.

The Intelligent Console function requires only Originate Mode operation. In order to check out the Answer Mode and supply a limited version of the Remote Console function the program can be placed in the answer mode. Again the BAUD rate can be selected from the 6 predefined values. When the phone rings, the system answers by placing a 2225 Hz tone on the line and awaits receipt of a 1270 Hz tone in reply. If not received within a specified period, it hangs up. If the tone is detected it sends a sign-on message and enters the CHARACTER TX/RX mode. This allows for communications between the console operator and someone at a conventional terminal; e.g., a teletype unit. It also provides for file transfers to be initiated either remotely or at the system console.

Since it is possible to initiate file transfers remotely, one can load the appropriate floppy disk, set up the system for Answer mode and retrieve the file at a remote system with the PC unattended. The details of system operation are given at the beginning of the program listing.

System Requirements

The program is designed to run under CP/M and except for the I/O peculiar to the 88-Modem board, uses CP/M supplied functions to perform conventional I/O. In order to operate at a given BAUD rate it is necessary that the console device be capable of a least twice that BAUD rate. Thus, if a 30cps printing terminal is used it is necessary to operate at 150 BAUD or less. Unfortunately, this also applies to an on-line printer even if the console is a high speed video display.

A second limitation occurs in file transfers. The data is read or written to the disk in 128 character blocks. This means that by the time the next record is transferred to the disk, the head has unloaded—mine unloads after 8 revolutions without access. Since it takes about 30ms to load the head and additional time for sector seek, two or three characters can be lost per record when operating at 300 BAUD. At 110 BAUD this problem does not exist. One solution to this problem is to use double buffering coupled with interrupt processing. I have purchased and assembled an IMSAI PIC-8 board for this purpose but have not yet had time to write the necessary software. Another approach is to have the TSC cooperate by interrupting data transfer on command from the PC and then continuing when commanded. Unfortunately, many TSCs are not easily made to cooperate in this manner. Of couse, if you are communicating with another PC there is no problem in establishing the necessary protocols to accomplish this.

While on the subject of floppy disk drives, I do not know how well the file transfer will perform at 300 BAUD even given double buffering with interrupt processing if the disk controller does not have a DMA interface. With DMA, the controller is given the buffer address and number of bytes to transfer and it uses cycle steal to effect the memory to disk transfer. The CPU is allowed to handle other processing chores as necessary. Without DMA, the CPU must control the transfer of every data byte between the memory and disk controller. This may load the CPU to the point that it can't handle both disk and modem transfers without losing data at the higher BAUD rates. I have a Digital Systems floppy disk system with DMA controller and have been very pleased with its performance. I have successfully run it under CP/M at 4MHz in an IMSAI chassis using a Cromemco Z-80 processor and Industrial Microsystems Memory boards. The disk controller is available with "dual density". This means it is both hardware and software switchable between single and double density. The system uses Shugart 800 drives. Further information on this system may be had by contacting:

Digital Systems
6017 Margarido Drive
Oakland, CA 94618
(415) 428-0950

Another potential problem has to do with transmitting a file to a TSC. Depending on the front end (communications) processor, it may or may not be able to take data continuously. All systems seem to take a single line, but since they are timeshared they may not be able to get back in time to catch the next line. To help correct this problem I have inserted a "handshake" routine. This is designed to wait for acknowledgment of a given line before transmission of the next line. The current implementation looks for an ETX character before transmitting the next line. In order to determine the appropriate handshake character for a particular TSC a subroutine called SHOW can be used. The normal CHARACTER TX/RX routine screens out all but a select group of control characters. If it is desired to display any received control characters (other than CR and LF) then the "SHOW" subroutine can be patched in. At location 0B62, replace "CALL TCHAR" with "CALL SHOW". SHOW begins at location 08A8. Handshaking is only employed in the FILE TX routine, and only there if specifically requested by the operator during system initialization (see operating instructions).

Perhaps after citing all these potential problems I should be specific about what I have been able to do with my system.

1. CHARACTER TX/RX MODE – I have communicated in both originate and answer modes without error at 110, 150, 300 BAUD. I have communicated at 450 BAUD in answer mode without error.
2. FILE TX – I have transmitted files in both originate and answer modes without error at 110 and 300 BAUD. The 300 BAUD rate was performed without handshake to one TSC and with handshake to another TSC.
3. FILE RX – I have received files in both originate and answer modes without error at 110 BAUD. The transfer is also error free at 300 BAUD but characters (2 or 3) are dropped during the disk writing operation due to head loading delay cited earlier.
4. REMOTE CONSOLE – I have set up a file and put the system in answer mode and then left it unattended. I then called the system, got the sign-on message and initiated a file transmission both at 110 and 300 BAUD.

Thus, it takes a while to learn the "personality" of each different TSC but the system definitely functions.

There is one routine in the program that is system dependent. It is the DELAY subroutine which begins at location 07DA. It outputs null characters to a 300 BAUD printer as a measure of delay time. This can easily be replaces by an equivalent timing loop. As I recall, it is only used to time the duration of a transmitted BREAK. Thus, the program will probably work in all other respects as is.

One area of the program you will want to change is the predefined phone numbers. These are found beginning at location 016E. A rotary dialer is supported for the first 4 phone numbers as detailed in the operating instructions. The last two numbers are good for checkout purposes. They are the TELCO time and weather service numbers. Entering a "W" in place of the first digit to be dialed generates the weather service number while a "Z" gets you the time service.

For those of you who may be new to CP/M there is a users' group headed up by Anthony Gold which is very active and currently has 25 single density floppies full of software. This software is available to all members ($4.00 annual membership fee) for only $8.00 per diskette. I intend to make the source for this program available to the group in the near future. Additional information can be had by contacting:

CP/M Users' Group
164 West 83rd Street
New York, NY 10024

I assume you will want to customize this program to suit your own needs and I believe what has gone before will be of help in doing so. If you are in the Baltimore-Washington area and need some help in checking out transmission at BAUD rates above 300 or would be interested in forming an information exchange group please give me a call.

```
;*****************************************************************
;
;       TITLE: 8B-MODEM DRIVER ROUTINE WITH DISK FILE TRANSFER
;
;       VERSION: 1.00               LAST REVISION: 4/25/78
;
;       AUTHOR: TIM PUGH
;               9655-M HOMESTEAD COURT
;               LAUREL, MD. 20810
;               (301) 776-5253
;
;*****************  SUMMARY OF CONTROL CHARACTERS  *****************
;
;CONTROL CHARACTERS ARE INDICATED BY A "^" PRECEEDING THE CHARACTER.
;
;       ^A      ASSIGN CONSOLE DEVICE
;       ^B      TRANSMIT A "BREAK"
;       ^D      DISCONTINUE CURRENT MODE
;       ^F      FILE NAME SPECIFIED FOR SUBSEQUENT RX/TX
;       ^R      RECEIVE (RX) A DISK FILE FROM A REMOTE DEVICE
;       ^T      TRANSMIT (TX) A DISK FILE TO A REMOTE DEVICE
;       ^W      ASCII ETB -- MUST BE THE LAST CHARACTER OF A TX FILE
;
;*****************  OPERATING INSTRUCTIONS  *****************
;
;THIS PROGRAM IS DESIGNED TO OPERATE UNDER CP/M.  IF A FILE TRANSFER
;IS TO BE PERFORMED, THE FILE NAME SHOULD BE SPECIFIED WHEN THE PROGRAM
;IS CALLED, E.G.       8B-MODEM <FILENAME>.<FILETYPE>
;
;FOLLOWING THE PROGRAM SIGN ON MESSAGE, THE SYSTEM WAITS FOR OPERATOR
;INPUT.  AT THIS POINT THE OPERATOR CAN INITIATE HANDSHAKING (AS DES-
;CRIBED UNDER "FILE TX MODE" BELOW) BY ENTERING AN "H".  ALTERNATIVELY,
;THE OPERATOR CAN SELECT FROM ONE OF 6 BAUD RATES BY ENTERING A "B".
;FINALLY, WE CAN DEFAULT TO 300 BAUD AND NO HAND-SHAKE BY ENTERING
;A "CR" (CARRIAGE RETURN) AS THE INITIAL ENTRY.
;
;NEXT, THE OPERATOR IS DIRECTED TO SELECT THE MODE OF OPERATION BY ENTERING
;EITHER AN "O" FOR ORIGINATE OR AN "A" FOR ANSWER.  IF ANSWER MODE IS SELECTED,
;THE PROGRAM INITIALIZES THE SYSTEM ACCORDINGLY AND WAITS FOR RING DETECT.
;THE PHONE WILL BE ANSWERED ON THE FIRST RING AND GENERATES THE 2225 HZ
;CARRIER SIGNAL.  UPON DETECTION OF THE RESPONDING 1070 HZ TONE, A SIGN-ON
;MESSAGE IS TRANSMITTED TO THE REMOTE DEVICE AND A LOCAL MESSAGE DIRECTING
;THE COMMENCEMENT OF NORMAL TRANSMISSION IS DISPLAYED ON THE CONSOLE.
;OF THE ABOVE MENTIONED CONTROL CHARACTERS, ONLY ^T, ^R, AND ^D ARE
;SUPPORTED AT THE REMOTE DEVICE.  HOWEVER, ALL ARE SUPPORTED AT THE LOCAL
;CONSOLE.  IF THE LOCAL CONSOLE IS UNATTENDED, A FILE TRANSFER CAN BE
;INITIATED REMOTELY PROVIDED THE FILE NAME WAS PREVIOUSLY SPECIFIED.
;COMMUNICATIONS CAN BE TERMINATED REMOTELY SIMPLY BY HANGING UP THE PHONE.
;THIS CAUSES THE SYSTEM TO RETURN TO INITIALIZATION OF ANSWER MODE AND
;AWAIT THE NEXT REMOTE CALL.  ALTERNATIVELY, A "^D" ISSUED REMOTELY WILL
;TERMINATE COMMUNICATIONS AND TAKE THE SYSTEM OUT OF ANSWER MODE.  THIS
;MEANS THAT THE NEXT CALLER WON'T BE GREETED BY A PERHAPS UNEXPECTED
;2225HZ TONE.
;
;IF ORIGINATE MODE IS SELECTED, THE SYSTEM WAITS FOR DIAL TONE DETECTION.
;ONCE THE DIAL TONE HAS BEEN DETECTED - ANNOUNCED ON THE CONSOLE - DIALING
;CAN BEGIN.  CONVENTIONAL DIALING IS PERFORMED BY SIMPLY ENTERING THE
;NUMBER TO BE DIALED AT THE CONSOLE KEYBOARD.  AS EACH NUMBER IS DIALED
;A DASH IS DISPLAYED.  IT IS POSSIBLE TO EXCEED THE DIALING SPEED OF THE
;DEVICE.  THIS IS INDICATED BY THE ABSENCE OF A DASH BETWEEN DIGITS.
;
;BOTH AUTODIALING AND ROTARY DIALING ARE SUPPORTED.  AUTODIALING IS INIT-
;IATED BY ENTERING ONE OF 4 ALPHA CHARACTERS AS THE FIRST "DIGIT" DIALED.
;THE CHARACTERS SUPPORTED ARE "H,M,Z,W".  OTHER CHARACTERS MAY BE SELECTED
;PROVIDED THE 2 LSB'S OF THE ASCII CODE FOR THE SET SELECTED COVER THE
;RANGE OF 00,01,10,11.  THE PHONE NUMBERS ASSIGNED TO THESE CHARACTERS
;CAN BE FOUND IN THE LISTING JUST PRIOR THE THE "MESSAGES" SECTION.
;ROTARY DIALING IS SUPPORTED FOR THE CHARACTER "M" WHEREBY UP TO 4
;NUMBERS WILL BE DIALED IN SEQUENCE IF EACH PRECEEDING NUMBER IS FOUND
;BUSY.  THE SIZE OF THE ROTARY GROUP IS NOT RESTRICTED TO 4.  ANY NUMBER
;GREATER THAN OR EQUAL TO 2 CAN BE SUPPORTED.
;
;AFTER DIALING IS COMPLETED AND THE REMOTE CARRIER HAS BEEN DETECTED,
;THE SYSTEM IS PLACED IN THE "CHARACTER TX/RX MODE".  THIS IS THE MODE
;USED FOR KEYBOARD COMMUNICATIONS WITH THE REMOTE DEVICE.  WHILE IN
;THIS MODE, A BREAK MAY BE TRANSMITTED BY ENTERING A "^B".  IF
;MORE THAN ONE FILE IS TO BE TRANSFERRED, THE NEW FILE NAME CAN BE
;SPECIFIED BY ENTERING A "^F".  CONSOLE MESSAGES WILL PROMPT THE
;OPERATOR FOR APPROPRIATE ENTRIES.  IT IS POSSIBLE TO REASSIGN THE
;CONSOLE DEVICE BY ENTERING A "^A".  HERE TOO, CONSOLE MESSAGES WILL
;PROMPT THE OPERATOR FOR APPROPRIATE ENTRIES.
```

```
;
;IN ORDER TO INITIATE THE "FILE TX MODE" THE USER SHOULD ENTER A "^T".
;THE DATA IS DISPLAYED ON THE CONSOLE DEVICE AS IT IS TRANSMITTED.  UPON
;COMPLETION OF FILE TRANSMISSION, THE SYSTEM REVERTS TO THE CHARACTER
;MODE.  RETURN TO CHARACTER MODE PRIOR TO COMPLETION CAN BE FORCED BY
;ENTERING "^D".  "HAND-SHAKING" AS REQUIRED BY SOME UNIVAC 1100 SERIES
;COMPUTERS HAS BEEN INCLUDED.  TO ENABLE THIS, ENTER A "H" AS THE
;DIALING INITIATION CHARACTER (AS THE CONSOLE MESSAGE DIRECTS).
;THE END OF FILE IN THIS MODE IS "^W".  THUS, THE FILE TO BE TRANS-
;MITTED MUST HAVE A "^W" WRITTEN AS THE LAST CHARACTER ON THE FILE.
;
;TO INITIATE THE "FILE RX MODE" THE USER ENTERS A "^R".  OPERATION IS
;SIMILAR TO FILE TX MODE DESCRIBED ABOVE.  SINCE CHARACTER TRANSMISSION
;IS SUPPORTED IN THIS MODE, THE USER SHOULD SET UP FOR FILE TRANSFER
;BY ISSUEING A COMMAND TO THE REMOTE DEVICE EXCEPT FOR THE CARRIAGE
;RETURN.  TYPE "^R" AND AFTER THE READY MESSAGE IS GIVEN, TYPE "CR".
;THE CR WILL GO TO THE REMOTE DEVICE AND INITIATE TRANSMISSION.
;THIS MODE DOES NOT TERMINATE AUTOMATICALLY.  A "^D" MUST BE TYPED
;IN ORDER TO RETURN TO THE CHARACTER MODE.
;
;ESCAPE FROM THE CHARACTER MODE TO THE INITIAL DIALING SEQUENCE IS
;MADE BY ENTERING A "^D".
;
;RETURN TO CP/M IS EFFECTED BY ENTERING A "^D" WHILE IN THE "INITIAL
;DIALING SEQUENCE MODE".
;
;********************* MODEM CONTROL EQUATES ***************************
                ;
                ;--I/O ADDRESSES
                ;
00C0 =          MODEM0  EQU     0C0H    ;UART STATUS/CONTROL
00C1 =          MODEM1  EQU     0C1H    ;DATA IN/OUT
00C2 =          MODEM2  EQU     0C2H    ;FILTER & CONTROL WORD SELECT
00C3 =          MODEM3  EQU     0C3H    ;(MODEM,DIALER,BAUD-RATE) CONTROL
                ;
                ;--UART/MODEM STATUS (IN MODEM0)
                ;
0001 =          TBE     EQU     001H    ;TRANSMIT BUFFER EMPTY
0002 =          RDA     EQU     002H    ;RECEIVE DATA AVAILABLE
0004 =          DBY     EQU     004H    ;DIALER BUSY
0008 =          PER     EQU     008H    ;PARITY ERROR
0010 =          OVR     EQU     010H    ;DATA OVERRUN
0020 =          BRK     EQU     020H    ;BREAK RECEIVED
0040 =          CTS     EQU     040H    ;CLEAR TO SEND
0080 =          DTD     EQU     080H    ;DIAL TONE DETECT
                ;
                ;--UART MODES (OUT MODEM0)
                ;
0024 =          ASCII1  EQU     024H    ;7 DATA BITS, EVEN PARITY, 1 STOP BIT
0004 =          ASCII2  EQU     004H    ;7 DATA BITS, ODD PARITY, 1 STOP BIT
002C =          BIN1    EQU     02CH    ;8 DATA BITS, EVEN PARITY, 1 STOP BIT
000C =          BIN2    EQU     00CH    ;8 DATA BITS, ODD PARITY, 1 STOP BIT
                ;
                ;--FILTER & CONTROL WORD SELECT CODES (OUT MODEM2)
                ;
0001 =          MMSELO  EQU     001H    ;MODEM MODE CONTROL SELECT - ORIGINATE
0031 =          MMSELA  EQU     031H    ;MODEM MODE CONTROL SELECT - ANSWER
0002 =          DLSEL   EQU     002H    ;DIALER DIGIT CONTROL SELECT
0004 =          BDSEL   EQU     004H    ;BAUD/DIAL RATE CONTROL SELECT
0008 =          INTEN   EQU     008H    ;INTERRUPT ENABLE
                ;
                ;--MODEM MODES (OUT MODEM3)
                ;
00C0 =          MMORIG  EQU     0C0H    ;ORIGINATE: OH.DTR
0043 =          MMLOW   EQU     043H    ;DTR,<=300,ESD,ELS
004B =          MMLREL  EQU     04BH    ;DTR,<=300,BRK-REL,ESD,ELS
0063 =          MMHIGH  EQU     063H    ;DTR,>300,ESD,ELS
006B =          MMHREL  EQU     06BH    ;DTR,>300,BRK-REL,ESD,ELS
000E =          MMTERM  EQU     00EH    ;TERMINATE: BRK-REL,TX-BRK,ESD,ESS
0004 =          MMBRK   EQU     004H    ;TX-BREAK
                ;
                ;--DIAL & BAUD RATE CONSTANTS (OUT MODEM3)
                ;
0066 =          DRATE1  EQU     066H    ;DIAL RATE 10 PPS
0034 =          DRATE2  EQU     034H    ;DIAL RATE 20 PPS
008E =          BAUD10  EQU     08EH    ;110 BAUD
0066 =          BAUD15  EQU     066H    ;150 BAUD
0034 =          BAUD30  EQU     034H    ;300 BAUD
0022 =          BAUD45  EQU     022H    ;450 BAUD
001F =          BAUD50  EQU     01FH    ;500 BAUD
001A =          BAUD60  EQU     01AH    ;600 BAUD
                ;
;********************* CP/M EQUATES ***************************
;
; CP/M FUNCTIONS ARE EXECUTED BY PASSING THE FUNCTION NUMBER IN
; REGISTER C AND THE INFORMATION (ADDRESS OR DATA) IN REGISTERS
; D,E (REGISTER E IF SINGLE BYTE DATA).  THE RESULT (IF APPLICABLE)
; IS RETURNED IN REGISTER A.  IF DOUBLE BYTE RESULT, THE MSB IS IN
; REGISTER B.
;
;--FDOS FUNCTIONS
;
0000 =          RESET   EQU     0       ;SYSTEM RESET
0001 =          RDCON   EQU     1       ;READ CONSOLE CHARACTER
0002 =          WRCON   EQU     2       ;WRITE CONSOLE CHARACTER
0003 =          RDRDR   EQU     3       ;READ CHARACTER FROM READER DEVICE
0004 =          WRPCH   EQU     4       ;WRITE CHARACTER TO PUNCH DEVICE
0005 =          WRLST   EQU     5       ;WRITE CHARACTER TO LIST DEVICE
0007 =          IOCHK   EQU     7       ;READ I/O CONFIGURATION BYTE
0008 =          IOSET   EQU     8       ;WRITE I/O CONFIGURATION BYTE
0009 =          WRCBUF  EQU     9       ;WRITE BUFFER CONTENTS TO CONSOLE
000A =          RDCBUF  EQU     10      ;READ BUFFER CONTENTS FROM CONSOLE
000B =          CONST   EQU     11      ;CONSOLE (DATA AVAILABLE) STATUS
                ;
000C =          UNLOAD  EQU     12      ;UNLOAD DISK R/W HEAD
000D =          DSKRES  EQU     13      ;DISK SYSTEM RESET
000E =          DSKSEL  EQU     14      ;DISK SELECT (0,1,2,3)
000F =          OPEN    EQU     15      ;OPEN DISK FILE (FCB ADDRESS)
0010 =          CLOSE   EQU     16      ;CLOSE DISK FILE  "     "
0011 =          SRCH    EQU     17      ;SEARCH FOR FILE  "     "
0012 =          SRCHN   EQU     18      ;SEARCH FOR NEXT FILE
0013 =          DELETE  EQU     19      ;DELETE DISK FILE (FCB ADDRESS)
0014 =          RDDISK  EQU     20      ;READ NEXT RECORD   "   "
                ;
0015 =          WRDISK  EQU     21      ;WRITE NEXT RECORD
0016 =          CREATE  EQU     22      ;CREATE DISK FILE   "   "
0017 =          RENAME  EQU     23      ;RENAME DISK FILE   "   "
0018 =          LOGALL  EQU     24      ;IDENTIFY ALL ON-LINE DISKS
0019 =          LOGSEL  EQU     25      ;IDENTIFY SELECTED DISK
001A =          SETDMA  EQU     26      ;SET DMA ADDRESS VALUE
                ;
                ;--CP/M ADDRESS CONSTANTS
                ;
0000 =          BOOT    EQU     0       ;WARM START
0005 =          FDOS    EQU     0005H   ;ENTRY TO FDOS
005C =          TFCB    EQU     005CH   ;TRANSIENT FILE CONTROL BLOCK
0080 =          TBUFF   EQU     0080H   ;TRANSIENT BUFFER
                ;
                ;--FCB (FILE CONTROL BLOCK) CONSTANTS
                ;
005C =          FCB     EQU     TFCB
005C =          FCBDN   EQU     FCB+0   ;DISK NUMBER (0,1,2,3)
005D =          FCBFN   EQU     FCB+1   ;FILE NAME (8 BYTES)
0065 =          FCBFT   EQU     FCB+9   ;FILE TYPE (3 BYTES)
0068 =          FCBEX   EQU     FCB+12  ;EXTENT (0 TO 15)
006B =          FCBRC   EQU     FCB+15  ;REC COUNT (0 TO 128) - SIZE OF CURRENT EXTENT
007C =          FCBNR   EQU     FCB+32  ;NEXT RECORD POINTER (0 TO RC)
                ;
;****************** MISCELLANEOUS EQUATES **************************
                ;
                ;--ASCII CONTROL CODES
                ;
0000 =          NUL     EQU     000H    ;(^@) NULL (TIME FILL)
0001 =          SOH     EQU     001H    ;(^A) START OF HEADING
0002 =          STX     EQU     002H    ;(^B) START TEXT
0003 =          ETX     EQU     003H    ;(^C) END TEXT
0004 =          EOT     EQU     004H    ;(^D) END OF TRANSMISSION
0005 =          ENQ     EQU     005H    ;(^E) ENQUIRY - "WHO ARE YOU"
0006 =          ACK     EQU     006H    ;(^F) ACKNOWLEDGE - "YES"
0007 =          BEL     EQU     007H    ;(^G) BELL - HUMAN ATTENTION REQUIRED
0008 =          BS      EQU     008H    ;(^H) BACKSPACE
0009 =          HT      EQU     009H    ;(^I) HORIZONTAL TABULATION
000A =          LF      EQU     00AH    ;(^J) LINE FEED
000B =          VT      EQU     00BH    ;(^K) VERTICAL TABULATION
000C =          FF      EQU     00CH    ;(^L) FORM FEED
000D =          CR      EQU     00DH    ;(^M) CARRIAGE RETURN
000E =          SO      EQU     00EH    ;(^N) SHIFT OUT - NONSTANDARD CODE FOLLOWS
000F =          SI      EQU     00FH    ;(^O) SHIFT IN - RETURN TO STANDARD CODE
                ;
0010 =          DLE     EQU     010H    ;(^P) DATA LINK EXCAPE - CHANGE DATA COM CNTRL
0011 =          DC1     EQU     011H    ;(^Q) DEVICE CONTROL 1
0012 =          DC2     EQU     012H    ;(^R)   "       "    2
0013 =          DC3     EQU     013H    ;(^S)   "       "    3
0014 =          DC4     EQU     014H    ;(^T) DEVICE CONTROL 4
0015 =          NAK     EQU     015H    ;(^U) NEGATIVE ACKNOWLEDGE - "NO"
```

```
0016 =          SYN     EQU     016H    ;(^V) SYNCHRONOUS IDLE - ENABLES RESYNC
0017 =          ETB     EQU     017H    ;(^W) END TRANSMISSION BLOCK
0018 =          CAN     EQU     018H    ;(^X) CANCEL PREVIOUS DATA
0019 =          EM      EQU     019H    ;(^Y) END MEDIUM
001A =          CLR     EQU     01AH    ;(^Z) CLEAR SCREEN ON LS-TERMINAL (ASCII: SUB)
001B =          ESC     EQU     01BH    ;(^[) ESCAPE - FOR CODE EXTENSION
001C =          FS      EQU     01CH    ;(^\) FILE SEPARATOR
001D =          GS      EQU     01DH    ;(^]) GROUP     "
001E =          RS      EQU     01EH    ;(^^) RECORD    "
001F =          US      EQU     01FH    ;(^_) UNIT      "
                ;
007F =          DEL     EQU     07FH    ;DELETE (RUBOUTS)
                ;
                ;--ZAPPLE I/O CONFIGURATION
                ;
F015 =          ZIOCHK  EQU     0F015H
F018 =          ZIOSET  EQU     0F018H
                ;
                ;################### PROGRAM STATEMENTS ##########################
                ;
0100                    ORG     100H
0100 C30009             JMP     START
0103                    DS      40      ;STACK AREA
012B            STACK:  DS      2       ;STACK POINTER
012D            MMSEL:  DS      A       ;MODEM MODE SELECT (ORIGINATE OR ANSWER)
0134            MMODE:  DS      1       ;MODEM MODE
0135            CHCNT:  DS      1       ;CHARACTER POSITION COUNT
0136            BUFPTR: DS      1       ;DATA BUFFER POINTER
0137            HSHAKE: DS      1       ;HAND-SHAKE FLAG
0138            AUTOD:  DS      2       ;AUTO-DIAL FLAG
013A            ROTFLG: DS      1       ;ROTARY DIALING FLAG
013B 01         ANSFLG: DB      1       ;ANSWER MODE FLAG
013C 01         LFDFLG  DB      1       ;LINE FEED FLAG
013D 2020202020BLANKS: DB       '               ' ;16 BLANKS
                ;
                ;--CONSOLE DATA BUFFER
014D 10         CBUFF   DB      16      ;BUFFER SIZE
014E            CBCNT   DS      1       ;COUNT OF CHARACTERS READ
014F            CBTXT   DS      16      ;CONSOLE DATA
                ;
                BDTBL:  ;BAUD RATE VALUE TABLE
015F 8E                 DB      BAUD10  ;110 BAUD
0160 68                 DB      BAUD15  ;150 BAUD
0161 34                 DB      BAUD30  ;300 BAUD
0162 22                 DB      BAUD45  ;450 BAUD
0163 1F                 DB      BAUD50  ;500 BAUD
0164 1A                 DB      BAUD60  ;600 BAUD
0165            BDVAL:  DS      1       ;SELECTED BAUD RATE VALUE
                ;
                ;************************ PHONE NUMBERS ******************************
                ;
                ADRTBL: ;PHONE NUMBER ADDRESS TABLE
0166 6F01               DW      PHONEH
0168 6E01               DW      PHONEM
016A 9701               DW      PHONEZ
016C 9F01               DW      PHONEW
                ;
                PHONEM: ;UNIV. OF MD.
016E 3435343438         DB      '4544821$'
0176 3435343532         DB      '4545201$'
017E 3435343534         DB      '4545401$'
0186 3435343534         DB      '4545405$'
018E 1D                 DB      GS      ;GROUP SEPERATOR
018F 3737363532PHONEH:  DB      '7765253$';HOME PHONE
0197 3834343132PHONEZ:  DB      '8441212$';TIME SERVICE
019F 3933363132PHONEW:  DB      '9361212$';WEATHER SERVICE
                ;
                ;************************ MESSAGES ***********************************
                ;
01A7 1A2020202020MSG1   DB      CLR,'                88-MODEM DRIVER ROUTINE'
01CD 0D0A0A0A           DB      CR,LF,LF,LF
01D1 544F20494E         DB      'TO INITIALIZE THE SYSTEM, TYPE ONE OF THE FOLLOWING '
0205 4348415241         DB      'CHARACTERS:',CR,LF,LF
0213 4820284841         DB      'H (HAND-SHAKE INIT.), B (BAUD-RATE SEL.), '
023D 4352202844         DB      'CR (DEFAULT TO 300 BAUD)',CR,LF,LF,LF,BEL,'$'
025B 535045434349MSG1A  DB      'SPECIFY THE MODE BY TYPING THE APPROPRIATE CHARACTER '
0290 0D0A0A4120         DB      CR,LF,LF,'A --> ANSWER-MODE    O --> ORIGINATE-MODE'
02BC 0D0A0A24           DB      CR,LF,LF,'$'
02C0 1A2020202020MSG1B  DB      CLR,'          ORIGINATE MODE -- WAITING FOR DIALTONE'
02F0 0D0A0A24           DB      CR,LF,LF,'$'
02F4 074449414CMSG2     DB      BEL,'DIAL TONE RECEIVED:  COMMENSE DIALING - '
031D 5445524D49         DB      'TERMINATED BY A CR',CR,LF,LF,'$'
0333 0A57414954MSG2A    DB      LF,'WAITING FOR CARRIER DETECT',CR,LF,'$'
0351 0A0A434F4EMSG3     DB      LF,LF,'CONNECTION HAS BEEN BROKEN',CR,LF,'$'
0370 1A07434F4EMSG4     DB      CLR,BEL,'CONNECTION HAS BEEN ESTABLISHED, BEGIN '
0399 4E4F524D41         DB      'NORMAL TRANSMISSION',CR,LF,LF,LF,'$'
03B1 1A07090952MSG5     DB      CLR,BEL,'                RETURN TO CP/M EXECUTION'
03CD 0D0A0A0A24         DB      CR,LF,LF,LF,'$'
03D2 07554E4142MSG6     DB      BEL,'UNABLE TO OPEN FILE FOR TRANSMISSION',CR,LF,'$'
03FA 074449534BMSG7     DB      BEL,'DISK READ ERROR',CR,LF,'$'
040D 07554E4142MSG8     DB      BEL,'UNABLE TO CREATE FILE DUE TO INSUFFICIENT'
0437 2044495245         DB      ' DIRECTORY SPACE',CR,LF,'$'
044A 0746494C45MSG9     DB      BEL,'FILE TRANSMISSION COMPLETED',CR,LF,LF,LF,'$'
046B 0D0A0A0A52MSG10    DB      CR,LF,LF,LF,'READY TO RECEIVE FILE',CR,LF,LF,LF,'$'
0489 07554E4142MSG11    DB      BEL,'UNABLE TO CLOSE FILE',CR,LF,'$'
04A1 074552524FMSG12    DB      BEL,'ERROR IN EXTENDING FILE',CR,LF,'$'
04BC 074449534BMSG13    DB      BEL,'DISK IS FULL',CR,LF,'$'
04CC 074449534BMSG14    DB      BEL,'DISK DIRECTORY IS FULL',CR,LF,'$'
04E6 202048414EMSG15    DB      '  HAND-SHAKE FLAG IS SET',CR,LF,LF,'$'
0502 2020424155MSG16    DB      '  BAUD-RATE SELECTION',CR,LF,LF
051A 53454C4543         DB      'SELECT A BAUD RATE BY ENTERING APPROPRIATE DIGIT '
054B 46524F4D20         DB      'FROM THE FOLLOWING LIST:',CR,LF
0565 3120283131         DB      '1 (110 BAUD), 2 (150 BAUD), 3 (300 BAUD)',CR,LF
058F 3420283435         DB      '4 (450 BAUD), 5 (500 BAUD), 6 (600 BAUD)',CR,LF,LF,'$'
05BB 1A2020202020MSG17  DB      CLR,'                          ANSWER MODE',CR,LF,LF,'$'
05E2 5741495449MSG18    DB      'WAITING FOR RING-INDICATOR AND CARRIER-DETECT'
060F 0D0A0A24           DB      CR,LF,LF,'$'
0613 0D0A0A2020MSG19    DB      CR,LF,LF,'                    HELLO THERE',CR,LF,LF
0638 2020202020         DB      '          PLEASE BEGIN YOUR TRANSMISSION',CR,LF,LF,'$'
0664 0D0A0A454EMSG20    DB      CR,LF,LF,'ENTER FILE NAME IN 2 PARTS AS FOLLOWS:',CR,LF
068F 3C46494C45         DB      '<FILENAME> CR',CR,LF
069E 3C46494C45         DB      '<FILETYPE> CR',CR,LF,LF,'>$'
06B0 0D0A0A5345MSG21    DB      CR,LF,LF,'SELECT THE CONSOLE DEVICE FROM THE '
06D6 464F4C4C4F         DB      'FOLLOWING LIST:',CR,LF
06E7 3020285454         DB      '0 (TTY), 1 (CRT), 2 (BATCH), 3 (MODEM)',CR,LF,LF,'$'
                ;
                ;************************ SUBROUTINES ********************************
                ;
                ;+++++++++++++++++++++ COMMON SUBROUTINES ++++++++++++++++++++++++++
                ;
                EXIT:   ;RETURN TO CP/M
0711 0E09               MVI     C,WRCBUF
0713 11B103             LXI     D,MSG5
0716 CD0500             CALL    FDOS    ;TYPE EXIT MESSAGE
0719 AF                 XRA     A       ;CLEAR A & CARRY
071A 2A2B01             LHLD    STACK
071D F9                 SPHL            ;RESTORE CP/M STACK POINTER
071E C9                 RET
                ;
                CSTAT:  ;READ CONSOLE STATUS
071F 0E0B               MVI     C,CONST
0721 CD0500             CALL    FDOS
0724 B7                 ORA     A
0725 C9                 RET
                ;
                CRD$ST: ;CONSOLE READ - GO TO "STOP" IF "^D"
0726 0E01               MVI     C,RDCON
0728 CD0500             CALL    FDOS
072B FE04               CPI     EOT
072D C0                 RNZ             ;"^D" ?
072E F1                 POP     PSW     ;  YES, CORRECT STACK FOR NON-RETURN
072F C3320D             JMP     STOP
                ;
                MSTAT:  ;READ MODEM STATUS AND DISPLAY ON PORT OFFH
0732 DBC0               IN      MODEMO
0734 2F                 CMA
0735 D3FF               OUT     0FFH
0737 2F                 CMA
0738 C9                 RET
                ;
                MTBE:   ;MODEM TX BUFFER EMPTY TEST
0739 CD3207             CALL    MSTAT
073C E601               ANI     TBE
073E CA3907             JZ      MTBE
0741 C9                 RET
                ;
                TCHAR:  ;TYPE A CHARACTER ON CONSOLE DEVICE
0742 0E02               MVI     C,WRCON
0744 5F                 MOV     E,A
0745 CD0500             CALL    FDOS
0748 C9                 RET
```

```
                        ;
                DCHAR:  ;DELETE A CHARACTER ON THE CONSOLE
0749 3E08               MVI     A,BS
074B CD4207             CALL    TCHAR
074E 3E20               MVI     A,' '
0750 CD4207             CALL    TCHAR
0753 3E08               MVI     A,BS
0755 CD4207             CALL    TCHAR
0758 3E07               MVI     A,BEL
075A CD4207             CALL    TCHAR
075D C3380A             JMP     DIAL
                        ;
0760 3E0D       CRLF:   MVI     A,CR
0762 CD4207             CALL    TCHAR
0765 3E0A               MVI     A,LF
0767 CD4207             CALL    TCHAR
076A C9                 RET
                        ;
                THEX:   ;TYPE THE PAIR OF HEX CHARACTERS IN REG A
076B F5                 PUSH    PSW
076C 1F                 RAR
076D 1F                 RAR
076E 1F                 RAR
076F 1F                 RAR
0770 CD7407             CALL    TNIB
0773 F1                 POP     PSW
0774 E60F       TNIB:   ANI     0FH
0776 FE0A               CPI     10
0778 DA7D07             JC      ISNUM
077B C607               ADI     7
077D C630       ISNUM:  ADI     '0'
077F C34207             JMP     TCHAR
                        ;
                DIALIT: ;DIAL THE DIGIT IN REG A
0782 E60F               ANI     00FH
0784 D3C3               OUT     MODEM3  ;SEND DIGIT TO DIALER
0786 CD1F07     DBUSY:  CALL    CSTAT   ;GET CONSOLE STATUS
0789 CA8F07             JZ      D2      ;DATA AVAILABLE?
078C CD2607             CALL    CRD$ST  ;  YES, TEST FOR "^D"
078F CD3207     D2:     CALL    MSTAT
0792 E604               ANI     DBY
0794 C28607             JNZ     DBUSY   ;BUSY?
0797 3E2D               MVI     A,'-'   ;  NO
0799 CD4207             CALL    TCHAR   ;PROMPT FOR NEXT DIGIT
079C C9                 RET
                        ;
                UART:   ;SET UP UART CONFIGURATION
079D 3E24               MVI     A,ASCII1
079F D3C0               OUT     MODEM0  ;SELECT THE NO. OF DATA BITS & PARITY TYPE
07A1 3E04               MVI     A,BDSEL
07A3 D3C2               OUT     MODEM2
07A5 3A0501             LDA     BDVAL
07A8 D3C3               OUT     MODEM3  ;SET THE BAUD-RATE VALUE
07AA C9                 RET
                        ;
                MDM:    ;SET UP MODEM CONFIGURATION
07AB 3A2D01             LDA     MMSEL
07AE D3C2               OUT     MODEM2
07B0 3A3401             LDA     MMODE
07B3 D3C3               OUT     MODEM3
07B5 000000             NOP!    NOP!    NOP
07B8 000000             NOP!    NOP!    NOP
07BB E6F7               ANI     0F7H    ;RESET THE "BREAK-RLS." BIT
07BD D3C3               OUT     MODEM3
07BF C9                 RET
                        ;
                        ;+++++++++++++++++ CHARACTER TX/RX SUBROUTINES ++++++++++++++++++++++
                        ;
                BREAK:  ;TRANSMIT A BREAK TO REMOTE DEVICE
07C0 3A2D01             LDA     MMSEL
07C3 D3C2               OUT     MODEM2
07C5 3A3401             LDA     MMODE
07C8 B4                 ORA     MMBRK
07C9 D3C3               OUT     MODEM3  ;TRANSMIT A BREAK
07CB 060F               MVI     B,15
07CD CDDA07             CALL    DELAY   ;INSERT 0.5 SEC DELAY
07D0 3A3401             LDA     MMODE
07D3 E6F3               ANI     0F3H
07D5 D3C3               OUT     MODEM3  ;RESET BREAK & BREAK.RLS BITS
07D7 C3510B             JMP     TRCHAR
```

```
                        ;
                DELAY:  ;INSERT DELAY BY OUTPUT OF NULLS TO 300 BAUD LIST DEVICE
07DA C5                 PUSH    B       ;B.REG CONTAINS DELAY CHARACTER COUNT
07DB 0E05               MVI     C,WRLST
07DD 1E00               MVI     E,NUL
07DF CD0500             CALL    FDOS    ;SEND NULL TO 300 BAUD PRINTER
07E2 C1                 POP     B
07E3 05                 DCR     B
07E4 C2DA07             JNZ     DELAY   ;DELAY COMPLETE?
07E7 C9                 RET
                        ;
                MOVE:   ;SOURCE.ADDR=H,L   DEST.ADDR=D,E   BYTE.COUNT=A
07E8 B7                 ORA     A
07E9 C8                 RZ              ;BYTE.COUNT=0 ?
07EA 47                 MOV     B,A     ;  NO
07EB 7E         MLOOP:  MOV     A,M
07EC EB                 XCHG
07ED 77                 MOV     M,A
07EE EB                 XCHG
07EF 13                 INX     D
07F0 23                 INX     H
07F1 05                 DCR     B
07F2 C2EB07             JNZ     MLOOP
07F5 C9                 RET
                        ;
                FILE:   ;SET UP FCB (FILE.CONTROL.BLOCK) AS SPECIFIED BY OPERATOR
07F6 3E0B               MVI     A,11
07F8 213D01             LXI     H,BLANKS
07FB 115D00             LXI     D,FCBFN
07FE CDE807             CALL    MOVE    ;BLANK FILL FCB <FILENAME> <FILETYPE>
0801 0E09               MVI     C,WRCBUF
0803 116406             LXI     D,MSG20
0806 CD0500             CALL    FDOS    ;REQUEST <FILENAME> & <FILETYPE> FROM OPERATOR
0809 0E0A               MVI     C,RDCBUF
080B 114D01             LXI     D,CBUFF
080E CD0500             CALL    FDOS    ;GET <FILENAME> FROM CONSOLE
0811 3A4E01             LDA     CBCNT
0814 214F01             LXI     H,CBTXT
0817 115D00             LXI     D,FCBFN
081A CDE807             CALL    MOVE    ;WRITE FCB <FILENAME>
081D CD6007             CALL    CRLF
0820 3E3E               MVI     A,'>'
0822 CD4207             CALL    TCHAR   ;PROMPT OPERATOR FOR <FILETYPE>
0825 0E0A               MVI     C,RDCBUF
0827 114D01             LXI     D,CBUFF
082A CD0500             CALL    FDOS    ;GET <FILETYPE> FROM CONSOLE
082D 3A4E01             LDA     CBCNT
0830 214F01             LXI     H,CBTXT
0833 115D00             LXI     D,FCBFT
0836 CDE807             CALL    MOVE    ;WRITE FCB <FILETYPE>
0839 CD6007             CALL    CRLF
083C C3510B             JMP     TRCHAR
                        ;
                ASSIGN: ;MAKE CONSOLE ASSIGNMENT (TEMPORARILY LIMITED TO TTY & CRT)
083F 0E09               MVI     C,WRCBUF
0841 11B006             LXI     D,MSG21
0844 CD0500             CALL    FDOS    ;PROMPT FOR CONSOLE SELECTION
0847 0E01               MVI     C,RDCON
0849 CD0500             CALL    FDOS    ;READ SELECTION
084C F5                 PUSH    PSW
084D CD6007             CALL    CRLF
0850 F1                 POP     PSW
0851 E601               ANI     01H     ;TEMPORARILY ALLOWS FOR TTY & CRT ONLY
0853 F5                 PUSH    PSW     ;SAVE THE SELECTION
0854 CD15F0             CALL    ZIOCHK
0857 E6FC               ANI     0FCH    ;MASK OUT THE 2 LSBS'
0859 47                 MOV     B,A
085A F1                 POP     PSW     ;GET THE SELECTION
085B B0                 ORA     B
085C 4F                 MOV     C,A
085D CD18F0             CALL    ZIOSET
0860 C3510B             JMP     TRCHAR
                        ;
                        ;++++++++++++++++++++ FILE IX SUBROUTINES ++++++++++++++++++++++++
                        ;
                OPENF:  ;OPEN DISK FILE FOR READING
0863 0E0F               MVI     C,OPEN
0865 115C00             LXI     D,FCB
0868 CD0500             CALL    FDOS
086B FEFF               CPI     255
086D C27C08             JNZ     OPNOK   ;IS FILE IN DIRECTORY?
0870 0E09               MVI     C,WRCBUF ;  NO
0872 11D203             LXI     D,MSG6
```

```
0875 CD0500             CALL    FDOS    ;TYPE ERROR MESSAGE
0878 F1                 POP     PSW     ;CORRECT STACK PTR FOR NON-RETURN
0879 C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
087C AF         OPNOK:  XRA     A
087D 327C00             STA     FCBNR   ;NEXT-RECORD = 0
0880 C9                 RET
                ;
                DISKR:  ;READ A DISK SECTOR (RECORD)
0881 0E14               MVI     C,RDDISK
0883 115C00             LXI     D,FCB
0886 CD0500             CALL    FDOS
0889 FE00               CPI     0
088B C8                 RZ              ;NORMAL READ?
088C FE01               CPI     1       ;  NO
088E C29C08             JNZ     DRERR   ;END OF FILE?
0891 0E09               MVI     C,WRCBUF ;  YES
0893 114A04             LXI     D,MSG9
0896 CD0500             CALL    FDOS    ;TYPE "END FILE" MSG
0899 C3A408             JMP     DRERRX
089C 0E09       DRERR:  MVI     C,WRCBUF
089E 11FA03             LXI     D,MSG7
08A1 CD0500             CALL    FDOS    ;TYPE "READ ERR" MSG
08A4 F1         DRERRX: POP     PSW     ;CORRECT STACK FOR NON-RETURN
08A5 C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
                ;
                SHOW:   ;SHOW CHAR TRANSMITTED OR RECEIVED
08A8 FE0A               CPI     LF
08AA CA4207             JZ      TCHAR
08AD FE0D               CPI     CR
08AF CA4207             JZ      TCHAR
08B2 FE20               CPI     ' '
08B4 DABC08             JC      SHEX    ;ASCII CONTROL CHARACTER?
08B7 FE7F               CPI     07FH    ;  NO
08B9 DA4207             JC      TCHAR   ;ASCII PRINTABLE CHARACTER?
                SHEX:   ;SHOW HEX VALUE FOR NON-PRINTABLE CHARACTER
08BC F5                 PUSH    PSW     ;  NO
08BD 3E28               MVI     A,'('
08BF CD4207             CALL    TCHAR
08C2 F1                 POP     PSW
08C3 CD6B07             CALL    THEX
08C6 3E29               MVI     A,')'
08C8 CD4207             CALL    TCHAR
08CB C9                 RET
                ;
                ;********************** FILE RX SUBROUTINES ************************
                ;
                MAKEF:  ;CREATE A FILE FOR SUBSEQUENT WRITING
08CC 0E16               MVI     C,CREATE
08CE 115C00             LXI     D,FCB
08D1 CD0500             CALL    FDOS
08D4 FEFF               CPI     255
08D6 C2E508             JNZ     MAKEOK  ;IS SUFFICIENT DIRECTORY SPACE AVAILABLE?
08D9 0E09               MVI     C,WRCBUF ;  NO
08DB 110D04             LXI     D,MSG8
08DE CD0500             CALL    FDOS    ;TYPE ERROR MSG.
08E1 F1                 POP     PSW     ;CORRECT STACK FOR NON-RETURN
08E2 C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
08E5 AF         MAKEOK: XRA     A
08E6 327C00             STA     FCBNR   ;NEXT-RECORD = 0
08E9 C9                 RET
                ;
                ST$BUF: ;STORE CONTENTS OF REG A IN DISK BUFFER
08EA F5                 PUSH    PSW
08EB 3A3601             LDA     BUFPTR
08EE 5F                 MOV     E,A     ;D,E = DATA BYTE ADDRESS
08EF 1600               MVI     D,0
08F1 3C                 INR     A
08F2 323601             STA     BUFPTR  ;INCREMENTED BUFFER POINTER
                ;--STORE A BYTE A ADDRESS: TBUFF+(D,E)
08F5 218000             LXI     H,TBUFF
08F8 19                 DAD     D
08F9 F1                 POP     PSW
08FA 77                 MOV     M,A
                ;--TEST FOR BUFFER FULL
08FB 3A3601             LDA     BUFPTR
08FE FE80               CPI     080H
0900 C0                 RNZ             ;IS BUFFER FULL?
0901 F1                 POP     PSW     ;YES, CORRECT STACK FOR NON-RETURN
0902 C3120D             JMP     RFULL
                ;
                DISKW:  ;WRITE A DISK SECTOR (RECORD)
0905 0E15               MVI     C,WRDISK
0907 115C00             LXI     D,FCB
090A CD0500             CALL    FDOS
090D FE00               CPI     0
090F C8                 RZ              ;NORMAL WRITE?
0910 FE01               CPI     1       ;  NO
0912 C22009             JNZ     DR1     ;ERROR IN EXTENDING FILE?
0915 0E09               MVI     C,WRCBUF ;  YES
0917 11A104             LXI     D,MSG12
091A CD0500             CALL    FDOS    ;TYPE "EXTEND ERROR" MSG
091D C33809             JMP     DWERRX
0920 FE02       DR1:    CPI     2
0922 C23009             JNZ     DR2     ;IS DISK FULL?
0925 0E09               MVI     C,WRCBUF ;  YES
0927 11BC04             LXI     D,MSG13
092A CD0500             CALL    FDOS    ;TYPE "DISK FULL" MSG
092D C33809             JMP     DWERRX
0930 0E09       DR2:    MVI     C,WRCBUF
0932 11CC04             LXI     D,MSG14
0935 CD0500             CALL    FDOS    ;TYPE "DIRECTORY FULL" MSG
0938 F1         DWERRX: POP     PSW     ;CORRECT STACK FOR NON-RETURN
0939 C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
                ;
                CLOSEF: ;CLOSE DISK FILE (WRITING COMPLETED)
093C 0E10               MVI     C,CLOSE
093E 115C00             LXI     D,FCB
0941 CD0500             CALL    FDOS
0944 FEFF               CPI     255
0946 C0                 RNZ             ;SUCESSFUL CLOSE?
0947 0E09               MVI     C,WRCBUF ;  NO
0949 118904             LXI     D,MSG11
094C CD0500             CALL    FDOS    ;TYPE ERROR MESSAGE
094F C9                 RET
                ;
                ;********************** MAIN PROGRAM *****************************
                ;
                START:  ;INITIALIZE OUR STACK
0950 210000             LXI     H,0
0953 39                 DAD     SP      ;HL=STACK FROM CP/M
0954 222B01             SHLD    STACK   ;SAVE CP/M STACK POINTER
0957 312B01             LXI     SP,STACK ;LOAD OUR STACK PIONTER
095A AF         RESTRT: XRA     A
095B 323701             STA     HSHAKE  ;RESET HAND-SHAKE FLAG
095E 323801             STA     AUTOD   ;RESET AUTO-DIAL FLAG
0961 323A01             STA     ROTFLG  ;RESET ROTARY DIALING FLAG
0964 323B01             STA     ANSFLG  ;RESET ANSWER MODE FLAG
0967 323C01             STA     LFDFLG  ;RESET LINE FEED FLAG
096A 3E34               MVI     A,BAUD30
096C 326501             STA     BDVAL   ;DEFAULT BAUD RATE VALUE = 300
096F 3E4B               MVI     A,MMLREL
0971 323401             STA     MMODE   ;DEFAULT MODEM MODE = MMLREL
0974 0E09               MVI     C,WRCBUF
0976 11A701             LXI     D,MSG1
0979 CD0500             CALL    FDOS    ;TYPE SIGN-ON MESSAGE
                ;--WAIT FOR CONSOLE INPUT
097C 0E01               MVI     C,RDCON
097E CD0500             CALL    FDOS    ;READ CONSOLE DATA
0981 FE04               CPI     EOT
0983 CA1107             JZ      EXIT    ;"^D"?
0986 E65F               ANI     05FH    ;  NO, SHIFT LC ALPLH TO UC
0988 FE48               CPI     'H'
098A C29D09             JNZ     BAUD    ;HAND-SHAKE REQUEST?
098D 3EFF               MVI     A,0FFH  ;  YES
098F 323701             STA     HSHAKE  ;SET THE HAND-SHAKE FLAG TO ON
0992 0E09               MVI     C,WRCBUF
0994 11E604             LXI     D,MSG15
0997 CD0500             CALL    FDOS    ;TYPE "HAND-SHAKE" MSG.
099A C3D209             JMP     D0
                ;--BAUD RATE MODIFICATION
099D FE42       BAUD:   CPI     'B'
099F C2D209             JNZ     D0      ;BAUD-RATE CHANGE REQUEST?
09A2 0E09               MVI     C,WRCBUF;  YES
09A4 110205             LXI     D,MSG16
09A7 CD0500             CALL    FDOS    ;PROMPT FOR BAUD SELECTION
09AA CD1F07     BA1:    CALL    CSTAT
09AD CAAA09             JZ      BA1     ;HAS OPERATOR RESPONDED?
09B0 0E01               MVI     C,RDCON ;  YES, GET THE CHARACTER
09B2 CD0500             CALL    FDOS
09B5 F5                 PUSH    PSW
09B6 FE34               CPI     '4'
09B8 DAC009             JC      BA2     ;BAUD-RATE > 300 ?
09BB 3E6B               MVI     A,MMHREL        ;  YES, CHANGE MODEM MODE ACCORDINGLY
09BD 323401             STA     MMODE
09C0 CD6007     BA2:    CALL    CRLF
```

```
09C3 F1             POP     PSW
09C4 E607           ANI     07H
09C6 3D             DCR     A
09C7 5F             MOV     E,A
09C8 1600           MVI     D,00H   ;D,E = TBL INDEX
09CA 215F01         LXI     H,BDTBL
09CD 19             DAD     D
09CE 7E             MOV     A,M
09CF 326501         STA     BDVAL   ;NEW TRANSMISSION BAUD-RATE VALUE
                ;--DETERMINE MODEM MODE (ORIGINATE/ANSWER)
09D2 0E09       DO:     MVI     C,WRCBUF
09D4 115B02             LXI     D,MSG1A
09D7 CD0500             CALL    FDOS    ;TYPE "MODE" MESSAGE
09DA 0E01               MVI     C,RDCON
09DC CD0500             CALL    FDOS    ;READ CONSOLE DATA
09DF FE04               CPI     EOT
09E1 CA5A09             JZ      RESTRT  ;"^D"?
09E4 E65F               ANI     05FH    ;  NO, SHIFT LC ALPHA TO UC
09E6 FE4F               CPI     'O'
09E8 CAF809             JZ      ORIG    ;ORIGINATE-MODE SELECTED?
09EB FE41               CPI     'A'     ;  NO
09ED C2D209             JNZ     DO      ;ANSWER-MODE SELECTED?
09F0 3EFF               MVI     A,0FFH  ;  YES
09F2 323B01             STA     ANSFLG
09F5 C3F40A             JMP     ANSWER
                ;
                ;++++++++++++++++++++++++ ORIGINATE MODE ++++++++++++++++++++++++++++++
                ;
                ;--SET UP DIALER BAUD RATE
09F8 3E04       ORIG:   MVI     A,BDSEL ;  NO
09FA D3C2               OUT     MODEM2
09FC 3E34               MVI     A,DRATE2
09FE D3C3               OUT     MODEM3
                ;--GO OFF-HOOK & SET BREAK RELEASE
0A00 3E01               MVI     A,MMSELO
0A02 D3C2               OUT     MODEM2
0A04 3EC0               MVI     A,MMORIG
0A06 D3C3               OUT     MODEM3
0A08 0E09               MVI     C,WRCBUF
0A0A 11C002             LXI     D,MSG1B
0A0D CD0500             CALL    FDOS    ;TYPE "WAITING" MESSAGE
                ;--WAIT FOR DIAL TONE OR "^D"
0A10 CD1F07     DTONE:  CALL    CSTAT   ;GET CONSOLE STATUS
0A13 CA190A             JZ      D1      ;CONSOLE DATA AVAILABLE?
0A16 CD2607             CALL    CRD$ST  ;  YES, TEST FOR "^D"
0A19 CD3207     D1:     CALL    MSTAT   ;GET MODEM STATUS
0A1C E680               ANI     DTD
0A1E C2100A             JNZ     DTONE   ;DIAL TONE PRESENT?
0A21 0E09               MVI     C,WRCBUF ;  YES, INFORM OPERATOR
0A23 11F402             LXI     D,MSG2
0A26 CD0500             CALL    FDOS
                ;--BEGIN DIALING SEQUENCE
0A29 3E02               MVI     A,DLSEL
0A2B D3C2               OUT     MODEM2
0A2D 3A3A01             LDA     ROTFLG
0A30 B7                 ORA     A
0A31 CA380A             JZ      DIAL    ;ROTARY DIALING IN PROGRESS?
0A34 F1                 POP     PSW     ;  YES, GET ROTARY CHARACTER
0A35 C3530A             JMP     AUTODL
0A38 CD2607     DIAL:   CALL    CRD$ST  ;CONSOLE READ
0A3B FE0D               CPI     CR
0A3D CA820A             JZ      UART0   ;CARRIAGE RETURN?
                ;--SCREEN OUT NON-DECIMAL "DIALED" CHARACTERS
0A40 FE30               CPI     '0'     ;  NO
0A42 DA4907             JC      DCHAR   ;CONTROL CHAR. OR SYMBOL?
0A45 FE3A               CPI     ':'     ;  NO
0A47 DA4D0A             JC      DIGIT   ;NON-DECIMAL CHARACTER?
0A4A C3530A             JMP     AUTODL  ;  YES, PERFORM AUTO-DIALING
0A4D CD8207     DIGIT:  CALL    DIALIT  ;DIAL A DIGIT
0A50 C3380A             JMP     DIAL
                ;--AUTOMATIC DIALING ROUTINE
0A53 21FFFF     AUTODL: LXI     H,0FFFFH
0A56 223801             SHLD    AUTOD   ;SET AUTO-DIAL FLAG
0A59 F5                 PUSH    PSW     ;SAVE CHARACTER FOR ROTARY DIALING TEST
0A5A E603               ANI     03H
0A5C 07                 RLC
0A5D 5F                 MOV     E,A
0A5E 1600               MVI     D,0
0A60 216601             LXI     H,ADRTBL
0A63 19                 DAD     D       ;H,L=INDIRECT ADDRESS OF PHONE NUMBER
0A64 5E                 MOV     E,M
0A65 23                 INX     H
0A66 56                 MOV     D,M
0A67 EB                 XCHG            ;H,L=ADDRESS OF PHONE NUMBER
0A68 7E         AU1:    MOV     A,M
0A69 FE1D               CPI     GS
0A6B CACF0A             JZ      ROTEND  ;END OF ROTARY GROUP?
0A6E FE24               CPI     '$'     ;  NO
0A70 CA810A             JZ      UART0-1 ;DIALING COMPLETE?
0A73 E5                 PUSH    H       ;  NO
0A74 F5                 PUSH    PSW
0A75 CD4207             CALL    TCHAR   ;TYPE DIGIT TO BE DIALED
0A78 F1                 POP     PSW
0A79 CD8207             CALL    DIALIT
0A7C E1                 POP     H
0A7D 23                 INX     H
0A7E C3680A             JMP     AU1
                ;--SET UP UART CONFIGURATION
0A81 E5                 PUSH    H       ;SAVE ADDRESS POINTER FOR ROTARY DIALING
0A82 0E02       UART0:  MVI     C,WRCON
0A84 1E0A               MVI     E,LF
0A86 CD0500             CALL    FDOS    ;SEND "LINE-FEED" TO CONSOLE
0A89 CD9D07             CALL    UART    ;ESTABLISH UART CONFIGURATION
                ;--WAIT FOR CTS (CARRIER DETECT)
0A8C 0E09               MVI     C,WRCBUF
0A8E 113303             LXI     D,MSG2A
0A91 CD0500             CALL    FDOS    ;TYPE "WAITING" MESSAGE
0A94 01FFFF             LXI     B,0FFFFH ;INITIALIZE TIME-OUT
0A97 C5                 PUSH    B
0A98 C1         CTS1:   POP     B
0A99 0B                 DCX     B
0A9A 78                 MOV     A,B
0A9B B1                 ORA     C
0A9C CAB40A             JZ      ROTARY  ;TIME-OUT EXPIRED?
0A9F C5                 PUSH    B       ;  NO
0AA0 CD1F07             CALL    CSTAT   ;GET CONSOLE STATUS
0AA3 CAA90A             JZ      C1      ;CONSOLE DATA AVAILABLE?
0AA6 CD2607             CALL    CRD$ST  ;  YES, TEST FOR "^D"
0AA9 CD3207     C1:     CALL    MSTAT
0AAC E640               ANI     CTS
0AAE CA980A             JZ      CTS1    ;CLEAR TO SEND?
0AB1 C3DD0A             JMP     MDMO    ;  YES, SET UP MODEM CONFIGURATION
                ;--ROTARY DIALING SEQUENCE
0AB4 3A3801     ROTARY: LDA     AUTOD
0AB7 B7                 ORA     A
0AB8 CA320D             JZ      STOP    ;AUTO-DIAL MODE?
0ABB E1                 POP     H       ;  YES, GET PH# ADDRESS
0ABC 23                 INX     H
0ABD F1                 POP     PSW     ;GET CHARACTER WHICH INITIATED AUTO-DIAL
0ABE FE4D               CPI     'M'
0AC0 C2320D             JNZ     STOP    ;ROTARY CHARACTER?
0AC3 F5                 PUSH    PSW     ;  YES, SAVE FOR NEXT ITERATION
0AC4 3EFF               MVI     A,0FFH
0AC6 323A01             STA     ROTFLG  ;SET ROTARY DIALING FLAG
0AC9 226801             SHLD    ADRTBL+2;STORE INCREMENTED POINTER
0ACC C3320D             JMP     STOP    ;TRY THE NEXT PHONE NUMBER
0ACF 216E01     ROTEND: LXI     H,PHONEM
0AD2 226801             SHLD    ADRTBL+2;RESTORE ORIGINAL POINTER
0AD5 AF                 XRA     A
0AD6 323A01             STA     ROTFLG  ;RESET ROTARY DIALING FLAG
0AD9 F1                 POP     PSW     ;RESTORE STACK TO NORMAL
0ADA C3320D             JMP     STOP
                ;--SET UP MODEM CONFIGURATION
0ADD 3E01       MDMO:   MVI     A,MMSELO
0ADF 322D01             STA     MMSEL
0AE2 CDAB07             CALL    MDM
                ;--PROMPT OPERATOR TO BEGIN COMMUNICATIONS
0AE5 0E09               MVI     C,WRCBUF
0AE7 117003             LXI     D,MSG4
0AEA CD0500             CALL    FDOS    ;TYPE START TX MESSAGE
0AED AF                 XRA     A
0AEE 323A01             STA     ROTFLG  ;RESET ROTARY DIALING FLAG
0AF1 C3510B             JMP     TRCHAR
                ;
                ;++++++++++++++++++++++++ ANSWER MODE ++++++++++++++++++++++++++++++
                ;
0AF4 0E09       ANSWER: MVI     C,WRCBUF
0AF6 11BB05             LXI     D,MSG17
0AF9 CD0500             CALL    FDOS    ;TYPE "ANSWER MODE" MSG
0AFC 3EFF               MVI     A,0FFH
0AFE 323C01             STA     LFDFLG  ;SET LINE.FEED FLAG
0B01 CD9D07             CALL    UART    ;SET UP UART CONFIGURATION
0B04 3E31               MVI     A,MMSELA
0B06 322D01             STA     MMSEL
0B09 CDAB07             CALL    MDM     ;SET UP MODEM CONFIGURATION
0B0C 0E09               MVI     C,WRCBUF
```

```
0B0E 11E205             LXI     D,MSG18
0B11 CD0500             CALL    FDOS    ;TYPE "WAITING" MESSAGE
                ;--WAIT FOR CTS (RING INDICATOR & CARRIER DETECT)
0B14 CD1F07     AN0:    CALL    CSTAT
0B17 CA2B0B             JZ      AN1     ;CONSOLE DATA AVAILABLE?
0B1A 0E01               MVI     C,RDCON ;  YES
0B1C CD0500             CALL    FDOS    ;READ CONSOLE DATA
0B1F FE04               CPI     EOT
0B21 C22B0B             JNZ     AN1     ;"^D"?
0B24 AF                 XRA     A       ;  YES
0B25 323B01             STA     ANSFLG  ;RESET ANSWER MODE FLAG
0B28 C3320D             JMP     STOP
0B2B DBC0       AN1:    IN      MODEM0  ;  NO
0B2D E640               ANI     CTS
0B2F CA140B             JZ      AN0     ;HAS CONNECTION BEEN ESTABLISHED?
0B32 0E09               MVI     C,WRCBUF;  YES, INFORM CONSOLE OPERATOR
0B34 117003             LXI     D,MSG4
0B37 CD0500             CALL    FDOS
                ;--TRANSMIT SIGN-ON MESSAGE TO REMOTE DEVICE
0B3A 211306             LXI     H,MSG19 ;H,L = MESSAGE ADDRESS
0B3D CD3207     SIGNON: CALL    MSTAT
0B40 E601               ANI     TBE
0B42 CA3D0B             JZ      SIGNON  ;TX BUFFER EMPTY?
0B45 7E                 MOV     A,M     ;  YES, GET NEXT CHARACTER
0B46 FE24               CPI     '$'
0B48 CA510B             JZ      TRCHAR  ;MESSAGE COMPLETE?
0B4B D3C1               OUT     MODEM1  ;  NO, TRANSMIT THE CHARACTER
0B4D 23                 INX     H
0B4E C33D0B             JMP     SIGNON
                ;
                ;+++++++++++++++++ CHARACTER TX/RX ROUTINE ++++++++++++++++++++++++++
                ;
                ;--TEST FOR LOSS OF "CTS"
0B51 DBC0       TRCHAR: IN      MODEM0
0B53 E640               ANI     CTS
0B55 CA320D             JZ      STOP    ;LOSS OF CTS?
                ;--TEST FOR "LINE CHARACTER" AVAILABILITY
0B58 DBC0               IN      MODEM0  ;  NO
0B5A E602               ANI     RDA
0B5C CA9C0B             JZ      NOCHAR  ;RECEIVE DATA AVAILABLE?
0B5F DBC1               IN      MODEM1  ;  YES, READ DATA
0B61 F5                 PUSH    PSW
0B62 CD4207             CALL    TCHAR   ;TYPE RECEIVED CHARACTER
0B65 3A3C01             LDA     LFDFLG
0B68 B7                 ORA     A
0B69 CA850B             JZ      TR1     ;LINE FEED FLAG SET?
0B6C F1                 POP     PSW     ;  YES
0B6D FE0D               CPI     CR
0B6F C2860B             JNZ     TR1+1   ;CHARRIAGE RETURN?
0B72 3E0A       TRLF:   MVI     A,LF    ;  YES, SEND A LINE FEED
0B74 CD4207             CALL    TCHAR
0B77 DBC0       TR0:    IN      MODEM0
0B79 E601               ANI     TBE
0B7B CA770B             JZ      TR0     ;TX BUFFER EMPTY?
0B7E 3E0A               MVI     A,LF    ;  YES
0B80 D3C1               OUT     MODEM1  ;TRANSMIT LINE FEED TO LINE
0B82 C39C0B             JMP     NOCHAR
0B85 F1         TR1:    POP     PSW
0B86 FE14               CPI     DC4
0B88 CACF0B             JZ      TXFILE  ;"^T"?
0B8B FE12               CPI     DC2     ;  NO
0B8D CA920C             JZ      RXFILE  ;"^R"?
0B90 FE04               CPI     EOT     ;  NO
0B92 C29C0B             JNZ     NOCHAR  ;"^D"?
0B95 AF                 XRA     A       ;  YES
0B96 323B01             STA     ANSFLG  ;RESET THE ANSER.MODE FLAG
0B99 C3320D             JMP     STOP    ;TERMINATE ANSWER MODE
                ;--TEST FOR "LOCAL CHARACTER" AVAILABILITY
0B9C CD1F07     NOCHAR: CALL    CSTAT   ;GET CONSOLE STATUS
0B9F CA510B             JZ      TRCHAR  ;CONSOLE DATA AVAILABLE?
0BA2 CD2607             CALL    CHD$ST  ;  YES, READ DATA & TEST FOR "^D"
0BA5 FE14               CPI     DC4
0BA7 CACF0B             JZ      TXFILE  ;"^T"?
0BAA FE12               CPI     DC2     ;  NO
0BAC CA920C             JZ      RXFILE  ;"^R"?
0BAF FE01               CPI     SOH
0BB1 CA3F0B             JZ      ASSIGN  ;IF "^A" THEN CHANGE CONSOLE ASSIGNMENT
0BB4 FE02               CPI     STX
0BB6 CAC007             JZ      BREAK   ;IF "^B" THEN TRANSMIT A "BREAK"
0BB9 FE06               CPI     ACK
0BBB CAF607             JZ      FILE    ;IF "^F" THEN SET UP NEW FILE NAME
0BBE D3C1               OUT     MODEM1  ;TRANSMIT DATA TO "LINE"
0BC0 FE0D               CPI     CR
0BC2 C2510B             JNZ     TRCHAR  ;CARRIAGE RETURN?
0BC5 3A3C01             LDA     LFDFLG  ;  YES
0BC8 B7                 ORA     A
0BC9 C2720B             JNZ     TRCF    ;LINE FEED FLAG SET?
0BCC C3510B             JMP     TRCHAR  ;  NO
                ;
                ;++++++++++++++++++++++ FILE TX ROUTINE ++++++++++++++++++++++++++
                ;
0BCF CD6306     TXFILE: CALL    OPENF   ;OPEN DISK FILE FOR READING
0BD2 3E80               MVI     A,80H
0BD4 323601             STA     BUFPTR  ;INITIALIZE THE BUFFER POINTER
0BD7 AF                 XRA     A
0BD8 323501             STA     CHCNT   ;INITIALIZE CHARACTER POSITION COUNT
                ;--START OF BYTE TRANSMIT LOOP
0BDB CD1F07     TXBYTE: CALL    CSTAT   ;GET CONSOLE STATUS
0BDE CAEB0B             JZ      TF0     ;DATA AVAILABLE?
0BE1 0E01               MVI     C,RDCON ;  YES
0BE3 CD0500             CALL    FDOS
0BE6 FE04               CPI     EOT
0BE8 CA510B             JZ      TRCHAR  ;"^D" ?
0BEB DBC0       TF0:    IN      MODEM0  ;  NO
0BED E602               ANI     RDA
0BEF CAF90B             JZ      TF1A    ;RX DATA AVAILABLE?
0BF2 DBC1               IN      MODEM1  ;  YES
0BF4 FE04               CPI     EOT
0BF6 CA510B             JZ      TRCHAR  ;"^D"?
0BF9 3A3601     TF1A:   LDA     BUFPTR  ;  NO
0BFC FE80               CPI     80H
0BFE C2050C             JNZ     TF1     ;BUFFER EMPTY?
                ;--READ A SECTOR & EXIT IF END OF FILE
0C01 CD8108             CALL    DISKR   ;  YES
0C04 AF                 XRA     A
0C05 5F         TF1:    MOV     E,A     ;D,E = DATA BYTE ADDRESS
0C06 1600               MVI     D,0
0C08 3C                 INR     A
0C09 323601             STA     BUFPTR  ;INCREMENTED BUFFER POINTER
                ;--READ A BYTE AT ADDRESS: TBUFF+(D,E)
0C0C 218000             LXI     H,TBUFF
0C0F 19                 DAD     D
0C10 7E                 MOV     A,M
0C11 F5                 PUSH    PSW
                ;--WAIT FOR UART TX BUFFER EMPTY
0C12 CD3207     TF2:    CALL    MSTAT
0C15 E601               ANI     TBE
0C17 CA120C             JZ      TF2     ;UART TX BUFFER EMPTY?
0C1A F1                 POP     PSW     ;  YES
0C1B FE17               CPI     ETB
0C1D CA8C0C             JZ      TFEND   ;(^W) END TRANSMISSION BLOCK?
0C20 FE09               CPI     HT      ;  NO
0C22 C2420C             JNZ     TF3     ;HORIZONTAL TAB?
                ;--TRANSLATE HT INTO APPROPRIATE NUMBER OF SPACES
0C25 3A3501             LDA     CHCNT   ;  YES
0C28 47         SPACES: MOV     B,A
0C29 CD3907             CALL    MTBE    ;INSURE MODEM TX BUFFER EMPTY
0C2C 3E20               MVI     A,' '
0C2E D3C1               OUT     MODEM1
0C30 C5                 PUSH    B
0C31 CD4207             CALL    TCHAR
0C34 C1                 POP     B
0C35 04                 INR     B
0C36 78                 MOV     A,B
0C37 E607               ANI     07H     ;MOD 8 COUNT VALUE
0C39 C2280C             JNZ     SPACES  ;FINAL SPACE SENT?
0C3C 323501             STA     CHCNT
0C3F C3DB0B             JMP     TXBYTE
                ;--SEND DATA BYTE & UPDATE CHCNT (CHARACTER POSITION COUNT)
0C42 F5         TF3:    PUSH    PSW
0C43 CDA808             CALL    SHOW    ;DISPLAY BYTE ON CONSOLE
0C46 F1                 POP     PSW
0C47 D3C1               OUT     MODEM1
0C49 FE0A               CPI     LF
0C4B CA580C             JZ      TFEOL   ;"LF" SIGNALS END OF TEXT LINE
0C4E 3A3501             LDA     CHCNT
0C51 3C                 INR     A
0C52 323501             STA     CHCNT   ;INCREMENTED CHAR POSITION COUNT
0C55 C3DB0B             JMP     TXBYTE
                ;--PERFORM END OF LINE PROCESSING
0C58 AF         TFEOL:  XRA     A
0C59 323501             STA     CHCNT   ;RESET CHAR POSITION COUNT
0C5C 3A3701             LDA     HSHAKE
0C5F B7                 ORA     A
0C60 C2680C             JNZ     TF4     ;HAND-SHAKE REQUIRED?
0C63 060F               MVI     B,15    ;  NO, GENERATE 0.5 SEC DELAY
```

```
0C65 CDDA07             CALL    DELAY
0C68 C3DB0B             JMP     TXBYTE  ;  YES, BEGIN NEXT LINE
                ;--PERFORM END OF TEXT LINE "HANDSHAKING"
0C6B CD1F07     TF4:    CALL    CSTAT
0C6E CA740C             JZ      TF4A    ;CONSOLE CHAR RECEIVED?
0C71 C38C0C             JMP     TFEND   ;  YES, TERMINATE FILE TX
0C74 CD3207     TF4A:   CALL    MSTAT   ;  YES
0C77 E602               ANI     RDA
0C79 CA6B0C             JZ      TF4     ;RECEIVE DATA AVAILABLE?
0C7C DBC1               IN      MODEM1  ;  YES
0C7E FE03               CPI     ETX
0C80 C26B0C             JNZ     TF4     ;"END TEXT" RECEIVED FROM LINE?
0C83 0620               MVI     B,20H   ;  YES, INSERT 60MS DELAY
0C85 05         TF5:    DCR     B
0C86 C2850C             JNZ     TF5
0C89 C3DB0B             JMP     TXBYTE
0C8C CD6007     TFEND:  CALL    CRLF
0C8F C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
                ;
                ;++++++++++++++++++++ FILE RX ROUTINE ++++++++++++++++++++++++++
                ;
0C92 0E13       RXFILE: MVI     C,DELETE
0C94 115C00             LXI     D,FCB
0C97 CD0500             CALL    FDOS    ;DELETE NAMED FILE (IF IT EXISTS)
0C9A CDCC08             CALL    MAKEF   ;CREATE FILE FOR RX DATA
0C9D AF                 XRA     A
0C9E 323601             STA     BUFPTR  ;INITIALIZE THE BUFFER POINTER
0CA1 0E09               MVI     C,WRCBUF
0CA3 116B04             LXI     D,MSG10
0CA6 CD0500             CALL    FDOS    ;TYPE "READY" MSG
                ;--RECEIVE A CHARACTER FROM THE REMOTE DEVICE
0CA9 DBC0       RCV:    IN      MODEM0
0CAB E602               ANI     RDA
0CAD C2C50C             JNZ     GOTONE  ;RECEIVED DATA AVAILABLE?
                ;--SUPPORT CHARACTER TX FROM CONSOLE
0CB0 CD1F07             CALL    CSTAT   ;  YES, GET CONSOLE STATUS
0CB3 CAA90C             JZ      RCV     ;DATA AVAILABLE?
0CB6 0E01               MVI     C,RDCON ;  YES
0CB8 CD0500             CALL    FDOS
0CBB FE04               CPI     EOT
0CBD CA1C0D             JZ      RFEND   ;"^D" ?
0CC0 D3C1               OUT     MODEM1  ;  NO, TRANSMIT CHARACTER
0CC2 C3A90C             JMP     RCV
                ;--READ CHARACTER FROM EXTERNAL DEVICE
0CC5 DBC1       GOTONE: IN      MODEM1
0CC7 FE17               CPI     ETB
0CC9 CAE50C             JZ      RF0     ;"^W"?
0CCC FE04               CPI     EOT     ;  NO
0CCE CA1C0D             JZ      RFEND   ;"^D"?
0CD1 FE0D               CPI     CR      ;  NO
0CD3 CAEB0C             JZ      RFCR    ;CARRIAGE RETURN?
0CD6 FE7F               CPI     DEL     ;  NO
0CD8 CAA90C             JZ      RCV     ;DO NOT RECORD RUBOUTS
0CDB FE20               CPI     ' '
0CDD DAA90C             JC      RCV     ;ASCII CONTROL CHARACTER?
0CE0 F5                 PUSH    PSW     ;  NO
0CE1 CD4207             CALL    TCHAR   ;DISPLAY CHARACTER ON CONSOLE
0CE4 F1                 POP     PSW
0CE5 CDEA08     RF0:    CALL    ST$BUF  ;STORE CHARACTER IN DISK BUFFER
0CE8 C3A90C             JMP     RCV
                ;--CARRIAGE RETURN PROCESSING
0CEB F5         RFCR:   PUSH    PSW
0CEC CD4207             CALL    TCHAR   ;TYPE "CR"
0CEF F1                 POP     PSW
0CF0 CDEA08             CALL    ST$BUF  ;PUT IT IN BUFFER
0CF3 3E0A               MVI     A,LF
0CF5 F5                 PUSH    PSW
0CF6 CD4207             CALL    TCHAR   ;TYPE "LF"
0CF9 F1                 POP     PSW
0CFA CDEA08             CALL    ST$BUF  ;PUT IT IN BUFFER
0CFD 3A3C01             LDA     LFDFLG
0D00 B7                 ORA     A
0D01 CAA90C             JZ      RCV     ;LINE.FEED FLAG SET?
0D04 DBC0       RF1:    IN      MODEM0  ;  YES
0D06 E601               ANI     TBE
0D08 CA040D             JZ      RF1     ;TX BUFFER EMPTY?
0D0B 3E0A               MVI     A,LF    ;  YES
0D0D D3C1               OUT     MODEM1  ;SEND LINE.FEED TO REMOTE DEVICE
0D0F C3A90C             JMP     RCV
                ;--BUFFER IS FULL - WRITE A DISK SECTOR
0D12 CD0509     RFULL:  CALL    DISKW   ;  YES
0D15 AF                 XRA     A
0D16 323601             STA     BUFPTR
0D19 C3A90C             JMP     RCV
                ;--WRITE FINAL SECTOR & CLOSE FILE
0D1C 3A3601     RFEND:  LDA     BUFPTR
0D1F 5F                 MOV     E,A
0D20 1600               MVI     D,0
0D22 218000             LXI     H,TBUFF
0D25 19                 DAD     D
0D26 3E1A               MVI     A,01AH  ;"^Z" INDICATES END OF FILE
0D28 77                 MOV     M,A
0D29 CD0509             CALL    DISKW   ;WRITE THE LAST RECORD
0D2C CD3C09             CALL    CLOSEF
0D2F C3510B             JMP     TRCHAR  ;REVERT TO CHARACTER MODE
                ;
                ;++++++++++++++++++ TERMINATE COMMUNICATIONS ++++++++++++++++++++++
                ;
                ;--SEND A BREAK, HANG UP THE PHONE, INFORM THE OPERATOR
0D32 3E01       STOP:   MVI     A,MMSELO
0D34 D3C2               OUT     MODEM2
0D36 3E0E               MVI     A,MMTERM
0D38 D3C3               OUT     MODEM3
0D3A 0E09               MVI     C,WRCBUF
0D3C 115103             LXI     D,MSG3
0D3F CD0500             CALL    FDOS    ;TYPE TERMINATION MESSAGE
                ;--CONTINUE TO LOOP TILL LOSS OF CTS (CARRIER DETECT)
0D42 CD3207     CTS2:   CALL    MSTAT
0D45 E640               ANI     CTS
0D47 C2420D             JNZ     CTS2    ;HAS CTS BEEN RESET?
0D4A 1605               MVI     D,5     ;  YES, INITIALIZE TIME-OUT
0D4C 01FFFF     STIME:  LXI     B,0FFFFH
0D4F 0B         STO:    DCX     B
0D50 78                 MOV     A,B
0D51 B1                 ORA     C
0D52 C24F0D             JNZ     STO     ;INNER T.O. COMPLETE?
0D55 15                 DCR     D       ;  YES
0D56 C24C0D             JNZ     STIME   ;TIME-OUT COMPLETE?
0D59 3A3A01             LDA     ROTFLG  ;  YES
0D5C B7                 ORA     A
0D5D C2F809             JNZ     ORIG    ;ROTARY DIALING IN PROGRESS?
0D60 3A3B01             LDA     ANSFLG  ;  NO
0D63 B7                 ORA     A
0D64 C2F40A             JNZ     ANSWER  ;ANSWER MODE?
0D67 C35A09             JMP     RESTRT  ;  NO
0D6A                    END     START
```

DOUBLE-HEADED DRIVE FOR $755

Received: 78 May 10

News Release

Pertec Computer Corporation's Pertec Division announces the FD650, a standard-sized (8-inch), double-headed flexible disk drive capable of recording and reading data on both sides of an IBM (or equivalent) Diskette 2 or 2D. The new drive also features double density recording to offer an immediately addressable unformatted storage capacity of 1.6 Megabytes. The FD650 is media compatible with IBM Diskettes 2 and 2D and electronically compatible with the current proposed ANSI standard. It will be incorporated into several MITS/Altair and iCOM systems manufactured and marketed by Pertec Computer Corporation's Microsystems Division.

The new two-sided diskette drive is capable of double-density operation using MFM encoding. Each drive is equipped with a door lock, and the drive has die-cast construction. All electronics are on a single printed circuit board. Track-to-track access time is 3 msec. The user can daisy chain up to eight drives. Head positioning is achieved by a steel band attached to the head carriage and to a drive pulley on the shaft of a four-phase 1.8° permanent magnet stepper motor. Each step of the motor causes the head to move one track.

The FD650 is priced at $755 is single quantities. OEM quantity discounts are available. Contact PCC, Pertec Division, 9600 Irondale Avenue, Chatsworth, CA 91311; (213) 822-9222, Neil McElwee.

AUTODUMP: A MEMORY DUMPER FOR POLY88 TAPE SYSTEM

BY RICHARD B. EMERSON
158 E. Butler Ave.
Ambler, PA 19002
© 1978

There comes a point in every programmer's life when it seems to be a good idea to store the contents of memory on a permanent medium like tape. This usually comes about two seconds after said programmer realizes that a particularly brilliant piece of code has just been clobbered by a stray glitch in memory. By then it is too late, but for those who plan ahead there is hope and AUTODUMP. This program of 1047 bytes will dump the contents of a POLY88 onto tape in either BYTE or POLYPHASE mode. That, in itself is not much, but AUTODUMP will also write a banner or comment record block of 255 characters and automatic execution of a program dumped by AUTODUMP. If you are sufficiently impressed with your handiwork, you can command AUTODUMP to produce up to 255 copies of that file. All that is required of your system is that the POLY88 USART kit be installed on the CPU card. There are three monitor subroutine references and four memory references related to the monitor. Each one will be explained in the following paragraphs.

The accompanying source listing has arbitrarily assembled to begin at 6000H but there is no limitation on where it can begin. The absolute references are all outside of the program and the various constants do not take advantage of page boundaries or any similar technique. The data area at the start is preceded by a "JMP" only to make sure that the data is not accidentally erased. The actual program starts at the label ENTER. If you wish to load AUTODUMP first and examine its workings later, be sure you read the following paragraph! There are some very interesting features in the program and it would be a shame to miss them.

Operation

As stated above, AUTODUMP will dump files in either BYTE or POLYPHASE mode. On beginning execution, the first message is:

TAPE MODE:

Type either P or B depending on the desired mode. Do not forget to connect the correct output lead to your tape recorder as this is CRUCIAL! With the mode selected the next message will be:

TAPE NAME:

Type in the file name that you want. All characters are legal except <RUBOUT> (A note about notation: <CHAR> denotes an ASCII symbol. Resist the urge to type the eight characters mentioned above to clear a letter.) It is suggested that names be limited to alpha-numeric characters. If a wrong character is typed in, typing <RUBOUT> will not appear in the file name as it does with some POLY88 routines. With the name loaded the next message will be:

START ADDR:

AUTODUMP is asking for a hex address that marks the start of the area to be dumped. Typing any character other than 0 through 9, A through F, or a carriage return, <CR>, will cause the question to be repeated. If you realize that the address typed was wrong, typing any invalid character will restart the request in progress. This rule applied to all hexadecimal entries in AUTODUMP. With the starting address loaded the next question will be:

END ADDR:

The address that marks the end of the dump area has been requested. It is suggested that, if the area ends at FF42, that the entry should be FF43. As you will see from the code, the area need not be on a page boundary. With the dump area defined, AUTODUMP will now want to know where the data is to go upon reloading. This address is requested with the message:

LOAD ADDR:

The address may be the same as the address for START ADDR or it may be completely different. Bear in mind, however, that AUTODUMP does not perform code relocation but merely tells the tape loader routines where the data is to be loaded. If you assemble a program at 8800H and reload it at 2000H, it will not run unless the address references were based on 2000H. The final address requested is the place where program execution is to begin and the message is:

EXEC ADDR:

This address may be any address in your system, it does not have to be within the dumped file. Thus you could load data and then go execute a program that will munch on the data or you can execute the data that has been loaded. Naturally it is also possible to begin execution anywhere in the dumped area. If you do not want the file to auto-execute, enter any address or type <CR>. Omitting the banner record will keep the file from causing automatic execution. With the execution address entered, the next question will be:

NO. COPIES
IN HEX:

Type any valid hex number and you will get that many copies of the dump, each with the banner or comment that will be typed in, each will start at the same place, and each will end at the same place. No additional type-ins will be needed save that of starting the next copy. The banner or comment record serves two purposes, first it allows a file to identify itself and second, it serves as a flag for auto-execution. If there are no characters in the banner table, at the end of the dump the file is closed without an auto-execute record block. The banner is requested with:

COMMENT <CTRL/D TO END>:

Avoid typing in control <CTRL/*> characters as they can confuse the POLY88. <RUBOUT> erases characters but each <RUBOUT> counts as a character so it is better to erase the banner when more than one <RUBOUT> is needed. Typing <CTRL/X> will clear the banner table and also clear the screen. If your keyboard cannot produce lower case letters, don't despair. AUTODUMP looks for a character that is reserved as the "flip" character. This character tells the banner routine to flip from UPPER CASE to lower case. Note, however, that you must "flip" back to upper case before typing <CR> or <RUBOUT>. Clearing the screen with <CTRL/X> also resets the "flip" flag. You may flip from one mode to the other as often as you wish; use of the "flip" character does not take up banner space. When the banner is typed in, type <CTRL/D> and you will get:

READY TAPE, AND PRESS ANY LETTER:

AUTODUMP starts dumping only when you give it the go-ahead. It does not start before the banner is finished unless you have typed more than 255 characters. If you find AUTODUMP has begun before you are ready, press RESET on the POLY88 and then <CTRL/Z> to enter the front panel mode. From there just type <SP Jaddr> where addr is the starting address for AUTODUMP. You will have to re-enter the addresses and so on, but at least you won't be waiting for 10K of core to dump out in BYTE mode! It is possible to dump different files without reloading or restarting AUTODUMP. When the last copy of the current file has been written out and its execution address listed, AUTODUMP will clear itself and say:

TAPE MODE:

This is where you came in. If you want to, go play with AUTODUMP and then come back to see how it works.

Description

In effect AUTODUMP is an oversized "block move" command except that the receiving area of the move is a tape and not a location in the memory of the POLY88. Additionally provisions are made for breaking the transfer up into 256 byte blocks to allow restarting a file in the middle of a read. The USART is driven much the way that it is in BPRINT, PolyMorphic's printer driver for BASIC, except that there is no large buffer. The USART generates an interrupt and AUTODUMP makes sure that data is ready when the USART asks for it.

The monitor references appear at the beginning of the listing. SETUP is a routine that initializes the USART with the data that follows "CALL SETUP". Notice how the manipulation of the stack saves much time and confusion. HEXC is a routine that accepts hex characters 0 through F and converts them to a hex address that is put in the HL register. B contains the number of characters in HL. If there are less than four characters the leftmost bits are set to zeroes.

Any non-hex character causes a return from HEXC and the last four valid characters are in HL. DEOUT outputs the contents of the DE register as a two byte hex address. TBUFF is a two byte area that holds a status flag and data for the USART when it is in the tape mode. SRA4 is part of the interrupt vector table and it points to a routine that will service the interrupting device. WormHole 0 is the keyboard driver and puts data from the keyboard into the A register. WormHole 1 is the entry to the CRT driver and puts the contents of the A register onto the CRT.

The data area of AUTODUMP consists of two areas, the record header text and scratch space including the banner table. The order of the header area must not be modified or the POLY88 monitor will not be able to read the tape. This is because the data stored here is copied directly by use of the HL register. The scratch space is not restricted and may be placed anywhere that has enough space. The jump over the data area helps to define the data space and is there for psychological reasons more than anything else. If you need the three bytes, take it out and start execution at ENTER's address.

ENTER puts the address of TISR into the Interrupt Service Routine Address table so that when the USART causes an interrupt the CPU will know where to go. After TISR is set up the screen is cleared and the mode request is displayed. SHOW is a simple routine that takes data from a table and ships it to the CRT via WH1. This wormhole takes the contents of the A register and puts it into the memory dedicated to the display. This process will continue until a 00 byte is seen. There is no limit on the string's length or content except that the string will end with a zero byte. The string is selected by putting the proper address in the HL register. WH0 is used to get the type-in and that character is returned in the A register. Bad data is ignored and good data is displayed or "echoed".

With the mode selected, SETUP is called. This routine takes the bytes that follow the call and outputs them into the USART command register. SETUP pops the contents of the stack into the HL register. Since this is the address of the call plus two bytes, HL now points at the first byte after the operand of the call. If "CALL SETUP" is at address 6000, starting at that address memory will look like this:

C3 AD 02 05

When the call is executed the stack will contain 6003 which is the first byte after SETUP's address. If "POP H" is executed HL will contain 6003 and a "MOV A,M" will put 05 into the A register. The question is then how to get back to where the "CALL" originated. As SETUP goes along it keeps sending out data and increasing HL by one until it hits a zero byte. This is the signal that there is no more data. All that is left is to put the HL register into the program counter with a "PCHL" instruction and the program comes back to the byte after the last byte of data! Thank some unknown wizard at PolyMorphic for writing that gem.

Even though the USART had data put into it, it still will not run and it is just as well because there is still more work to be done. The file must have a name and a size. The name is a fairly straight-forward process. The area where it is to go is set to spaces and then data is filled in, letter by letter. It is

RUBOUT's job to back up over any character that is to be erased but not to go past the start of FILENM. When there is no room for more letters or when a <CR> is typed the file name is complete and the starting addrsss is requested.

All addresses are brought in by HEXC, a monitor subroutine that puts a valid address in the HL register, the number of digits typed in are in the C register and the last character seen is in the B register. HEXC also displays the data as it comes in. A check is made in AUTODUMP to make sure that HEXC ended with a <CR>. Anything else is invalid and the address is requested again. This allows errors to be corrected without restarting AUTODUMP. Once the starting and ending addresses have been entered the length of the file is computed by subtracting the ending address from the starting address. This takes place shortly after GETEND and the start of the subtraction is "MOV A,L". Ordinarily the carry flag would have to be cleared before the subtraction can start but the "JNZ GETEND" instruction insures that the carry flag is cleared to zero. The program can only transfer here when CPI 0DH sets the zero flag and at the same time clears the carry flag. This may seem a trivial thing but it is important to know that this little trick saves two bytes of code. It would otherwise take "STC" (SeT Carry) and "CMC" (CoMpliment Carry) commands to clear the carry flag. With the subtraction completed the length of the file is set. By figuring the length it is possible to test for the end of the dump without having to test two address bytes. When FILESZ is less than 100H the last block has been written and the dump is over.

After all of the file size data has been collected the load point and execution address are requested, then it's time to decide how many copies are to be made. Each item is a simple type-in and their values are not really used until the file is reloaded. For that reason discussion about them will be delayed until the tape writing begins. What remains, then, is the banner. If a simple <CR> is typed in, a one byte long banner is created and the whole matter is ignored. On the other hand, if data is typed in, it will be displayed when the file is reloaded. Remember that short files read in with POLYPHASE do not take long to load and the banner may not be displayed for long.

Each byte typed in belongs to one of three classes. The most common class is a simple display character. It is placed in the table after being ORed with the B register. If B has a 20H in it, the character becomes a lower case letter, otherwise it remains as it was. The next possible class is the "flip" character and it may be any character you wish, excluding CTRL characters. Its function is to change B from 00H to 20H or vice versa and this "flips" letters from upper to lower case. Because the "flip" character is not stored, it is possible to "flip" as often as one wishes. Note, however, that some characters like <CR> do not survive being "flipped". The final item that may be entered is <CTRL/X>. This is the character that causes the screen and banner table to be cleared and allows new data to be entered. This does not clear the addresses or number of copies counter. It does mean that 255 characters are again free and that all old data was removed. The <CTRL/X> is the best way to correct the banner because <RUBOUT> does not remove data, it only covers it up and takes up banner space as well.

With the banner loaded the program waits for a type-in that indicates the tape is ready. This allows time to change cassettes and check the recorder. On receiving the type-in the size of the banner is checked. If it is one byte long the program goes on to dump data; if not, the banner is dumped to tape. The dump procedure is similar for the banner and data records. Each record contains 16 <SYNC> characters, a <SOH> (Start Of Header), the file's name, the number of bytes in the data area, two zero bytes if the record is a comment otherwise the starting address for loading the data, 01 for a comment or 00 for a core image, the checksum for the header, the data or banner, and the data's checksum.

The program transfers to DUMP and it displays the address being dumped out. The inter-record gap is generated by GAP while SYNC outputs the <SYNC>'s. These characters are used to ensure that the USART knows how fast the data is moving so that the USART can either say it "knows" what the data rate is or that something is wrong. Notice that there is no "OUT" instruction in SYNC, the data is actually sent out by a call to TAPOT. We will discuss it after the header has been described. After completing SYNC, a <SOH> is sent to indicate that the data that follows is the header proper. The A register is given the number of characters to be output from the header and HL is set to point at the beginning of the header data, FILENM. Since the header will be concluded with a checksum, CHKSM1 is called. Notice that all of the data for the header is grouped together in the constant area at the beginning of the program.

Checksumming allows the verification of data and, although far from perfect, gives some measure of assurance that what is received is what was sent. The method used here is based on modulus 256 arithmetic. The data ranges from 00 to FF (0 to 255) and if each byte is added to the sum of all the previous bytes we have the basis of the checksum. Naturally the sum will go back to zero if 1 is added to FF, but for the present scheme that is acceptable. When the last item has been processed, the 1's complement is found and 1 added to that to give the 2's complement of the checksum. To verify that all has gone well, take the sum of the data and add the checksum to the summed data. The correct answer is zero and anything else is a sign that something is wrong. Look at CHKSM1 and CHKSM2 and you will see the steps described above. Keep in mind that the C register keeps track of the number of bytes yet to be written. When C finally equals zero, program flow falls through to do the complement and on to TAPOT which writes the checksum onto the tape.

TAPOT saves HL, looks at TBUFF, and saves PSW. TBUFF is a two byte area that holds a status byte and a data byte. When TISR services the USART, it will set TBUFF to zero indicating that putting data into TBUFF+1 will not damage any existing data. The three instructions "TAPOT1 MOV A,M/ORA A/JNZ TAPOT 1" may seem to be a closed loop but HL is pointing at TBUFF. TISR can put a zero into TBUFF so it is possible to exit the loop. At this point there are probably readers who are trying to figure out how the program gets to TISR. Up at the very beginning of AUTODUMP, at label ENTER, the address for TISR was stored in SRA4. When the USART generates an interrupt, the CPU will "CALL" the address at SRA4 effectively calling TISR.

TISR clears TBUFF and then puts TBUFF+1 into A. With the data in A, an "OUT 0" sends the data off to the USART and the tape. The stack is restored and control returns to where the CPU was when it saw the interrupt. With TBUFF set

to zero, the data is loaded into TBUFF+1, TBUFF is set to one, the data pointer is restored to HL, and control is returned from TAPOT.

With the header written out, the data itself must be dumped out to tape. The process is controlled by DMPFULL and DMPPART. A check of the data block's length is made. If the block is 100 hex bytes long, DMPFULL causes the bytes to be output along with a checksum. This process continues until the size of the remaining data is under 100 hex bytes. At that time there is a "JMP" to DMPPART. Notice that the program falls directly through from DMPPART to TAPOT1. At the end of TAPOT1 the "RET" causes a return to the "MVI A,2" after the "CALL" to DMPFULL. Here the banner size is checked. A one byte banner causes a simple end of file record to be written. A longer banner causes the execution address to be put into an Auto-Execute record. If the number of copies requested is zero the program goes back to ENTER. If not, the counters are reset to their values at the start of the dump and the program goes to NXTCPY.

That's all there is to AUTODUMP. It is a program that scans memory in 255 byte chunks which can be passed to tape. Now that you know how the files are written, why not try to write files in other formats? Since you know how the files are constructed it should be easier to write a program to read those files. There is no reason why your POLY88 can't read other tape formats.

MICROSOFT'S COBOL-80

News Release *Received: 78 April 13*

Microsoft is the first COBOL for 8080/Z-80/8085 microprocessor systems. COBOL-80 conforms to the 1974 ANSI standard, thus giving users immediate access to programs already written in COBOL. All Level 1 features and the most useful Level 2 options for the "Nucleus" and for Sequential, Relative and Indexed file handling facilities are included. Additionally, Level 1 Table Handling, Library and Inter-Program Communication facilities are provided. Of the advanced Level 2 features, Microsoft has included the verbs STRING, UNSTRING, COMPUTE, SEARCH and PERFORM (varying/until), along with condition specification by way of condition-names, compound conditions and abbreviated conditions. COBOL-80 allows a packed decimal data representation to conserve memory on floppy disks.

The COBOL-80 system consists of two packages: a compiler for translating source code into relocatable object code, and a runtime system containing standard routines needed by the object code at execution time. The whole system may be run in less than 32K bytes. Rate of compilation is 250 lines per minute. Complete documentation is supplied with the system or may be purchased separately for $20. COBOL-80 is available off the shelf to run under the CP/M and ISIS-11 operating systems for $750 per copy. Microsoft, 300 San Mateo, NE, Suite 819, Albuquerque, NM 87108; (505) 262-1486.

```
;AUTODUMP V2.4 WITH INPUT CORRECTION & RELOCATION
;4.0 MONITOR REFERENCES
02AD            SETUP   EQU     02ADH   ;INITIALIZES USART
03AA            HEXC    EQU     03AAH   ;ACCEPTS HEX CHARS FROM KEYBD
03D1            DEOUT   EQU     03D1H   ;DISPLAYS DATA IN DE REG.
0C08            TBUFF   EQU     0C08H   ;OUTPUT STATUS & BUFFER
0C16            SRA4    EQU     0C16H   ;SERVICE ROUTINE ADDRESS
0C20            WH0     EQU     0C20H   ;WORMHOLE 0, KEYBD I/P
0C24            WH1     EQU     0C24H   ;WORMHOLE 1, CRT O/P
                ;
6000 C31F61     START   JMP     ENTER   ;DEFINES START OF PGM
6003            FILESZ  DS      2       ;LENGTH OF FILE TO BE DUMPED
                ;HEADER TEXT AREA - DO NOT MODIFY!
6005            FILENM  DS      8       ;FILE NAME FITS HERE. 8 CHARS MAX
600D            RECNUM  DS      2       ;RECORD NUMBER
600F            BLKSZ   DS      1       ;SIZE SCRATCH PAD
6010            LODSCR  DS      2       ;LOAD ADDRESS SCRATCH PAD
6012            RECTYP  DS      1       ;RECORD TYPE BEING WRITTEN
                ;END OF HEADER TEXT AREA
6013            BNRTBL  DS      100H    ;BUFFER FOR COMMENT
6113            BNRSZ   DS      1       ;SIZE OF COMMENT
6114            EXCADD  DS      2       ;BEGIN PGM EXECUTION HERE
6116            STRADD  DS      2       ;STARTING ADDRESS OF DUMP
6118            ADDSCR  DS      2       ;ADDRESS SCRATCH PAD
611A            LODADD  DS      2       ;STARTING LOADER ADDRESS
611C            LENSCR  DS      2       ;# OF BYTES TO BE DUMPED
611E            COPYNO  DS      1       ;# OF COPIES
                ;
611F 21F662     ENTER   LXI     H,TISR  ;GET USART SERVICE ROUTINE ADD.
6122 22160C             SHLD    SRA4    ;INSERT TISR INTO SRA TABLE
6125 3E0C               MVI     A,0CH   ;CLEAR CRT WITH <FORM FEED>
6127 CD240C             CALL    WH1
612A 21F663     GETMD   LXI     H,MDTXT ;POINT TO TEXT TABLE
612D CD6363             CALL    SHOW    ;REQUEST MODE
6130 CD200C             CALL    WH0     ;GET RESPONSE
6133 FE42               CPI     'B'
6135 CA4D61             JZ      BYTE    ;SET BYTE MODE
6138 FE50               CPI     'P'
613A C23061             JNZ     GETMD   ;BAD I/P, TRY AGAIN
613D CD240C             CALL    WH1     ;ECHO I/P
6140 CDA902             CALL    SETUP   ;SET UP USART FOR POLY MODE
6143 05AA400C           DB      05H,0AAH,40H,0CH,0E6H,0E6H,0    ;SETUP DATA
6147 E6E600
614A C35861             JMP     GETNM   ;GET TAPE NAME
614D CD240C     BYTE    CALL    WH1     ;ECHO I/P
6150 CDA902             CALL    SETUP   ;SET UP USART FOR BYTE MODE
6153 06AA40CE           DB      06H,0AAH,40H,0CEH,0     ;SETUP DATA
6157 00
6158 210000     GETNM   LXI     H,0     ;CLEAR HL
615B 220D60             SHLD    RECNUM  ;RESET RECORD NUMBER
615E 0E08               MVI     C,08H   ;USE C TO COUNT BACK FROM 8
6160 210C60             LXI     H,FILENM+7      ;SET HL TO END OF FILENM
6163 3620       NAMCLR  MVI     M,20H   ;PUT <SP> IN A BYTE OF FILENM
6165 2B                 DCX     H       ;HL=HL-1 & COUNT BACK TO FILENM
6166 0D                 DCR     C
6167 C26361             JNZ     NAMCLR  ;IF C<>0 THEN TRY AGAIN
616A 217D63             LXI     H,NMTXT ;POINT TO NAME TEXT
616D CD6363             CALL    SHOW    ;REQUEST FILE NAME
6170 210560             LXI     H,FILENM        ;HL POINTS TO FILE NAME FIELD
6173 0E08               MVI     C,8     ;ACCEPT UP TO 8 CHARS
6175 CD200C     GETNM1  CALL    WH0     ;GET CHAR
6178 CD240C             CALL    WH1     ;ECHO CHAR
617B FE7F               CPI     7FH     ;CHAR = <RUBOUT>?
617D CA4F63             JZ      RUBOUT  ;YES, RUB OUT CHAR
6180 FE0D               CPI     0DH     ;CHAR = <CR>?
```

```
6182 CA8B61            JZ      GETSTAD ;YES, GET STARTING ADDRESS OF DUMP
6185 77                MOV     M,A     ;NO, MOVE CHAR TO FILENM
6186 23                INX     H
6187 0D                DCR     C
6188 C27561            JNZ     GETNM1  ;IF C<>0, GET NEXT CHAR
618B AF        GETSTAD XRA     A       ;CLEAR ACC.
618C 321260            STA     RECTYP  ;CLEAR RECORD TYPE FIELD
618F 218B63            LXI     H,STTXT ;POINT TO START ADD TEXT
6192 CD6363            CALL    SHOW    ;REQUEST START ADDRESS
6195 CDAA03            CALL    HEXC    ;GET HEX ADDRESS
6198 78                MOV     A,B     ;GET LAST CHAR
6199 FE0D              CPI     0DH     ;CHAR = <CR>?
619B C28B61            JNZ     GETSTAD ;NO, RETRY START ADDRESS
619E 221661            SHLD    STRADD  ;MOVE DATA IN HL TO START ADD FLD
61A1 EB                XCHG            ;HL TO DE TO COMPUTE FILE SIZE
61A2 219963    GETEND  LXI     H,ENDTXT        ;POINT TO END ADD TEXT
61A5 CD6363            CALL    SHOW    ;REQUEST ENDING ADDRESS
61A8 CDAA03            CALL    HEXC    ;GET HEX ADDRESS
61AB 78                MOV     A,B     ;GET LAST CHAR
61AC FE0D              CPI     0DH     ;CHAR = <CR>?
61AE C2A261            JNZ     GETEND  ;NO, RETRY ENDING ADDRESS
61B1 7D                MOV     A,L     ;COMPUTE FILE LENGTH IN BYTES
61B2 93                SUB     E       ;HL = HL - DE
61B3 6F                MOV     L,A
61B4 7C                MOV     A,H
61B5 9A                SBB     D
61B6 67                MOV     H,A
61B7 220360            SHLD    FILESZ  ;STORE LENGTH OF DUMP FILE
61BA 221C61            SHLD    LENSCR  ;LENGTH TO LENSCR
61BD 21A763    GETLOD  LXI     H,LODTXT        ;POINT TO LOAD ADD TXT
61C0 CD6363            CALL    SHOW    ;REQUEST LOADING ADDRESS
61C3 CDAA03            CALL    HEXC    ;GET HEX ADDRESS
61C6 78                MOV     A,B     ;GET LAST CHAR
61C7 FE0D              CPI     0DH     ;CHAR = <CR>?
61C9 C2BD61            JNZ     GETLOD  ;NO, RETRY LOAD ADDRESS
61CC 221A61            SHLD    LODADD  ;SAVE LOADING ADDRESS
61CF 221060            SHLD    LODSCR  ;RESET LOAD ADD IN HEADER
61D2 21B563    GETEXC  LXI     H,EXCTXT        ;POINT TO EXEC ADD TXT
61D5 CD6363            CALL    SHOW    ;REQUEST ADDRESS FOR EXECUTION
61D8 CDAA03            CALL    HEXC    ;GET HEX ADDRESS
61DB 78                MOV     A,B     ;GET LAST CHAR
61DC FE0D              CPI     0DH     ;CHAR = <CR>?
61DE C2D261            JNZ     GETEXC  ;NO, RETRY EXECUTE ADDRESS
61E1 221461            SHLD    EXCADD  ;SAVE EXEC ADDRES
61E4 21C363    GETCPY  LXI     H,CPYTXT        ;POINT TO # OF COPIES TEXT
61E7 CD6363            CALL    SHOW    ;REQUEST # OF COPIES IN HEX
61EA CDAA03            CALL    HEXC    ;MAX NUMBER = FF
61ED 78                MOV     A,B     ;GET LAST CHAR
61EE FE0D              CPI     0DH     ;CHAR = <CR>?
61F0 C2E461            JNZ     GETCPY  ;NO, RETRY COPY NO
61F3 7D                MOV     A,L     ;LOW BYTE IS # OF COPIES
61F4 321E61            STA     COPYNO
61F7 21D863            LXI     H,CMTTXT        ;POINT TO COMMENT I/P RULES
61FA CD6363            CALL    SHOW    ;DISPLAY RULES
61FD 0E01      BNRST   MVI     C,1     ;SET CHAR CNT=1
61FF 211360            LXI     H,BNRTBL        ;CURRENT CMNT TBL PNTR IN HL
6202 221861            SHLD    ADDSCR  ;RESET ADDRESS SCRATCH PAD TO BNRTBL
6205 0600              MVI     B,0
6207 CD200C    MORCHR  CALL    WH0     ;GET CHAR FOR CMNT
620A FE04              CPI     4       ;CHAR = <CTRL/D>?
620C CA3A62            JZ      BNRDON  ;YES, NO MORE CMNT
620F FE18              CPI     18H     ;CHAR = <CTRL/X>?
6211 C22462            JNZ     CHGFLG  ;NO, CHECK FOR FLAG CHANGE
6214 3E0C      CTRLX   MVI     A,0CH   ;A = <FORM FEED>
6216 CD240C            CALL    WH1     ;CLEAR SCREEN
6219 3E20              MVI     A,20H   ;A = <SP>
621B 77        CTRLX1  MOV     M,A     ;CLEAR BANNER TABLE
621C 2B                DCX     H
621D 0D                DCR     C
621E C21B62            JNZ     CTRLX1
6221 C3FD61            JMP     BNRST   ;BANNER CLEARED
6224 FE40      CHGFLG  CPI     '@'     ;CHAR = <@>?
6226 C23062            JNZ     LODCHR  ;NO, STORE CHAR IN CMNT TBL
6229 78                MOV     A,B     ;YES, FLIP LOWER/UPPER CASE FLAG
622A EE20              XRI     20H
622C 47                MOV     B,A     ;STORE FLAG
622D C30762            JMP     MORCHR  ;GET MORE COMMENT
6230 B0        LODCHR  ORA     B       ;CONVERT LOWER CASE TO UPPER IF B=20H
6231 77                MOV     M,A     ;STORE CHAR
6232 CD240C            CALL    WH1     ;SHOW CHAR
6235 23                INX     H       ;POINT TO NEXT CHAR
6236 0C                INR     C       ;ADD 1 TO CHAR CNT
6237 C20762            JNZ     MORCHR  ;MORE ROOM LEFT, GET MORE
623A 360D      BNRDON  MVI     M,0DH   ;PUT <CR> IN BNRTBL
623C 79                MOV     A,C     ;MOVE COMMENT SIZE TO BNRTBL
623D 321361            STA     BNRSZ   ;SAVE SIZE OF CMNT TBL
6240 21F363    NXTCPY  LXI     H,RDYTXT        ;POINT TO READY TEXT
6243 CD6363            CALL    SHOW    ;TELL USER TO READY TAPE
6246 CD200C            CALL    WH0     ;WAIT FOR ANY CHAR
6249 3A1361            LDA     BNRSZ   ;GET SIZE OF BANNER
624C FE01              CPI     1       ;IS BANNER EMPTY?
624E CA6162            JZ      STDMP   ;YES, SKIP PAST BANNER DUMP
6251 320F60            STA     BLKSZ   ;PUT SIZE IN SIZE SCRATCH PAD
6254 3E01              MVI     A,1     ;PUT COMMENT FLAG IN RCD TYPE
6256 321260            STA     RECTYP
6259 CD0663            CALL    DUMP    ;OUTPUT BANNER
625C 3E00              MVI     A,0     ;SET RCD TYPE TO ABSOLUTE BINARY MODE
625E 321260            STA     RECTYP
6261 2A1661    STDMP   LHLD    STRADD  ;GET STARTING ADDRESS
6264 221861            SHLD    ADDSCR  ;PUT IN ADDRESS SCRATCH PAD
6267 CDAC62            CALL    DMPFULL ;DUMP OUT DATA
626A 3E02              MVI     A,2     ;SET DUMP MODE TO END
626C 321260            STA     RECTYP
626F 3A1361            LDA     BNRSZ   ;GET BANNER SIZE
6272 FE01              CPI     1       ;EMPTY BANNER?
6274 CA8262            JZ      CLSDMP  ;YES, CLOSE DUMP FILE W/O AUTO-EXEC.
6277 3E03              MVI     A,03    ;NO, SET RCD TYPE TO AUTO-EXEC.
6279 321260            STA     RECTYP
627C 2A1461            LHLD    EXCADD  ;PUT EXECUTION ADDRESS IN HEADER
627F 221060            SHLD    LODSCR
6282 3E01      CLSDMP  MVI     A,1     ;SET RECORD SIZE TO ZERO LENGTH
6284 320F60            STA     BLKSZ
6287 CD0663            CALL    DUMP    ;CLOSE FILE
628A 211E61            LXI     H,COPYNO        ;GET COPY COUNT
628D 35                DCR     M       ;COPYNO=COPYNO-1
628E CA1F61            JZ      ENTER   ;IF M=0 THEN DONE, GO RESTART
6291 2A1C61            LHLD    LENSCR  ;NOT FINISHED, REMEMBER FILE SIZE
6294 220360            SHLD    FILESZ
6297 2A1A61            LHLD    LODADD  ;RESET LOAD ADDRESS SCRATCH PAD
629A 221060            SHLD    LODSCR
629D 211360            LXI     H,BNRTBL        ;POINT TO TOP OF BANNER TBL
62A0 221861            SHLD    ADDSCR  ;SAVE ADDRESS
62A3 210000            LXI     H,0000  ;SET RECORD NUMBER BACK TO 0
62A6 220D60            SHLD    RECNUM
62A9 C34062            JMP     NXTCPY  ;START DUMPING NEXT COPY
62AC 210460    DMPFULL LXI     H,FILESZ+1      ;GET HI BYTE OF SIZE
62AF 7E                MOV     A,M
62B0 B7                ORA     A       ;A=0?
62B1 CACA62            JZ      DMPPART ;YES, SIZE<100H
62B4 35                DCR     M       ;NO, DECREMENT SIZE BY 100H
62B5 AF                XRA     A       ;A=0
62B6 320F60            STA     BLKSZ   ;PUT (1)00H IN BLOCK SIZE
62B9 CD0663            CALL    DUMP    ;DUMP BLOCK OF DATA ≥55 BYTES LONG
62BC 2A1861            LHLD    ADDSCR  ;GET DATA POINTER
62BF 24                INR     H       ;INCREMENT DATA POINTER BY 100H
62C0 221861            SHLD    ADDSCR
62C3 211160            LXI     H,LODSCR+1      ;INCREMENT LOAD POINTER BY 100H
62C6 34                INR     M
62C7 C3AC62            JMP     DMPFULL
```

```
62CA 2B       DMPPART DCX     H       ;LO BYTE IS BLOCK SIZE
62CB 7E               MOV     A,M
62CC 320F60           STA     BLKSZ
62CF C30663           JMP     DUMP    ;DUMP BLOCK OF BLKSZ BYTES
62D2 0600     CHKSM1  MVI     B,0     ;CLEAR CHECKSUM REGISTER
62D4 4F               MOV     C,A     ;MOVE BLOCK SIZE TO C
62D5 7E       CHKSM2  MOV     A,M     ;GET BYTE OF DATA
62D6 23               INX     H       ;HL=HL+1
62D7 F5               PUSH    PSW     ;PUSH DATA ONTO STACK
62D8 80               ADD     B       ;ADD DATA TO SUM OF PREVIOUS BYTES
62D9 47               MOV     B,A
62DA F1               POP     PSW     ;DONE ADDING, RESTORE DATA
62DB CDE562           CALL    TAPOT   ;OUTPUT DATA TO TBUFF
62DE 0D               DCR     C       ;DECREMENT BLOCK SIZE BY 1
62DF C2D562           JNZ     CHKSM2  IF C<>0, GO DUMP NEXT BYTE
62E2 78               MOV     A,B     ;BLOCK DONE, COMPUTE CHECKSUM
62E3 2F               CMA             ;GET 1'S COMPLIMENT OF ACC.
62E4 3C               INR     A       ;A=A+1 CONVERTS ACC TO 2'S COMPLIMENT
              ;WAIT FOR STATUS BYTE TO CLEAR & O/P ACC. TO USART
62E5 E5       TAPOT   PUSH    H       ;STORE DATA POINTER
62E6 21080C           LXI     H,TBUFF ;POINT TO USART XMIT BUFFR
62E9 F5               PUSH    PSW     ;SAVE DATA TO CHECK USART STATUS
62EA 7E       TAPOT1  MOV     A,M     ;WAIT FOR INTERRUPT TO ZERO TBUFF
62EB B7               ORA     A       ;DID TBUFF CLEAR?
62EC C2EA62           JNZ     TAPOT1  ;IF NOT, TRY AGAIN
62EF 23               INX     H       ;TBUFF = 0, POINT TO TBUFF+1
62F0 F1               POP     PSW     ;RESTORE DATA
62F1 77               MOV     M,A     ;PUT ACC. IN TBUFF+1
62F2 2B               DCX     H       ;POINT TO TBUFF, AGAIN
62F3 34               INR     M       ;SET TBUFF TO 1, SAYS DATA IS LOADED
62F4 E1               POP     H       ;RESTORE DATA POINTER
62F5 C9               RET
62F6 F3       TISR    DI              ;KEEP OUT CHAOS
62F7 AF               XRA     A       ;CLEAR STATUS FLAG
62F8 32080C           STA     TBUFF
62FB 3A090C           LDA     TBUFF+1 ;GET DATA FROM BUFFER - PUT IT IN ACC.
62FE D300             OUT     0       ;SEND IT TO USART THROUGH PORT 0
6300 E1               POP     H       ;RESTORE STACK
6301 D1               POP     D
6302 C1               POP     B
6303 F1               POP     PSW
6304 FB               EI              ;ALLOW INTERRUPTS
6305 C9               RET
6306 3E21     DUMP    MVI     A,21H   ;ENABLE USART THROUGH COMMAND PORT
6308 D301             OUT     1
630A 2A1861           LHLD    ADDSCR  ;GET ADDRESS OF CURRENT BLOCK
630D EB               XCHG            ;AND DISPLAY IT
630E CDD103           CALL    DEOUT
6311 CD5D63           CALL    CRLF    ;SCROLL 1 LINE
6314 21FF0F           LXI     H,0FFFH ;AND WAIT UNTIL H=0
              ;GENERATE INTER-RECORD GAP
6317 2B       GAP     DCX     H       ;COUNT BACKWARDS
6318 7C               MOV     A,H
6319 B7               ORA     A
631A C21763           JNZ     GAP     ;IF NOT 0, DO SOME MORE
631D 0E10             MVI     C,10H   ;COUNT OUT 16 PASSES THROUGH LOOP
631F 3EE6             MVI     A,0E6H  ;PUT <SYNC> IN ACC.
6321 CDE562   SYNC    CALL    TAPOT   ;OUTPUT <SYNC>
6324 0D               DCR     C
6325 C22163           JNZ     SYNC    ;UNTIL C=0
6328 3E01             MVI     A,1     ;OUTPUT <SOH>
632A CDE562           CALL    TAPOT
632D 3E0E             MVI     A,0EH   ;SET ACC. TO 14 CHARS OF RECORD HEADER
632F 210560           LXI     H,FILENM        ;POINT TO START OF HEADER
6332 CDD262           CALL    CHKSM1  ;OUTPUT HEADER TO USART
6335 3A0F60           LDA     BLKSZ   ;GET SIZE & ADDRESS FOR REST OF BLOCK
6338 2A1861           LHLD    ADDSCR
633B CDD262           CALL    CHKSM1  ;OUTPUT BLOCK TO USART
633E 210D60           LXI     H,RECNUM        ;ADD 1 TO RECORD NUMBER
6341 34               INR     M
6342 AF               XRA     A       ;FLUSH OUT USART PIPELINE
6343 CDE562           CALL    TAPOT   ;WITH ZERO BYTES
6346 CDE562           CALL    TAPOT
6349 CDE562           CALL    TAPOT
634C D301             OUT     1       ;PUT 0 IN USART STATUS, SHUTTING OFF
634E C9               RET             ;USART AND MOTOR
              ;RUB OUT CHARACTER FROM FILE NAME
634F 3E20     RUBOUT  MVI     A,20H   ;PUT <SP> IN FILENM
6351 77               MOV     M,A
6352 79               MOV     A,C     ;AT FIRST CHAR IN FILENM?
6353 FE08             CPI     8
6355 CA7561           JZ      GETNM1  ;YES, DON'T BACK UP
6358 0C               INR     C       ;NO, BACK UP 1 CHAR
6359 2B               DCX     H
635A C37561           JMP     GETNM1  ;GO GET NEW CHAR
              ;OUTPUT CARRIAGE RETURN
635D 3E0D     CRLF    MVI     A,0DH   ;A = <CR>
635F CD240C           CALL    WH1     ;DISPLAY <CR>
6362 C9               RET
              ;DISPLAY TEXT STRING TERMINATED WITH ZERO BYTE
6363 C5       SHOW    PUSH    B       ;SAVE B BEFORE SHOWING STRING
6364 7E       SHOW1   MOV     A,M     ;GET CHAR
6365 CD240C           CALL    WH1     ;DISPLAY CHAR
6368 23               INX     H       ;POINT TO NEXT CHAR
6369 B7               ORA     A       ;TEXT DONE?
636A C26463           JNZ     SHOW1   ;A<>0 THEN TRY AGAIN
636D C1               POP     B       ;YES, RESTORE B AND LEAVE
636E C9               RET
              ;MESSAGE TEXT STRINGS
636F 0D544150 MDTXT   DB      0DH,'TAPE  MODE: ',0
6373 4520204D
6377 4F44453A
637B 2000
637D 0D544150 NMTXT   DB      0DH,'TAPE  NAME: ',0
6381 4520204E
6385 414D453A
6389 2000
638B 0D535441 STTXT   DB      0DH,'START ADDR: ',0
638F 52542041
6393 4444523A
6397 2000
6399 0D454E44 ENDTXT  DB      0DH,'END   ADDR: ',0
639D 20202041
63A1 4444523A
63A5 2000
63A7 0D4C4F41 LODTXT  DB      0DH,'LOAD  ADDR: ',0
63AB 44202041
63AF 4444523A
63B3 2000
63B5 0D455845 EXCTXT  DB      0DH,'EXEC  ADDR: ',0
63B9 43202041
63BD 4444523A
63C1 2000
63C3 0D4E4F2E CPYTXT  DB      0DH,'NO. COPIES',0DH,'IN HEX: ',0
63C7 20434F50
63CB 4945530D
63CF 494E2048
63D3 45583A20
63D7 00
63D8 0D434F4D CMTTXT  DB      0DH,'COMMENT <CTRL/D TO END):',0DH,0
63DC 4D454E54
63E0 20284354
63E4 524C2F44
63E8 20544F20
63EC 454E4429
63F0 3A0D00
63F3 0D524541 RDYTXT  DB      0DH,'READY TAPE, AND PRESS ANY LETTER: ',0
63F7 44592054
63FB 4150452C
63FF 20414E44
6403 20505245
6407 53532041
640B 4E59204C
640F 45545445
6413 523A2000
6417 00       FINIS   DB      0
```

HIGH SPEED CASSETTE I/O
FOR THE KIM-1

BY THEODORE E. BRIDGE
54 Williamsburg Dr.
Springfield, MA 01108

This high speed cassette tape handler will load or dump at 12 times the speed of KIM-1 or twice the speed of the MICRO-ADE[1] Editor-Assembler. It replaces and overlays the editor routines CREAD and CWRITE at the end of the third 4K block of memory (2EA9-3000).

If you are not now using the MICRO-ADE[1] Editor-Assembler, you should consider it. I think it is well worth the $25 I paid for it.

Stand-alone operation is permitted with this high speed tape handler. You may call it from any program as follows:

```
20  A9  2E    JMS CREAD    Cassette Read
20  54  2F    JMS CWRITE   Cassette Write
```

If your program is to call CWRITE, be sure it first loads the starting and ending addresses as given on page 47 of the KIM-1 User Manual. This is just as if you were dumping with KIM-1 before starting at 1800.

You can use these high speed routines to dump from the keypad. Load the addresses as with KIM-1, but instead of starting at address 1800, start at address 2F54. You can also load from keypad control. Instead of starting at 1873 with KIM-1, start at 2EA9. This loader will save starting and ending addresses ready for you to copy the file on cassette by starting at 2F54.

Trade-offs

You pay a price for the increased speed. When you use the MICRO-ADE editor, with its HYPERTAPE[2] tape dump, you need a better cassette recorder than you would with KIM-1. I have had good results with my Realistic[3] seventy dollar recorder.

When you try to double the speed using these routines, you lose compatibility with KIM-1 and you lose the search capability of the monitor. You always load the next program on the tape rather than hunt for the (ID) you want. You need a meter on your recorder so that you can position the read head just in front of the program you want to load. Also, you must keep a journal showing the cassette number and the meter reading for all data you record on tape. You really ought to do this anyway.

Loss of the search capability is not altogether a disadvantage. If you have only one copy of a program on tape, a blip on the tape may prevent your loading with the MICRO-ADE loader. But with this high speed loader, even if you flag a checksum error, most of the program will probably load properly. I have found it helpful. Either way, you really need two cassette recorders: one for input, and another for output.

You pay yet another price when you use these high speed routines during an assembly. When you load the object tape, it will reside at address 0200 where it was assembled instead of at the ORG address where it belongs. After loading, you must use the BLOCK MOVE command to put it in its proper location. After all units of your program have been properly relocated, you must perform a SAVE operation to get a usable object tape.

Apart from making a quicker compilation when using these routines (even after relocating as described above), you can also recover more quickly from checksum errors encountered while reading the source tape during compilation. I have found them very convenient for general housekeeping as well as for compilation.

Description

These routines are similar to the MICRO-ADE HYPERTAPE[2] routines that they replace. They simply gulp the entire 8 bit byte at once instead of taking it in two nybbles. Thus, they go twice as fast. The CREAD routine cannot look for an end-of-data symbol ("/" = 2F) to terminate the load as this might appear anywhere in the data being loaded. Instead, it reads the start and end addresses at the beginning of the tape. It terminates the load when the loading address (VEB+1, 2) equals or exceeds the end address. At the end, the display is lit; so that you can see if a checksum error has occurred; and if you have the right ID.

Since we are not loading ASCII characters, we lose the validity check on each character as is made in the HYPERTAPE program. We must depend on the checksum error to flag all errors. This has not been a problem for me. Actually, I think it is easier to recover from a checksum error when using these

high speed routines than when using the HYPERTAPE routines.

At the end of a tape load, this high speed loader will halt with the LED display showing:

00F9 (ID) if there was no checksum error, and
17F9 (ID) if there was a checksum error.

You look to see if the ID is the one you want. If you depress (C = Continue) you will return to the calling program. If you depress (B = Bad tape) you will read the next program on tape. Of course, you can rewind to try again on the same data if you wish. The VU-TAPE[2] featured by Jim Butterfield is incorporated in this high speed loader. With it you can see the loader lock on to the sync pulses when it starts to load.

At the end of a tape dump, you will halt with the LED display lit. If you depress (C = Continue), you will return to the calling program. If you depress (B = Begin) you will load the next program on the input tape. If you depress (RS, AD, 2F54, GO), you will dump another copy of the data to the output cassette.

Remote program control of the cassette recorders is provided for in these high speed routines. I use the HELP RELAY PACKAGE[4]. Application pin 11 activates the input cassette, and pin 12 activates the output cassette. If you are not now using automatic tape control, you should consider it. Later, we will describe two patches: one to permit manual tape control, and another to slow down the speed if your cassette recorder can't handle the high speed tapes.

Tape problems are not to be taken lightly. I have adopted the following procedures to try to minimize these problems.

1. Always rewind before ejecting a tape.
2. Stop the recorder before removing remote control plug.
3. Use short tapes. Short tapes can be obtained from PYRAMID DATA SYSTEMS[5] – five 10 minute tapes for $3.00, or five 20 minute tapes for $3.25.
4. You need a medium quality cassette recorder. Robert Tripp[4] reports good success with Panasonic RQ212 DAS, and Radio Shack Realistic CTR 39 model 14-819.

To load these high speed routines from cassette, you use the KIM-1 tape loader at 1873. Therefore, you must keep a copy on cassette in HYPERTAPE or KIM-1 format. I can furnish such a cassette for $4.00 or a paper tape for $2.00.

Patches

This patch for the Micro-Ade Assembler-Editor will link with the high speed routines:

```
2E97 4C A9 2E    JMP CREAD
2E9A 4C 4F 2F    JMP CWRITE
```

This manual control patch will permit you to operate without relays for program control of the cassette recorder:

```
2EA9 A9 1C       LDAIM $1C
2EAB 85 FB       STA POINTH
2EAD 85 FA EA    STA POINTL
2EBO 20 24 2F    JSR DISP
```

On entering CREAD, this patch will display 1C1CXX on the LEDS to remind you to start the input cassette immediately after you depress C to continue with the reading of the tape.

At the end of the load, the CREAD routine will display: 00F9 (ID) on the LEDs if there was no checksum error, and 17F9 (ID) if there was a checksum error. At this time, you must remember to stop the input cassette before depressing C to continue, or B to reread the tape.

On entering CWRITE, this patch will display OCOCXX to remind you to start the output cassette and to wait a few seconds before depressing C to continue with the dump:

```
2F56 85 FB       STA POINTH
2F58 85 FA EA    STA POINTL
2F5B 20 24 2F    JSR DISP
```

Before leaving the CWRITE routine, this patch will display COFF on the LEDs to remind you to turn off the output cassette beofre depressing C to return to the calling program:

```
2FA4 A9 CO       LDAIM $CO
2FA6 85 FB       STA POINTH
2FA8 A9 FF       LDAIM $FF
2FAA 85 FA       STA POINTL
2FAC 4C 24 2F    JMP DISP
```

A patch to lower the recording speed is given at the end of the listing. This patch is necessary only if your cassette recorder cannot handle the higher speed.

Conclusion

I think it is very worthwhile to speed up the tape operations by use of these routines. I recently wrote a program to compare the dental record of a corpse with up to 60,000 missing person records to seek a match. In writing this program, I was very conscious of the need for faster cassette reading. With these routines, it would take 4 hours to read 60,000 records. With HYPERTAPE it would take eight hours. With the KIM-1 loader it would take 48 hours.

In writing a program like this HIGH SPEED I/O, with only two overlays, the assembler reads the source tape twice to edit and twice to compile. Also it spends the same time to save a corrected copy, being equivalent to six reads. If we make ten corrections during the debugging, we would spend one hour for I/O using HYPERTAPE.

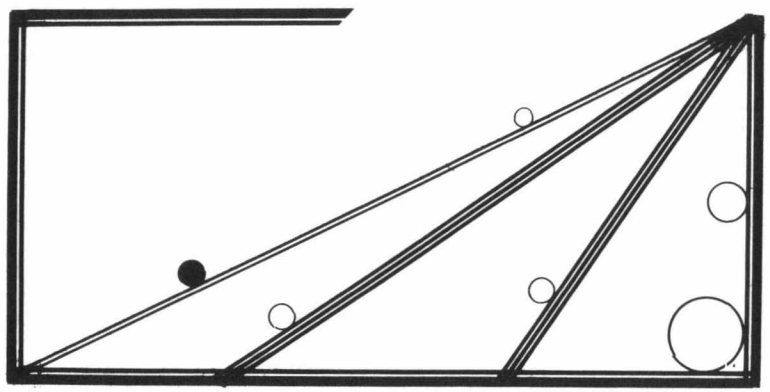
Furthermore, it is very convenient to use these routines Stand-Alone. It is nice to do a little housekeeping without needing to fire up the teletype and the Assembler-Editor.

Notes

1. MICRO-ADE™ for the 6502—Assembler, Disassembler, Editor, by Peter F. Jennings. Micro-Ware Ltd., 27 Firstbrooke Road, Toronto, Ontario, Canada, M4E 2L2.
2. *The First Book of KIM*, Hayden Book Co., Essex St., Rochelle Park, N.J. 07662. Cat. 5119-0. $9.00.
3. Realistic™ CTR-39, Model 14-819 by Radio Shack.
4. HELP RELAY PACKAGE, the Computerist, P.O. Box 3, S. Chelmsford, MA 01824.
5. Pyramid Data Systems, 6 Terrace Ave., New Egypt, N.J. 08533.

```
MCIO    MICIO-TAPE ASSEMBLER 6502-1.0 PAGE 01    2-8-78
These high speed I/O routines are patterned after the low speed
(hypertape) routines that they replace in the MICRO-ADE assembler,
editor by Peter Jennings.
0010:                   I/O ROUTINES EMPLOY TAPE CONTROL USING
0020:                   COMPUTELIST HELP DELAY PACKAGE
0030: 2EA9              NCIO    STC     $2EA9   NOW COMPATIBLE
0040: 2EA9              OUTCH   *       $1EAC   CASSETTE I/O
0050: 2EA9              RDTBY   *       $1E3E
0060: 2EA9              CRLF    *       $1E2F
0070: 2EA9              CHKL    *       $17E7
0080: 2EA9              CHKH    *       $17E8
0090: 2EA9              CHKT    *       $194C
0100: 2EA9              CANC    *       $00F0
0110: 2EA9              TIC     *       $00F1
0120: 2EA9              COUNT   *       $00F2
0130: 2EA9              ACC     *       $00F3
0140: 2EA9              CNT     *       $00F4
0150: 2EA9              TTID    *       $00FE
0160: 2EA9              POINTL  *       $00FA
0170: 2EA9              POINTH  *       $00FB
0180: 2EA9              PACKT   *       $1A00
0190: 2EA9              SED     *       $1742
0200: 2EA9              RDBIT   *       $1A41
0210: 2EA9              SAL     *       $17F5
0220: 2EA9              SAH     *       $17F6
0230: 2EA9              EAL     *       $17F7
0240: 2EA9              EAH     *       $17F8
0250: 2EA9              ID      *       $17F9
0260: 2EA9              INIT    *       $1E8C
0270: 2EA9              INTVEB  *       $1932
0280: 2EA9              INCVEB  *       $19EA
0290: 2EA9              VEB     *       $17EC
0300: 2EA9 A9 0C        CREAD   LDAIM   $0C     SET BIT 2 = 0, TO TURN ON
0310: 2EAB 8D 03 17             STA     $1703   CASSETTE
0320: 2EAE 29 FD                ANDIM   $FD
0330: 2EB0 8D 02 17             STA     $1702   PB2 APPLICATION PIN 11
0340: 2EB3 A9 F9                LDAIM   $F9
0350: 2EB5 85 FA                STA     POINTL
0360: 2EB7 A9 17                LDAIM   $17     IF ERROR, DISPLAY 17
0370: 2EB9 85 FB                STA     POINTH
0380: 2EBB A9 7F                LDAIM   $7F     TURN ON KIM DISPLAY
0390: 2EBD 8D 41 17             STA     $1741
0400: 2EC0 D8                   CLD
0410: 2EC1 A9 13                LDAIM   $13     TURN ON INPUT PORT
0420: 2EC3 8D 42 17             STA     SED
0430: 2EC6 20 41 1A     SYNC    JSR     RDBIT   READ A BIT AT A TIME
0440: 2EC9 46 F0                LSR     CANC    SHIFT INTO CANC
0450: 2ECB 05 F0                ORA     CANG
0460: 2ECD 85 F0                STA     CANC
0470: 2ECF 8D 40 17             STA     $1740   PUT ON LED
0480: 2ED2 C9 16        TST     CMPIM   $16     IS IT A SYNC ?
0490: 2ED4 D0 F0                BNE     SYNC    KEEP TRYING
0500: 2ED6 20 48 2F             JSR     RDCHT   IN SYNC, READ CHARAC
0510: 2ED9 8D 40 17             STA     $1740   DISPLAY ON LED
0520: 2EDC C9 2A                CMPIM   $2A     START OF DATA  *  ?
0530: 2EDE D0 F2                BNE     TST     KEEP TRYING
0540: 2EE0 20 48 2F             JSR     RDCHT   READ  ID
0550: 2EE3 8D F9 17             STA     ID      SAVE IT
0560: 2EE6 85 F9                STAZ    $F9
0570: 2EE8 A2 03                LDXIM   $03     GET 4 ADDRESSES
0580: 2EEA 20 48 2F     LOADAD  JSR     RDCHT
0590: 2EED 9D F5 17             STAAX   SAL
0600: 2EF0 CA                   DEX
0610: 2EF1 10 F7                BPL     LOADAD
0620: 2EF3 A9 8D                LDAIM   $8D     SET UP VEB
0630: 2EF5 8D EC 17             STA     VEB
0640: 2EF8 20 32 19             JSR     INTVEB
0650: 2EFB 20 48 2F     READIT  JSR     RDCHT
0660: 2EFE 20 4C 19             JSR     CHKT    ADD TO CHECKSUM
0670: 2F01 20 EC 17             JSR     VEB     STORE IT
0680: 2F04 20 38 2F             JSR     INCVE    INCREMENT ADDR.
0690: 2F07 90 F2                BCC     READIT  GET NEXT BYTE
0700: 2F09 20 48 2F     ENDRD   JSR     RDCHT   GET CHECKSUM
0710: 2F0C CD E7 17             CMP     CHKL    AND CHECK IT
0720: 2F0F D0 0B                BNE     AOK
0730: 2F11 20 48 2F             JSR     RDCHT
0740: 2F14 38                   SEC
0750: 2F15 ED E8 17             SBC     CHKH
0760: 2F18 D0 02                BNE     AOK
0770: 2F1A 85 FB                STA     POINTH
0780: 2F1C AD 02 17     AOK     LDA     $1702
0790: 2F1F 09 04                ORAIM   $04     SET BIT 2 =1, TO
0800: 2F21 8D 02 17             STA     $1702   STOP CASSETTE
0810: 2F24 20 19 1F     DISP    JSR     $1F19   SCAND
0820: 2F27 20 6A 1F             JSR     $1F6A   GETKEY
0830: 2F2A C9 0C                CMPIM   $0C
0840: 2F2C F0 07                BEQ     DWN     CONTINUE
0850: 2F2E C9 0B                CMPIM   $0B
0860: 2F30 D0 F2                BNE     DISP
0870: 2F32 4C A9 2E             JMP     CREAD   EAD TAPE
0880: 2F35 4C 8C 1E     DWN     JMP     INIT
0890: 2F38 20 EA 19     INCVE   JSR     INCVEB  POINT TO NEXT BYTE
0900: 2F3B AD ED 17             LDA     VEB     +01  CHECK FOR END
0910: 2F3E CD F7 17             CMP     EAL
0920: 2F41 AD EE 17             LDA     VEB     +02
0930: 2F44 ED F8 17             SBC     EAH
0940: 2F47 60                   RTS
0950: 2F48 20 24 1A     RDCHT   JSR     $1A24
0960: 2F4B AD EA 17             LDA     $17EA
0970: 2F4E 60                   RTS
0980: 2F4F A5 62        CRITE   LDAZ    $62
0990: 2F51 8D F9 17             STA     ID
ID=31

0010:                    THIS OUTPUT ROUTINE WAS ADAPTED FROM HYPERTAPE
0020:                    BY JIM BUTERFIELD.  SEE FIRST BOOK OF  KIM
0060: 2F54 A9 0C        CWRITE LDAIM $0C        SET BIT 3 = 0, TO
0070: 2F56 8D 03 17             STA   $1703     TURN ON CASSETTE
0080: 2F59 29 F7                ANDIM $F7
0090: 2F5B 8D 02 17             STA   $1702     PB3 APPLICATION PIN 12
0100: 2F5E A9 AD                LDAIM $AD       SET UP VEB
0110: 2F60 8D EC 17             STA   VEB
0120: 2F63 20 32 19             JSR   INTVEB
0130: 2F66 A9 27                LDAIM $27       SET FLAG FOR
0140: 2F68 85 F0                STA   GANG      SBD TOGGLE
0150: 2F6A A9 BF                LDAIM $BF       TURN ON
0160: 2F6C 8D 43 17             STA   $1743     OUTPUT
```

```
0170: 2F6F A2 F0            LDXIM $F0      SEND 240 SYNC CHARS
0180: 2F71 A9 16            LDAIM $16
0190: 2F73 20 AF 2F         JSR   HIC
0200: 2F76 A9 2A            LDAIM $2A      SEND START OF DATA SIGNAL
0210: 2F78 20 BE 2F         JSR   OUTCHT
0220: 2F7B A2 04            LDXIM $04
0230: 2F7D BD F5 17  PUTAD  LDAAX SAL
0240: 2F80 20 BE 2F         JSR   OUTCHT
0250: 2F83 CA               DEX             PUT ADDR. & ID ON TAPE
0260: 2F84 10 F7            BPL   PUTAD
0270: 2F86 20 EC 17  DUMPTA JSR   VEB      GET A BYTE
0280: 2F89 20 BE 2F         JSR   OUTBTC   AND SEND
0290: 2F8C 20 38 2F         JSR   INCVE     POINT TO NEXT BYTE
0300: 2F8F 90 F5            BCC   DUMPTA
0310: 2F91 AD E7 17         LDA   CHKL     SEND CHECK SUM
0320: 2F94 20 BE 2F         JSR   OUTCHT
0330: 2F97 AD E8 17         LDA   CHKH
0340: 2F9A 20 BE 2F         JSR   OUTCHT
0350: 2F9D A2 02            LDXIM $02      SEND 2
0360: 2F9F A9 04            LDAIM $04      EOT CHARS
0370: 2FA1 20 AF 2F         JSR   HIC
0380: 2FA4 AD 02 17         LDA   $1702    TURN OFF CASSETTE
0390: 2FA7 09 08            ORAIM $08      BY SETTING BIT 3 = 1
0400: 2FA9 8D 02 17         STA   $1702    PB3 APPLICATION PIN 12
0410: 2FAC 4C 24 2F         JMP   DISP      RETURN TO MONITOR
0420:                    SEND X CHARACTERS TO TAPE
0430: 2FAF 86 F1     HIC    STX   TIC
0440: 2FB1 48        HICA   PHA
0450: 2FB2 20 BE 2F         JSR   OUTCHT
0460: 2FB5 68               PLA
0470: 2FB6 C6 F1            DEC   TIC
0480: 2FB8 D0 F7            BNE   HICA
0490: 2FBA 60               RTS
0500:                    SEND WITH CHECKSUM
0510: 2FBB 20 4C 19  OUTBTC JSR   CHKT     CALC. CHECKSUM
0520:                    ROUTINE TO SEND 8 BIT BYTE
0530: 2FBE A0 08     OUTCHT LDYIM $08      8 BIT COUNT
0540: 2FC0 84 F2            STYZ  COUNT
0550: 2FC2 A0 02     TRY    LDYIM $02      START AT 3600 HERTZ
0560: 2FC4 84 FE            STY   TRIE
0570: 2FC6 85 F3     ZON    STA   ACC      SAVE ACCUMULATOR
0580: 2FC8 B9 FE 2F         LDAAY NPUL     PUT NO. OF HALF
0590: 2FCB 85 F4            STA   CNT      CYCLES IN COUNTER
0600: 2FCD A5 F3            LDA   ACC      RESTORE ACC.
0610: 2FCF 48               PHA             SAVE CHARACTER
0620: 2FD0 2C 47 17  ZONA   BIT   $1747    WAIT FOR END OF CYCLE
0630: 2FD3 10 FB            BPL   ZONA
0640: 2FD5 B9 FC 2F         LDAAY TIMG     SET UP TIMER
0650: 2FD8 8D 44 17         STA   $1744    FOR THIS PULSE
0660: 2FDB A5 F8            LDA   CANC     CHANGE STATE
0670: 2FDD 49 80            EORIM $80      OF OUTPUT
0680: 2FDF 8D 42 17         STA   $1742    PORT
0690: 2FE2 85 F8            STA   CANC
0700: 2FE4 C6 F4            DEC   CNT      IF NOT DONE
0710: 2FE6 D0 E8            BNE   ZONA     CONTINUE
0720: 2FE8 68               PLA           RETRIEVE BYTE
0730: 2FE9 C6 FE            DEC   TRIE
0740: 2FEB F0 05            BEQ   SETZ
0750: 2FED 30 07            BMI   ROUT
0760: 2FEF 4A               LSRA            PUT BIT IN CARRY
0770: 2FF0 90 D4            BCC   ZON
0780: 2FF2 A0 00     SETZ   LDYIM $00      SWITCH TO 2400 HZ
0790: 2FF4 F0 D0            BEQ   ZON
0800: 2FF6 C6 F2     ROUT   DEC   COUNT
0810: 2FF8 D0 C8            BNE   TRY      CONTINUE
0820: 2FFA 60               RTS           TILL DONE
0830:                    TIMING TABLE                KIM
0840: 2FFB 02            NPUL   =     $02      03  06  0C
0850: 2FFC C3            TIMG   =     $C3
0860: 2FFD 03                   =     $03      04  09  12
0870: 2FFE 7E                   =     $7E
ID=  SPEED RATIO (SS/KIM)       =     12      8.5  4   2
     If using a cheaper cassette, you may need to reduce speed by
-T   loading a larger number at addresses  2FFB  and  2FFD as in-
     dicated in above line.
    SYMBOL TABLE 3000 313E
    ACC    00F3   AOK    2F1C   CHKH   17E8   CHKL   17E7
    CHKT   194C   CNT    00F4   COUNT  00F2   CREAD  2EA9
    CRITE  2F4F   CRLF   1E2F   CWRITE 2F54   DISP   2F24
    DUMPTA 2F86   DWN    2F35   EAH    17F8   EAL    17F7
    ENDBD  2F09   GANC   00F8   HICA   2FD1   HIC    2FAF
    ID     17F9   INCVE  2F38   INCVEB 19EA   INIT   1E8C
    INTVEB 1932   LOADAD 2EEA   NCIO   2EA9   NPUL   2FFB
    OUTBTC 2FBB   OUTCH  1EA0   OUTCHT 2FBE   PACKT  1A00
    POINTH 00FB   POINTL 00FA   PRTBY  1E3B   PUTAD  2F7D
    RDBIT  1A41   RDCHT  2F48   READIT 2EFB   ROUT   2FF6
    SAH    17F6   SAL    17F5   SLD    1742   SETZ   2FF2
    SYNC   2EC6   TIC    00F1   TIMG   2FFC   TRIE   00FE
    TRY    2FC2   TST    2EB2   VEB    17EC   ZONA   2FD0
    ZON    2FC6
```



THE ZILOG Z-8

BY U. R. SOURCE

Zilog's New Z-8 is the first of a new generation of single-chip microcomputers. It is also capable of being used as an 8-bit microprocessor which can address up to 128K bytes of external memory. The architecture of the chip is such that under software control, it can be given a variety of different I/O and memory configurations, the most important of which are: a) an I/O intensive single-chip microcomputer and b) a memory intensive microprocessor with up to 64K bytes of data and 64K bytes of program memory. Such flexibility is made possible by merging the multiplexed data/address bus (and the associated control signals) with the I/O-oriented ports. Thus external memory can be addressed while as many I/O lines are needed can be maintained.

I/O

The Z-8 has a total of 32 pins dedicated to I/O. They are arranged as four 8-line ports and are capable of being configured for input, output or bidirectional operation; They all have pull-ups and pull-downs compatible with TTL loads. In addition, all ports are programmable to provide status signals, timing and parallel or serial I/O features with or without handshake.

Port 1 can be used as a byte I/O port or bidirectionally in conjunction with Port 0 for accessing external memory. In the latter application, the lower 8 bits of the address are multiplexed with data. If more than 256 external memory locations must be accessed, the upper 4 or 8 bits of the address are provided by Port 0.

Port 0 is nibble programmable as either address or I/O. If the address requirements are 12 bits or less, the upper 4 bits of Port 0 can be independently configured as I/O while using the lower 4 bits in conjunction with Port 1 for addressing.

Port 2 has 8 buffered lines which can be programmed in bit mode as inputs or outputs. The output buffers can be programmed to inhibit the active pull-ups and thus provide open-drain outputs.

Port 3 is a control port and an I/O port. Its 8 lines are fixed directionally as four in and four out. In addition, two of its lines can be programmed as a full-duplex serial asynchronous receiver/transmitter. Port 3 provides handshake control lines RDY and STB for the other 3 ports when required. Other control functions of Port 3 include timer in and out, status out and interrupt request.

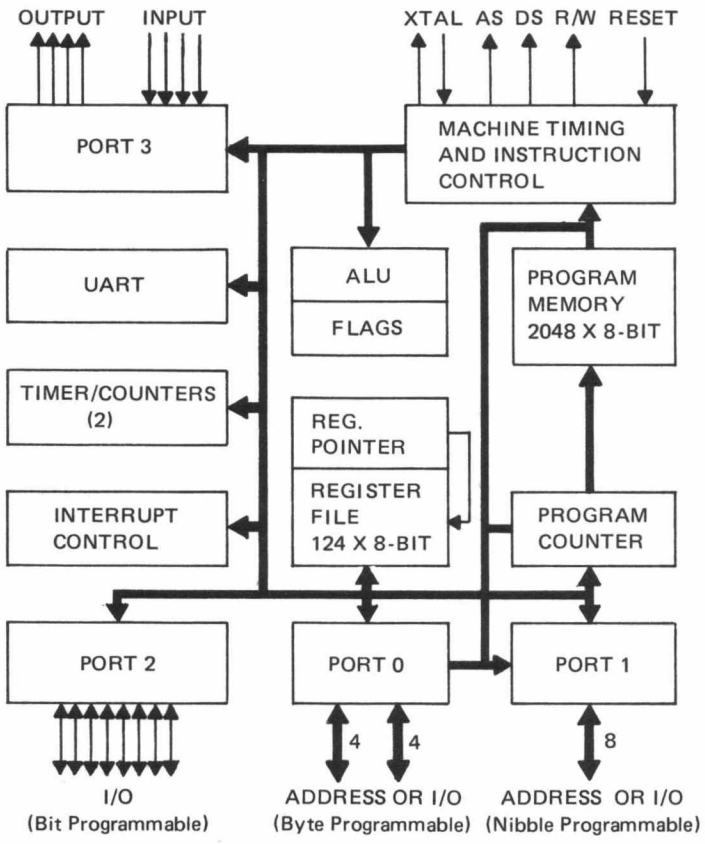
When the two lines of Port 3 are programmed as the full-duplex asynchronous receiver/transmitter, a bit rate of up to 62.6K bits per second can be generated with the crystal frequency at the maximum of 8MHz (4MHZ true internal operation). This configuration also provides for programmable odd parity.

Interrupts

The Z-8 has six maskable and prioritized interrupts as well as reset, which acts as a vectored nonmaskable interrupt of highest priority. The six sources of interrupt are the four input lines of Port 3 and the two timers, T0 and T1. There is an Interrupt Request Register for storing the requests. The Interrupt Mask Register can store bits to disable the six interrupts individually or use a seventh bit for a global disable of all interrupts except reset.

In handling prioritized interrupts, the priority encoder, controlled by the Interrupt Priority Register, outputs an address which points to a vector in program memory. When the interrupt is serviced, the Z-8 globally disables all other interrupts, pushes flags and program counter onto the stack and branches to the location specified by the vector. Disabled interrupts can be re-enabled either globally or individually and the use of separate Interrupt Request and Mask registers makes it possible to use interrupt nesting and interrupt polling together.

Z8 Block Diagram



The Z-8 is equipped with two internal 8-bit counters with individual 6-bit prescalers. The counter/timer circuits can operate independently of the processor instruction sequence and thus unburden the program from such functions as event counting and timekeeping. They have two user-selectable clock sources: the microprocessor clock (maximum internal operation of 4MHz) divided by four, or the external timer input line at Port 3 can function as a external clock source (max 1MHz), a trigger input (which can also be retriggered), or as a gate input for the internal clock.

The prescalers can be programmed to divide their input frequency by any number from 1 to 64. The counters count down from any number from 1 to 256 which is stored in a holding register. At the end of the count, an interrupt request is generated. The two software-controlled operating modes for the counters are: a) stop at zero, and b) automatic reload and continue count. The counter/prescalers can be started, then stopped anywhere in the count; they can then be restarted anywhere in the count or restarted at the initial value. One of the Port 3 lines serves as a timer output for Timers 0 and 1 or for the internal clock. It toggles each time the end of a count is reached.

CPU

The Z-8 can access an internal Program Memory, external Program Memory, external Data Memory and an internal Register File. These different memories are implemented as separate address spaces and other address such as I/O and control registers are mapped into the Register File.

The first 2K bytes of Program Memory are on-chip mask-programmable ROM. The first 14 bytes of this on-chip memory contain the address for vectoring the reset and six interrupts to the proper locations in Program Memory. At addresses above 2048, the CPU automatically addresses external Program Memory.

In addition to the 64K of Program Memory, the Z-8 can directly address up to 64K of external Data Memory using Port 1 and if necessary, all or part of Port 0 as described above in the secion on I/O.

The internal Register File consists of 4 I/O port registers, 124 general purpose registers and 16 status and control registers for setting the various configurations of the Z-8. This latter group includes the input and output registers for the full-duplex asynchronous receiver/transmitter. The Register File is organized into 9 working register groups of 16 registers each. The current group of working registers is selected by a register pointer. The working registers within the current group require only a 4-bit address field.

The fact that the I/O and control registers are mapped into the Register File allows any instruction to process I/O or control information and thus eliminates the need for specialized instructions. Each of the 124 general purpose registers can serve as an accumulator, a pointer or an index register, and registers and register pairs may be directly or indirectly addressed.

The lowest group of working registers includes the four I/O ports. For I/O intensive operations and interrupt handling, the register pointer may be set to this group and fast instructions used to address the four I/O registers while using the 12 remaining registers for temporary storage and calculation. In like manner, all the registers for timers, port operation, serial interface and interrupt control are included in another working register group so that when the register pointer is set to it those sections of program dealing with system configuration can be substantially reduced in size.

In addition to these, the remaining seven groups may be indicated by the register pointer to perform calculations and other program operations. The Z-8 can still access them directly at any time regardless of the value of the register pointer.

Instruction Set

The Z-8 provides six main addressing modes: Register (R), Indirect Register (IR), Immediate (IM), Direct Address (DA), Relative Address (RA) and Indexed (X). There are in addition other addressing modes implied by certain instructions (Autoincrement, Register Pair and Indirect Register Pair). Basically, all instructions can use the Register, Indirect Register and Immediate addressing modes. The Load instruction may also use Indexed Addressing and Program Memory space instructions use 16 and 8-bit Relative Address modes. These register addressing modes can imply any register in the Register File of ony one of the 16 working registers. The Z-8 instructions permit the addressing of registers in the Register File or working registers or a combination of the two when two addressing modes are used.

The fact that the I/O and control registers are mapped into the Register File means that any register instruction can execute I/O or control operations. Using the working register concept, a programmer can execute an input or output operation between any I/O port and any Register File location in only two bytes (6 cycles).

The six levels of vectored interrupts—maskable and prioritized under software control—plus the capability of mixing interrupts and polling give the Z-8 a very flexible interrupt structure. To support this interrupt structure, the Z-8 is provided with a flexible stack scheme and fast context switching capability. Context switching is the saving and restoring of such pertinent information as flags and working registers when an interrupt occurs.

The Z-8 can use an internal or external stack limited only by the available memory and all under software control. One of the registers or register pairs in the Register File contains an 8-bit or 16-bit stack pointer. The stack is utilized by Call instructions to the 16-bit return value of the program counter. When interrrupts occur, the 16-bit program counter and an 8-bit flag register are automatically saved on the stack. Push and Pop instructions can also save any register from the Register File on the stack. Any bit within any byte in the Register File can be set, reset and tested by the Z-8 including any I/O port and control registers. Thus, any byte can be used as a flag byte to be set, reset and tested by a single instruction.

In the Direct Addressing mode, the 16-bit addresses for jumps and calls are contained in 3-byte instructions. Indirect jumps and calls are 2-byte instructions which point to the register pair containing the actual 16-bit destination address. Two instructions which operate on 16-bit values stored in register pairs are Increment Word and Decrement Word. They manipulate 16-bit addresses for Program Memory and external Data Memory and can also be applied to 16-bit data words.

High-speed block transfers between external memory and the Register File in the Z-8 occur with the Load Constant Autoincrement and Load External Autoincrement operations.

These instructions automatically increment the source and destination address much the same way as the DMA operation. Other forms of address manipulation are possible for block transfers such as using Load Constant and Load External instructions in conjunction with the increment/decrement byte and word instructions. The following is a summary of the Z-8 instruction set:

- *Load and Exchange.* Clear, Exchange, Load, Load Constant, Load External, Pop, Push.
- *Arithmetic.* Add, Add With Carry, Decimal Adjust, Decrement Byte And Word, Increment Byte And Word, Subtract, Subtract With Carry.
- *Logical.* AND, Compare, Complement, OR, Exclusive OR.
- *Program Control.* Call; Decrement And Jump On Non-Zero; Interrupt Return; Jump Conditional; Jump Relative Conditional; Return; Set, Reset and Complement of Carry Flag.
- *Bit Manipulation.* Test Under Mask, Test Complement Under Mask.
- *Rotate and Shift.* Rotate Left, Rotate Left Circular, Rotate Right, Rotate Right Circular, Shift Right Arithmetic, Swap Nibbles.
- *Block Transfer.* Load Constant Autoincrement, Load External Autoincrement.
- *CPU Control.* Disable Interrupts, Enable Interrupts, No Op, Set Register Pointer.

When operating on working register data, most one- and two-operand instructions in the Z-8 require 6 T-cycles. This is the minimum instruction execution time. With a crystal frequency of 8MHz (giving a true internal operation frequency of 4MHz), one T-cycle is 250ns. Two-operand instructions manipulating Register File data generally execute in 9 T-cycles. The longest instruction, CALL, takes 19 T-cycles.

TEST FOR SWTPC GRAPHICS DISPLAY

Dear Dr. Dobbs, *Received: 78 May 11*

The following is a program which I have found quite useful in hardware debugging my SWTP GT-6144. I hope some other readers will benefit by it.

The program should be self explanatory. It is relocatable to other memory locations as the following MIKBUG dumps show. The first one is from the enclosed program:

```
P
S113A014FEA0028600A70186FFA7008634A701BD1F
S10CA024E055FEA002A70020F69D
```

The next one is from the program which was reassembled to start at $0100:

```
P
S1130100FEA0028600A70186FFA7008634A701BDD2
S10C0110E055FEA002A70020F650
```

Those who are not familiar with MIKBUG may notice that the last byte on each line is different. This is the checksum generated by MIKBUG and reflects the different address (2nd and 3rd bytes) in each line.

Sincerely,
Jim Lanford

P.O. Box 60725
Sunnyvale, CA 94088

```
                        NAM     GTTEST

                    * a TEST program for the SWTP GT-6144
                    * graphics display unit.

                    *       written by Jim Lanford
                    * for the SWTP 6800 using MIKBUG

                    * this is a quick and dirty data inputer
                    *       -it inputs hex from the terminal
                    *       -thru the MP-La (port address at A002)
                    *       -to the GT-6144  (usually hooked to the "A"
                    *        side of the MP-La)
                    *       -this test program will also work on the "B"
                    *        side of the MP-La if it is jumpered for output
                    *        and the port address + 2 is in location A002

                    * This program is very useful for hardware
                    * types who want to debug the GT-6144 under a
                    * controlled, one-byte-at-a-time environment.
                    * It is also useful for some specialized
                    * software display bugs.

                    * If you get lost or type a non-HEX digit, the
                    * program returns to MIKBUG, and is warm started
                    * by typing "G".

                    * The HEX address of the MP-La parallel
                    * interface should be set in location A002/3.
                    * Port #3 (800C) is usually used.

                    * GT-6144 Data reminders
                    *   Horz bit select & write  $40 -> $7F
                    *   Horz bit select & erase  $00 -> $3F
                    *   Vertical bit select....  $80 -> $DF
                    *       $F0 = invert screen
                    *       $F1 = normal screen
                    *       $F2 = enable  CT-1024
                    *       $F3 = disable CT-1024
                    *       $F4 = enable  graphics
                    *       $F5 = disable graphics


                    *  MIKBUG subroutine
E055                BYTE    EQU     $E055       inputs two HEX char.

                    * register addresses and select definitions

A002                PIALOC  EQU     $A002       PIA port address from A002
0001                CR      EQU     1           control register
0000                DDR     EQU     0           data direction register
0000                OR      EQU     0           output register
0000                SELDDR  EQU     $00         select DDR
00FF                DEFOUT  EQU     $FF         define all bits output
0034                SELOR   EQU     $34         select OR

                    * initialize 6820

A014                        ORG     $A014
A014 FE A0 02               LDX     PIALOC      get port address from memory
A017 86 00                  LDA A   #SELDDR
A019 A7 01                  STA A   CR,X        select DDR via CR
A01B 86 FF                  LDA A   #DEFOUT
A01D A7 00                  STA A   DDR,X       all bits output in DDR
A01F 86 34                  LDA A   #SELOR
A021 A7 01                  STA A   CR,X        select OR via CR

                    * input/output loop

A023 BD E0 55       LOOP    JSR     BYTE        input a byte
A026 FE A0 02               LDX     PIALOC      reload index to allow warm start
A029 A7 00                  STA A   OR,X        output to GT-6144
A02B 20 F6                  BRA     LOOP        do it again
                            END
      NO ERROR(S) DETECTED


   SYMBOL TABLE:
BYTE    E055        CR      0001        DDR     0000        DEFOUT  00FF
LOOP    A023        OR      0000        PIALOC  A002        SELDDR  0000
SELOR   0034
```

MAILING LIST SHOP-AROUND

Dated: 78 April 13

Here at PCC we are collecting sources of mailing list programs, especially those with enough coding space for use by small magazines. This is not only for our own continued well being, it seems that there is a tremendous variety of people and businesses which use mailing lists—from salesmen to public interest lobbyists. We'd like to invite our readership to send us any sources, or suggestions in this project. If our research uncovers anything of interest, we'll try to let you know.

William Bruneau,
Circulation Manager

LAZARUS

A Program to Resurrect BASIC Programs on the Apple-II Computer

BY ANDY HERTZFELD
2220 Dwight Way, #210
Berkeley CA 94704
(415) 548-3072

Many unfortunate souls have experienced the frustration which occurs when you realize that you have just accidentally erased the program you were working on. Kicking the computer may provide some emotional satisfaction but it won't bring your program back. This article describes Lazarus, a short machine language program which can resurrect inadvertently erased BASIC programs on the Apple-II computer.

When an Apple-II integer BASIC program is erased, it is usually not destroyed. It is probably still sitting around somewhere in memory; however, the BASIC interpreter no longer knows where to find it so it is effectively lost. Lazarus scans through memory trying to find the largest valid BASIC program that it can. If it finds one it resets the interpreter's pointers to point to the found program, thereby resurrecting it. If no valid program can be found an error message is printed. In either case control is returned to the BASIC interpreter.

Of course, things are not quite this simple. If there are many program fragments floating around the workspace, Lazarus might not resurrect the one you are interested in. If you had performed some deletions of low-numbered statements prior to the erasure, Lazarus may restore the deleted lines along with the rest of the program. If your program was located in an unusual part of memory, Lazarus may not know where to look for it. Finally it is possible but unlikely that Lazarus could introduce spurious statement at the beginning and end of the resuscitated program. In practice these problems are rare; Lazarus should be successful most of the time.

The Details

To understand how Lazarus works, you must first know something about the internal representation of an Apple-II integer BASIC program. We will examine this briefly here; for a more detailed exposition see Steve Wozniak's article in *Dr. Dobb's* #23 (vol. 3, issue 3) or the Apple reference material when they finally get around to putting it out (the current reference manual isn't very helpful).

The BASIC "workspace" is the area of memory used to store the currently active BASIC program and its associated variables. On the Apple-II it is defined by two zero-page pointers: LOMEM, which points to the lowest addresses byte in the workspace and HIMEM, which points one past the highest. Program statements are placed into memory starting at HIMEM and are pushed down as each successive statement is entered. Another pointer called PP (for program pointer) is used to point to the first, lowest numbered statement. PP decreases as the program grows; a program is erased by setting PP to HIMEM. Thus all Lazarus has to do is scan memory from LOMEM to HIMEM and isolate the beginning and end of the longest BASIC program residing there. To accomplish the resurrection, it simply stores the address of the start of the program into PP and its ending address (plus 1) into HIMEM.

As each line of BASIC program is entered, it is translated into a compact internal form consisting of four parts. First there is a length byte indicating the total length of the statement. It is followed by a 2-byte line number, low order byte first. Next comes a string of code syllables or tokens which make up the body of the statement. Finally there is a special end-of-statement token ($01) at the end of every statement.

Knowing this information, it is not too hard to find a BASIC program segment; simply scan through memory searching for an end-of-statement token. When one is found we can check if it ends a stanement by examining length bytes. The fact that the statement's line numbers must be in ascending order is used to perform an additional validity check. The program segment ends as soon as this structure is violated.

There may be many valid fragments floating around the workspace (like pieces of the Startrek program you played an hour ago); Lazarus will resurrect the largest one since that is usually the one we are interested in. Since that is not always the one we want, another entry point is provided that causes the last segment (which tends to be the most recent program worked on) to be chosen.

Lazarus was implemented using SWEET 16, a very useful little 16-bit virtual machine that is interpreted by the Apple-II firmware. SWEET 16 saves a lot of code at the expense of a significantly longer execution time. Since Lazarus does a lot of 16-bit operations, Sweet 16 was able to spare me a large amount of tedious coding. (See Woznick's article in the Nov 77 *Byte* for a detailed description of SWEET 16).

Using Lazarus

The code for Lazarus is relocatable since all of its branches are relative and the only local storage it uses is in page zero. $300 seems to be a convenient place to load it. You can load it for the first time by going into the monitor mode and typing:

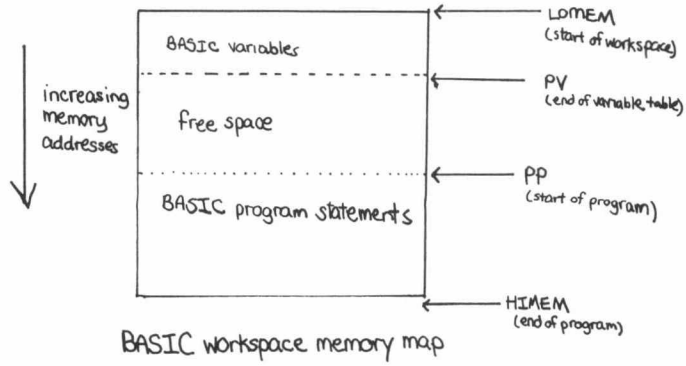
```
300: A9 FF D0 02 A9 00 85 F0 85 F1 20 89 F6 B0 3A E0
310: 35 13 04 00 16 4A 00 17 4C 00 66 36 32 67 37 22
320: D7 03 59 42 D5 07 F8 22 F0 38 39 E8 48 D3 02 18
330: A9 34 44 D5 07 12 27 D4 05 0E 24 F0 39 68 31 E4
340: 64 D1 05 04 29 38 01 E3 22 F0 34 D9 06 D1 F4 F4
350: F4 11 05 00 84 D1 06 0D E1 10 FF 00 D1 02 C0 24
360: D6 05 BC 01 EF 29 32 E2 B4 31 18 F0 00 68 06 04
370: 21 DA 02 AB 21 3A 24 3B 29 3C 01 A3 2A 06 11 11
380: CA 00 2B 71 11 4C 00 2C E0 71 00 20 3A FF 4C 03
390: E0 00 20 2D FF 4C 03 E0
```

Once it has been manually loaded, it can be saved on tape by typing "300.3A0W" and then subsequently loaded back in by typing "300.3A0R".

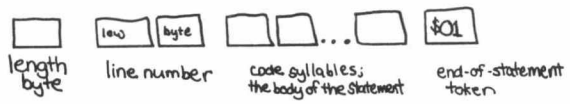
Lazarus is very easy to use. After cursing yourself out a few times for accidentally erasing your program, return to monitor mode and load Lazarus. Then return to BASIC and type "CALL 768" or "CALL 772". The former will cause the longest program segment in the workspace to be restored; the latter will cause the last to be selected. When it returns, do a "LIST" to see what it found. If nothing could be dredged up, an error message is printed. If both entry points recover the wrong program, you can adjust LOMEM and HIMEM accordingly and try again. If all else fails, you can still always kick the computer.

A useful extension to Lazarus would be to recover the variable table along with the program, which could be very important in some cases. I would like to thank Bruce Tognazzini for providing the initial idea for this program.

A complete source listing follows. . .



BASIC workspace memory map



BASIC statement representation

Lazarus Source Listing

```
                        Sweet 16 Register Assignments

                        R1:  auxiliary
                        R2:  main scan pointer
                        R3:  contains $4 for comparisons
                        R4:  auxiliary; backwards scan pointer
                        R5:  contains $1 for comparisons
                        R6:  LOMEM
                        R7:  HIMEM
                        R8:  used in forward scan
                        R9:  used in forward scan
                        R10: maximum size found so far
                        R11: PP of max segment
                        R12: HIMEM of max segment

                *       Entry point $300 sets the flag so the largest program is found
                *       Entry point $304 sets the flag so the last program is found

0300: A9 FF             LDA  #$FF       set flag to all ones
0302: D0 02             BNE  STORE      always taken
0304: A9 00             LDA  #$00       set flag to all zeros
0306: 85 F0     STORE   STA  $F0
0308: 85 F1             STA  $F1        set the switch at $F0

                *       Initialize registers, etc.

030A: 20 89 F6          JSR  SWEET16    enter SWEET 16 mode
030D: B0                SUB  R0         zero the accumulator
030E: 3A                ST   R10        zero max size
030F: E0                INR  R0
0310: 35                ST   R5         set R5 to 1
0311: 13 04 00          SET  R3,4       set R3 to 4
0314: 16 4A 00          SET  R6,LOMEM   get address of LOMEM
0317: 17 4C 00          SET  R7,HIMEM   HIMEM, too
031A: 66                LDD  @R6
031B: 36                ST   R6         set R6 to LOMEM
031C: 32                ST   R2         initialize scan ptr there, too
031D: 67                LDD  @R7
031E: 37                ST   R7         set R7 to HIMEM

                *       The main loop starts here.  We scan for an end-of-statement token  ($01)

031F: 22        MAINLOOP LD  R2         test to see if we're past
0320: D7                CMP  R7         HIMEM; if we are, we are
0321: 03 59             BC   ALLDONE    finally done
0323: 42                LD   @R2        pick up the next byte
0324: D5                CMP  R5         is it the EOS token?
0325: 07 F8             BNZ  MAINLOOP   if its not, continue the scan
0327: 22                LD   R2         initialize R8 and R9
0328: F0                DCR  R0         to point to the
0329: 38                ST   R8         end-of-statement token
032A: 39                ST   R9

                *       At this point a candidate for a statement has been found.  The
                *       following loop performs validity checks on successive statements
                *       until an invalid one is found or HIMEM is reached.
                *       R9 points to the last byte of the most recent valid statement.

032B: E8        CHKLOOP  INR R8         bump R8 to the new length byte
032C: 48                 LD  @R8        pick up the length byte
032D: D3                 CMP R3         make sure its greater than 4
032E: 02 18              BNC INVALID    if its not, its invalid
0330: A9                 ADD R9
0331: 34                 ST  R4         R4 now holds the end address
0332: 44                 LD  @R4        pick up the last byte of statement
0333: D5                 CMP R5         it better be an EOS token
0334: 07 12              BNZ INVALID    if its not, its invalid
```

```
*   At this point, R8 points one past the beginning of a valid statement
*   and R4 points to the start of the next, possibly valid one.  We
*   perform line no. validity checks and test against HIMEM
336: 27                     LD   R7       pick up HIMEM and see if
337: D4                     CMP  R4       we've passed it
338: 05 0E                  BM   INVALID  if we have, its invalid
33A: 24                     LD   R4
33B: F0                     DCR  R0       its valid, so mark it
33C: 39                     ST   R9       in R9

33D: 68                     LDD  @R8      pick up the line number of the
33E: 31                     ST   R1       old line and save in R1
33F: E4                     INR  R4
340: 64                     LDD  @R4      pick up the newer line number
341: D1                     CMP  R1       and compare it with the older one
342: 05 04                  BM   INVALID  if new is less than old, its no good

344: 29                     LD   R9       the statements OK so update R8
345: 38                     ST   R8       with R9
346: 01 E3                  BR   CHKLOOP  go examine the next statement

*   At this point we have reached the end of a valid program segment;
*   R9 points to the last byte of the last valid statement
*   R2 points one past the last byte of the first valid statement
*   If R2 = R9 + 1, we didn't find anything

348: 22          INVALID    LD   R2       point R4 to the last byte
349: F0                     DCR  R0       of the last valid statement
34A: 34                     ST   R4
34B: D9                     CMP  R9       if R9=R4, then nothing was found
34C: 06 D1                  BZ   MAINLOOP   so forget it and go continue scan
34E: F4                     DCR  R4
34F: F4                     DCR  R4       subtract 3 from R4 to prepare
350: F4                     DCR  R4       for the backwards scan
351: 11 05 00               SET  R1,5     R1 will hold the tentative length

*   In the following loop we scan back searching for the beginning of the
*   segments initial statement

354: 84         SCANLOOP    POP  @R4      pick up the next byte
355: D1                     CMP  R1       compare with the length count
356: 06 0D                  BZ   DONE     if they're equal, we're done
358: E1                     INR  R1       increment R1 for the next iteration
359: 10 FF 00               SET  R0,$FF   test if we've gone 256 bytes
35C: D1                     CMP  R1       if we have, its too long
35D: 02 C0                  BNC  MAINLOOP so give up the segment
35F: 24                     LD   R4       make sure LOMEM isnt passed
360: D6                     CMP  R6
361: 05 BC                  BM   MAINLOOP if it has, give up
363: 01 EF                  BR   SCANLOOP go check the next byte

*   At this point we have found a valid program segment.  R4 points to the
*   first byte of the segment; R9 points to the last byte

365: 29         DONE        LD   R9       make R2 point past the recently
366: 32                     ST   R2       found segment to continue the
367: E2                     INR  R2       scan in the main loop
368: B4                     SUB  R4       compute the lenght of the segment
369: 31                     ST   R1       and save it in R1
36A: 18 F0 00               SET  R8,$F0   get the address of the flag
36D: 68                     LDD  @R8      load the flag
36E: 06 04                  BZ   UPDATE   if flag=0, always do it
370: 21                     LD   R1       restore the size
371: DA                     CMP  R10      and compare with the max so far
372: 02 AB                  BNC  MAINLOOP if its smaller, we don't want it

374: 21         UPDATE      LD   R1       restore the size
375: 3A                     ST   R10      and update the max size with it
376: 24                     LD   R4
377: 3B                     ST   R11      update the max PP
378: 29                     LD   R9
379: 3C                     ST   R12      update the max HIMEM
37A: 01 A3                  BR   MAINLOOP go find the next segment

*   We reach this point when the main loop has run its course (i.e., the
*   main scan pointer has reached HIMEM).  Set PP and HIMEM appropriately
*   and then return to the BASIC interpreter.  Print "ERR" if nothing found.

37C: 2A         ALLDONE     LD   R10      get the size of what we found
37D: 06 11                  BZ   ERROR    if its zero, signal an error
37F: 11 CA 00               SET  R1,PP    get the address of PP
382: 2B                     LD   R11
383: 71                     STD  @R11     set PP to start of program
384: 11 4C 00               SET  R1,HIMEM get the address of HIMEM
387: 2C                     LD   R12
388: E0                     INR  R0       increment it by 1
389: 71                     STD  @R1      set HIMEM to point one past program
38A: 00                     RTN           return to 6502 mode
38B: 20 3A FF               JSR  BELL     ring the bell to wake up user
38E: 4C 03 E0               JMP  BASIC+3  jump back to the interpreter
391: 00         ERROR       RTN           return to the 6502 mode
392: 20 2D FF               JSR  ERR      print the error message (+ring bell)
395: 4C 08 ED               JMP  BASIC+3  return to the BASIC interpreter
THE END
```

NEEDED: SYSTEM IDEAS FOR THE BLIND

Received: 78 April 10

Dear Jim,

A printer in a nearby town lost his sight a year ago. I have been asked to take a preliminary look at hardware and software that might help him maintain a useful role in his business. He is undergoing the usual training. His partner values his judgement and knowledge but he is restricted in access for the data on which to act.

My initial reaction is that a small data-base system with audio response would be a good place to start. An audio response type-writer keyboard should be feasible for input although audio input is the ideal. I have looked over your articles in *Dr. Dobb's* and am hoping you might be able to supply an update. I know there are groups working in this area but lack references.

Fortunately, some financial resources are available. If a suitable proposal can be assembled, I feel there is a good chance for Department of Rehabilitation assistance. This is, of course, speculative until we can get an initial reaction.

I have written Lloyd Rice from Computalker and will send out inquiries as I run across references. Any information you can pass along will be appreciated.

Sincerely,

James E. Tarvid
Systems Consultants, Inc.

Dellwood, WI 53927
(608) 339-7259

Any readers who may be able to help out are urged to make contact.

—TRW

A TINY BASIC EXTENSIONS PACKAGE

BY LEOR ZOLMAN
362 Memorial Dr.
Cambridge, MA 02139

After getting Li Chen Wang's Tiny Basic up and running about six months ago, from the version published in *Dr. Dobb's Journal*, I knew I had finally found what I had been searching for ever since first getting my IMSAI up: a well structured and documented Tiny Basic interpreter which could easily be modified to let me do my own thing. Thanks to Dr. Wang's consideration for the needs of his Basic's users, there was ample space available in the command and function tables to implement the extensions presented here.

The routines fall into three independent catagories: cassette tape I/O, video I/O for the Processor Technology VDM board, and a set of four routines to perform IN, OUT, PEEK, and POKE ala MITS Basic. The cassette routines are intended for use with a Tarbell Cassette Interface, but shouldn't be too difficult to adapt to any serial storage device with a little rewriting.

The listings are presented in a "pseudo assembled" form as opposed to an absolute form for two reasons'

1. Since there are two versions of Tiny Basic available the one from *Dr. Dobb's* and one published in *Interface Age* more recently, the absolute addresses intended for one of the versions wouldn't necessarily work for the other.
2. Many of you have undoubtedly implemented some extra features of your own already (into the *Dr. Dobb's* version) and would need to relocate additional routines into higher memory anyway.

My own particular configuration, for those of you with the *Dr. Dobb's* version who may be interested, is as follows:

I used the free space from 074F to 07FF for my VDM routine (I call it ZVDM) and a TTY output routine (ZTTY). The latter was an overflow from the OUTC section, where I didn't have enough room to check for the TTY output switch, output to the TTY, *and* output to the VDM. This problem doesn't occur if you use Dr. Wang's own VDM driver, published in a subsequent issue of *Dr. Dobb's*, but I preferred to use my own VDM driver due to some of its special features.

Since the ZVDM and ZTTY used up all the space alloted for customized routines, I simply changed the several locations which specify TXTBGN (address of the start of the text save area). Then, each time more routines need inserting, all that needs to be changed are those same locations. They are, by the way:

For the *Dr. Dobb's* version: 022A-B, 0237-8, 061A-B
For the *Interface Age* version: 0136-7, 0145-5, 053E-F
(in low order first, high order last format)

These three locations are within Basic as it stands now. There are also three such locations within the cassette tape I/O routines to follow—two in 'SAVE,' and one in 'CHAIN,' making a total of six location-pairs that need modifying whenever the starting location of the text-save area is changed. If the idea seems a little troublesome, then you can always cheat and pull a little trick: change 5 of the six LXI instructions into LHLD instructions which reference the operand of the one remaining LXI instruction. Then only that one LXI instruction would ever need changing, instead of six!

By "semi-assembled" form I meant that all opcodes are defined for you, but you must fill in the operands of JMP, CALL, etc., instructions yourself. Once you have decided where the routines are to reside (they may all be put into a single continuous block—probablly the best way to do it), then you fill in an address to the left of each line, corresponding to where the first byte (out of a possible 1, 2, or 3) or the instruction will reside. It may be desireable to leave NOP's between routines, just in case you ever desire to add bits and bytes to the code. Also, this padding may save a lot of sweat should you miscount the addresses and need to insert bytes afterward.

When finished, you'll have an exact image of the code, tailored for your software configuration, and it can be hex-loaded (or toggled, if you haven't yet written or acquired a hex-loader) right into where you want it.

The last chore that needs doing is to add the commands and functions to the appropriate tables. If you're using the *Dr.Dobb's* version, you simply insert the ascii string for the commands and functions, together with a two byte address of their first location (in HI-LO order, not the 8080-normal LO-HI) whose most significant bit has been set to one, in the appropriate tables. You may call the commands whatever you desire, but shortening the names (say, from SAVE, LOAD, CHAIN, and NAME to SV, LD, CHN, and NM) would leave more room in the tables for other things. I ended up running out of room for commands, so I took out some of the padding following the function table and transferred it to within the command table, shifting the function table starting address to higher memory in the process. The only reference to the function table is at location 04E7-04E8 (*Dr. Dobb's* version) which should be set to the new address of the first byte of the function table *minus one.* The instruction is the one at label EXPR4. This procedure is only necessary if you need more space in the command table, and happen to have plenty of extra space in the function table. If you are running the *Interface Age* version of Basic, then you must shift the entire command and function tables from their present locations to a free location, leaving plenty of space between the two tables. In that case, there are a few instructions referring to TAB1, TAB2, and TAB4 which will need modification. When that is accomplished, set TXTBGN (in all the places it needs setting) to the first free byte of text area. Some of the routines presented here may be inserted into the space vacated by the command and function tables, so that it won't be a waste of space. When all the tables and routines are intact, you're up and running!

Tape I/O

Four commands have been written to handle the naming, saving and loading of programs, utilizing the Tarbell Cassette Interface. The commands all center around a ten-character program name, stored in a special buffer in memory called the "name register." Whenever you save a program on tape, the dump is preceded by this ten character name. When loading from tape, you tell the machine to search for a particular program by inserting that program's name into the name register, and then executing the LOAD command. Basic will then proceed to compare the name register's contents to whatever comes in from tape, until a match is found. The restrictions on a program name are as follows:

1. It cannot exceed 10 characters in length. Leading blanks are ignored when setting the name, but any blank following the first non-blank character become a part of the name.
2. It cannot begin with a '?'. Any other non-blank character (except, of course. Basic's input routine control characters and some VDM controls) are legal, including most control characters. These, in fact, provide an elegant method of 'protection' of programs from unauthorized use, since no one can see the printout of the program name.

Examples: " NAME1" equivalent to "NAME1"
"NAME 1" is not equivaltne to "NAME1"
"?ABLE" is illegal; "?" not allowed as first character.
"THIS IS A TEST" gets truncated to "THIS IS A"

If you enter less than 10 characters, blanks will always be padded in on the right for you. If you want the current program name to be printed following every RUN and LIST command, and for it to be cleared (to all blanks) following every execution of the NEW command, then change the following 3 sections within Basic:

addr. in *Dr. Dobb's* version	addr. in *I.A.* version	change to this
0233-0234-0235	0141-0142-0143	CALL RNAME
0267-0268-0269	0175-0176-0177	CALL LNAME
0226-0227-0228	0132-0133-0134	CALL NNAME

Command Syntaxes and Operation

1. NAME command
   syntax:
   NAME (name) return
   or
   NAME? return

Sets the name register to the given name, where (name) follows the rules for a program name given above.

Examples:

NAME WIMPY return	(OK)
NAME WIMPY return	(same as above)
NAME A BIG DEAL return	(OK)
NAME YOUR PRICE! return	(exclamation point is lost)

Typing NAME? return causes Basic to respond with the name currently occupying the name register.

2. LOAD command
   syntax:
   LOAD (name) return
   or
   LOAD return

Loads a program from cassette tape. If the name is not specified, Basic assumes that you are looking for the program whose name is currently occupying the name register. If a name *is* specified, then Basic first replaces the contents of the name register with the given name, and then searches for it on the tape.

There is no need to set the tape player running in any particular place before the command is typed. In fact, you'll find it easier to first type the command, then hit return, and only then set the tape going. The tape may be started anywhere, and Basic will skip through any random programs that may pass by before the one named in the name register. Of course, if you know where a program is located on the tape, it would speed things up to set the tape going just before that point.

The LOAD command executes an automatic NEW before searching, so that the program will always get loaded at the start of text memory. The program resident before LOAD was typed goes into oblivion, of course. If you wish to append a routine from tape onto the tail end of the program currently active, see the CHAIN command.

PROGRAM LOADED
OK

gets printed. If the program does not load correctly, you will be treated to the message:

CHECKSUM ERROR; RELOAD
OK

whereupon you may try it again. Note that all you need to type the second time is:

LOAD return

because the name has already been "registered" from the first attempt. Should you repeatedly get checksum errors, or no response at all, then try adjusting your Tarbell or your player as per the Tarbell manual.

For IMSAI owners, there is a special added bonus. Whatever the input routine sees on tape is echoed to the programmed output leds on your front panel, so that by watching the behaviour of the leds, you can tell whether a program has been found yet and is in the process of loading, or if it has not been found yet. You can also tell whether the Tarbell has lost a bit: Since bit 7 is usually set to 0 in ASCII, then programmed output bit 7 should stay pretty solidly lit (reverse IMSAI logic) during the load process. If this doesn't happen, then one of the other bits will appear steady and you can bet that the interface has dropped a bit or two.

Typing a control C at any time during a load will terminate it and restart Basic.

Examples:

LOAD JOHN return	(OK)
NAME HORSES	
LOAD return	(this combination will search for HORSES)
LOADED DICE return	(will search for a program called "ED DICE")

3. SAVE command
   syntax:
   SAVE (name) return
   or
   SAVE return

Saves a program on tape. If the name is omitted, Basic uses the name currently in the name register. Specifying the name alters the name register and then performs the save under the new name. After hitting return, Basic will say:

HIT RETURN WHEN READY:

whereupon you position the tape to the desired point, hit record, and hit return when ready to save. Basic will prompt OK when finished. Control C aborts the save.

Examples:

SAVE THIS MESS return	(names the current program "THIS MESS" and saves it)
SAVE THE TIGER return	(names the program "THE TIGER" and saves)
etc.	

4. CHAIN command
   syntax:
   CHAIN (name) return
   or
   CHAIN return

Loads the program into memory following the last byte of the program currently in activity. The command is identical to LOAD, except that the automatic NEW is suppressed. It is useful for loading common subroutines.

Restriction: the lowest numbered statement of the CHAINed program must be greater than the highest numbered line of the program currently in memory. Exceptions to this will give Basic (and its users) headaches in text editing.

Example:

CHAIN RULE return	(loads a program named "RULE" onto the end of the program currently in memory, and also changes the name register to "RULE". If this is not desired, you can change the program name back to what it was with the NAME command.)

That's it for the tape I/O routines. May you never need to key in a Basic program twice again.

Special Functions

This section consists of two functions and two commands, all based on their MITS 8K Basic pseudo-counterparts (meaning that they work toward the same ends, but do it differently).

1. PEEK function
   syntax:
   PEEK (expr.)

This function returns a value from 0 to 255 corresponding to the data stored in the memory address specified by the argument expression. In order to PEEK at any address in the 8080's address space,

a. Take the decimal value of the address you want (0-65535).
b. If it is less than or equal to 32767, leave it alone. If it is greater than 32767, then subtract it from 65535, add one, and negate that result.
c. The result from step b. is the argument to use with the PEEK function.

Example:

to set the variable X to the contents of location 0D97.
a. 0D97 is 3479 decimal (7 plus 9*16 plus 13*256)
b. 3479 is less than 32767; leave it alone
c. X = PEEK (3479) performs the assignment.

Another example:

to see location CC00:
a. CC00 is 52224 in decimal.
b. (65535-52224) is equal to 13311; 13311 plus 1 is 13312; negative of that is −13312.
c. PRINT PEEK (−13312) displays the location's contents.

The strange values which result for addresses greater than 32767 are a result of two's complement 16-bit values being used by Basic.

2. IN function
   syntax:
   IN (expr.)

This function returns whatever happens to be present at the input port whose number is specified by the low-order byte of the 16-bit binary equivalent of the given decimal expression. In other words, you give it a value from 0 to 255, and it comes back with the data at that input port.
Example: to set M equal to the data on the sense switches:

```
M = IN (255); REM 255 IS THE SENSE SWITCH
    INPUT PORT
```

3. POKE command
   syntax:
   POKE expr. return

This command presupposes that you have set the variable P equal to a value between 0 and 255. It then deposits that value at the memory address specified by the expression. To compute the decimal equivalent of a given memory location, use the procedure outlined in the PEEK function description.
Example: fill the VDM memory with ASCII blanks (VDM memory is located at address CC00):

```
10 P = 32; REM 32 IS AN ASCII SPACE
20 FOR I = -13312 TO -12289; REM VDM RANGE
   (CC00-CFFF)
30 POKE I  ; REM POKES VALUE OF P AT LOCA-
   TION I
40 NEXT I
```

4. OUT command
   syntax:
   OUT expr. return

This command outputs the value of variable D to the output port specified in "expr."
Example:
Turn on all the programmed output leds on an IMSAI:

```
10 D = 0; REM A = 0 TURNS LEDS ON
20 OUT 255; REM LEDS ARE AT OUTPUT
   PORT FF.
```

NOTE: For the commands POKE and OUT, only the low order 8 bits of variables P and D, respectively, are used. Thus, if P or D were equal to 3, they would have the same effect in POKE or OUT as they would with the values 259, 515, 771, 8963, etc.

VDM I/O

The program I call "ZVDM" is a "character output to video" routine using Processor Technology's VDM-1 video interface board. The program will reside at location 074F, for those of you with the *Dr. Dobb's* version of Tiny Basic, and there is ample room between the end of it and location 0800 to insert the ZTTY routine to accommodate both a TTY (via the ZTTY routine) and the VDM. This is a general purpose routine; it will run with anything that supplies it with a character in A and a stack in SP.

ZVDM is a scroll-based video driver. The cursor originates at the bottom left of the screen, scans across for a printed line, and returns to the beginning while everything scrolls up one line. When given continuous data (a listing, or extremely wordy run), scrolling is held to a maximum of about 4 lines per second, utilizing the VDM on-board one-shot timer.

The output rate may be increased to full speed by typing the F key during output, or may be halted momentarily by typing the space bar. If the keyboard you are using happens to latch its data, then typing any other character (after you have sped up the display with F or stopped it with a space bar) resumes the normal 4 lines/second rate. If your keyboard is not latched, then simply letting go of the special key (F or space) returns to the normal speed.

There are two other special VDM control characters: Whenever a control Z is received, ZVDM will clear the screen and restart Basic. Typing a control V shifts the effective bottom of the screen up by one line. Since implementing this feature, though, I cannot say honestly that I have found any practical use for it.

Carriage return and linefeed characters (CR and LF) function just as if the VDM were a teletype, except that you shouldn't linefeed from anywhere except column one. Violating this may cause your cursor and subsequent text to be invisible, since I use VDM hardware blanking to curtain off the portion of every line following a CR.

When ZVDM receives a backarrow (ASCII 5F, or underscore), it physically backspaces on the screen, wiping out the previous character. If the Basic you are using currently interprets rub-outs as character deletes, or any other special character for that function, then I suggest that you change it to recognize 5F's for that purpose. Then, change the "print" handling routines to recognize, say, a $ for the linefeed suppression character within a print statement. Another feature of ZVDM is automatic wraparound. You may print many continuous lines without a carriage return, without any ill effects.

Following the ZVDM listing is my rewrite of the OUTC routine. Simply deposit it over the existing OUTC routine, and then insert ZVDM and ZTTY wherever you want them to go, inserting their address at the appropriate CALLS in OUTC.

When running, control 0 should still turn teletype output on and off, but VDM output should never be affected. VDM dip-switch options should be set to: cursor on, blanking enabled, and control characters blanked. The other options are, well, optional.

Last and least, you'll probably need to hit a control Z immediately after starting up Basic, in order to clear all the garbage out of the VDM memory, and initialize the scroll control port. The listings to all the described routines follow. HAVE FUN!!

"SAVE" ROUTINE

addr.	op code	mnemonic		
____	EF	SAVE:	IGNBLK	rst 5
____	FE 0D		CPI 0D	new name specified?
____	C4 __ __		CNZ GNAME	if so, insert it at the NR
____	11 __ __		LXI D,ZST1	get message
____	3E 3A		MVI A,':'	last char of message
____	CD __ __		CALL PRTSTG	TB system routine
____	CD __ __		CALL GETLIN	TB system routine
____	3E 3C		MVI A,3C	Tarbell start byte
____	CD __ __		CALL COUT	
____	3E E6		MVI A,E6	Tarbell sync byte
____	CD __ __		CALL COUT	
____	11 __ __		LXI D,PNAME	save program name
____	1A	WNAM:	LDAX D	
____	CD __ __		CALL COUT	
____	13		INX D	
____	FE 0D		CPI 0D	
____	C2 __ __		JNZ WNAM	
____	21 __ __		LXI H,TXTBGN	addr. of text area
____	7D		MOV A,L	negate it
____	2F		CMA	
____	6F		MOV L,A	
____	7C		MOV A,H	
____	2F		CMA	
____	67		MOV H,A	
____	EB		XCHG	save in DE
____	2A __ __		LHLD TXTUNF	get(addr of last byte)+1
____	19		DAD D	compute TXTUNF-TXTBGN
____	23		INX H	
____	7D		MOV A,L	output the length to tape
____	CD __ __		CALL COUT	
____	7C		MOV A,H	
____	CD __ __		CALL COUT	
____	EB		XCHG	and keep it in DE
____	21 __ __		LXI H,TXTBGN	get start of text
____	06 00		MVI B,0	initialize checksum
____	CD __ __	DMP:	CALL SCHK	check for control C
____	7E		MOV A,M	get data byte
____	CD __ __		CALL COUT	save on tape
____	80		ADD B	add to checksum
____	47		MOV B,A	and save new checksum
____	23		INX H	bump pointer
____	1B		DCX D	de-bump count
____	7A		MOV A,D	finished?
____	B3		ORA E	
____	C2 __ __		JNZ DMP	
____	78		MOV A,B	yes. save checksum
____	CD __ __		CALL COUT	
____	C7		RST 0	and restart Basic (or JMP)
____	F5	COUT:	PUSH PSW	save a byte
____	DB 6E	CLOP:	IN 6E	Tarbell status port
____	E6 20		ANI 20	ready to write?
____	C2 __ __		JNZ CLOP	
____	F1		POP PSW	yeah. get data
____	D3 6F		OUT 6F	give it to Don
____	C9		RET	

"CHAIN" AND "LOAD" ROUTINES

____	EF	CHAIN:	IGNBLK	rst 5
____	FE 0D		CPI 0D	if name is given,
____	C4 __ __		CNZ GNAME	process it.
____	3E 10	FIND:	MVI A,10	reset Tarbell
____	D3 6E		OUT 6E	
____	21 __ __		LXI H,PNAME	get address of name register
____	CD __ __	FLOP:	CALL INB	get byte from Tarbell
____	BE		CMP M	match name?
____	C2 __ __		JNZ FIND	if not, start over
____	23		INX H	if yes, try next character
____	FE 0D		CPI 0D	oh, if it was CR, we found name.
____	C2 __ __		JNZ FLOP	if not CR, go check next char.
____	CD __ __		CALL INB	get program length:
____	5F		MOV E,A	low byte
____	CD __ __		CALL INB	first,
____	57		MOV D,A	high byte last.
____	2A __ __		LHLD TXTUNF	this is where the program'll go.
____	06 00		MVI B,0	reset checksum
____	CD __ __	LODE:	CALL INB	get a byte
____	77		MOV M,A	store it
____	80		ADD B	add to checksum
____	47		MOV B,A	save new checksum
____	1B		DCX D	decrement count
____	23		INX H	bump pointer
____	7A		MOV A,D	are
____	B3		ORA E	we
____	C2 __ __		JNZ LODE	done?
____	CD __ __		CALL INB	yes. get checksum from tape
____	B8		CMP B	does it match ours?
____	CA __ __		JZ OK	if so, don't be nasty
____	11 __ __		LXI D,ZST2	be nasty.
____	AF		XRA A	tell user to give it another try
____	CD __ __		CALL PRTSTG	TB system routine
____	C7		RST 0	restart basic (can be JMP)
____	22 __ __	OK:	SHLD TXTUNF	save new free space pointer
____	11 __ __		LXI D,ZST3	announce success
____	AF		XRA A	
____	CD __ __		CALL PRTSTG	TB system routine
____	C7		RST 0	
____	CD __ __	INB:	CALL SCHK	check for control C
____	BD 6E		IN 6E	Tarbell status
____	E6 10		ANI 10	and off nonimportant bits
____	C2 __ __		JNZ INB	check some more if no data yet.
____	DB 6F		IN 6F	Tarbell data port.
____	D3 FF		OUT FF	echo to leds on IMSAI
____	C9		RET	
____	21 __ __	LOAD:	LXI H,TXTBGN	reset TXTUNF
____	22 __ __		SHLD TXTUNF	
____	C3 __ __		JMP CHAIN	and do a CHAIN

"NAME" and friends

```
                        NAME:   IGNBLK          does user want to see name?
EF
FE 3F                           CPI '?'
C2 __ __                        JNZ NIN         if not, he must want new one.
11 __ __                        LXI D,PNAME     yes. print
AF                              XRA A              out the
CD __ __                        CALL PRTSTG           name register
C7                              RST 0                    and restart(or JMP)
CD __ __                NIN:    CALL GNAME      set a new name
C7                              RST 0           and restart (or JMP)


                * THIS ROUTINE IS CALLED FROM NAME, LOAD,
                * AND CHAIN TO SET THE NAME REGISTER TO THE
                * STRING AT (DE), THE TEXT POINTER OF BASIC.
                *
EF                      GNAME:  IGNBLK          rst 5
FE 0D                           CPI 'CR'
CA __ __                        JZ AHOW         complain if no name (AHOW is
CD __ __                        CALL NNAME2     clear NR.  (TB system rou)
21 __ __                        LXI H,PNAME     get addr.of name register
06 0A                           MVI B,0A        get byte count
1A                      LOP:    LDAX D          get char from line buffer
FE 0D                           CPI 'CR'        carriage return?
C8                              RZ              if yes, we are done.
77                              MOV M,A         if no, insert another char.
23                              INX H           bump
13                              INX D               pointers
05                              DCR B                  and counter
C2 __ __                        JNZ LOP         too many characters in name?
C9                              RET             if yes, return


                * THIS ROUTINE CHECKS FOR CONTROL C BEING
                * PRESSED AT THE KEYBOARD DATA PORT.
                *
DB __                   SCHK:   IN xx           your keyboard data port
FE 03                           CPI 03          control C?
C0                              RNZ             do nuthin if not
C7                              RST 0           restart (or JMP) if yes.


                * FOLLOWING ARE THE NAME REGISTER AND TEXT STRINGS
20 20 20 20 20 20 20 20 20 20 0D        PNAME: 10(space),CR
48 49 54 20 52 45 54 55 52 4E 20        ZST1: "HIT RETURN
57 48 45 4E 20 52 45 41 44 59 3A              WHEN READY:"
43 48 45 43 4B 53 55 4D 20 45 52        ZST2: "CHECKSUM ER
52 4F 52 3B 20 52 45 41 4C 4F 44              ROR; RELOAD",CR
0D
50 52 4F 47 52 41 4D 20 4C 4F 41        ZST3: "PROGRAM LOA
44 45 44 0D                                   DED",CR

        * THE NEXT ROUTINE IS A PATCH FROM "RUN", TO PRINT
        * OUT THE PROGRAM NAME BEFORE EACH PROGRAM EXECUTION.

CD __ __        RNAME:  CALL ENDCHK     TB system routine
11 __ __        IT:     LXI D,PNAME     get address of program name
AF                      XRA A           clear A for PRTSTG
CD __ __                CALL PRTSTG     TB system routine
CD __ __                CALL CRLF       TB system routine
C9                      RET
```

```
        * THE NEXT ROUTINE IS A PATCH FROM "NEW", TO CLEAR THE
        * NAME REGISTER EVERY TIME THE 'NEW' COMMAND IS EXECUTED.
        * NNAME2 IS CALLED FROM OTHER PLACES THAT NEED THE NAME
        * REGISTER CLEARED.
CD __ __        NNAME:  CALL ENDCHK     TB system routine
21 __ __        NNAME2: LXI H,PNAME     get name register address
06 0A                   MVI B, 0A       set count
36 20           CLNM:   MVI M,20        insert ASCII space
05                      DCR B
23                      INX H
C2 __ __                JNZ CLNM        keep going till name register
36 0D                   MVI M,0D        trailing CR.            is clear.
C9                      RET             return
        * THE NEXT ROUTINE IS A PATCH FROM "LIST", TO PRINT
        * OUT THE PROGRAM NAME BEFORE EVERY LISTING.

E5              LNAME:  PUSH H          save
D5                      PUSH D              junk
CD __ __                CALL IT         print program name
D1                      POP D           get back
E1                      POP H               junk
CD __ __                CALL FNDLN      TB system routine ( this call used
C9                      RET             go back to LIST.    to be where
                                                            the call to
                                                            LNAME is now,
                                                            within the
                                                            LIST routine.)
```

"PEEK", "POKE", "IN", "OUT" ROUTINES

```
CD __ __        IN:     CALL PARN       TB system routine
7D                      MOV A,L         get port #
32 __ __                STA DATIN+1     force as operand of IN instruction.
DB 00           DATIN:  IN xx           get data (operand set by STA above)
6F                      MOV L,A         leave in HL
26 00                   MVI H,0           (clear H)
C9                      RET


DF              OUT:    EXPR            TB routine; get port # in HL.
7D                      MOV A,L         get in A
32 __ __                STA OUTPT+1     force as operand of OUT instruction
31 __ __                LXI H,VARBGN    TB system constant
3E 08                   MVI B,08        get value
85                      ADD L               of variable D
6F                      MOV L,A               in A
7E                      MOV A,M
D3 00           OUTPT:  OUT xx          and output it.
F7                      FINISH          rst 6


CD __ __        PEEK:   CALL PARN       TB system routine
7E                      MOV A,M         get what we seek in A
6F                      MOV L,A          and put
AF                      XRA A               it into
67                      MOV H,A               HL
C9                      RET                     and go home.


DF              POKE:   EXPR            get address
E5                      PUSH H          save it
21 __ __                LXI H,VARBGN    system variable
3E 20                   MVI A,20        get value
85                      ADD L              of variable
6F                      MOV L,A              P into
7E                      MOV A,M                the accumulator
E1                      POP H           get back address
77                      MOV M,A         poke
F7                      FINISH          rst 6
```

"ZVDM" ROUTINE

```
____ E5          ZVDM:  PUSH H          save H
____ F5                 PUSH PSW        push character
____ 2A __ __           LHLD VDMP       pointer to current screen pos.
____ 7E                 MOV A,M         get character at pointer
____ E6 7F              ANI 7F          strip cursor bit
____ FE 0B              CPI 0B          was it a VT?
____ C2 __ __           JNZ VDM2
____ 3E 0D              MVI A,0D        yes. change to CR
____ 77         VDM2:   MOV M,A         move back to screen
____ F1                 POP PSW         get back character to be output.
____ F5                 PUSH PSW        repush
____ E6 7F              ANI 7F          strip bit 7
____ FE 0D              CPI 'CR'        carriage return?
____ CA __ __           JZ CR           if yes, go process carriage ret.
____ FE 5F              CPI 5F          backspace?
____ C2 __ __           JNZ NXT         if not, go away
____ 7D                 MOV A,L         yes, backspace
____ E6 3F              ANI 3F               from column one?
____ CA __ __           JZ VDM7              if yes, ignore it.
____ 2B                 DCX H           decrement pointer
____ C3 __ __           JMP VDM7        finish up
____ FE 16      NXT:    CPI 16          control V?
____ CA __ __           JZ SCRL         if yes, shift screen bottom up
____ FE 0A              CPI 'LF'        linefeed?
____ CA __ __           JZ LF           if yes, go do it
____ FE 1A              CPI 1A          control Z?
____ C2 __ __           JNZ WEED        if not, go away
____ 21 00 CC           LXI H, CC00       yes.
____ 36 20      CLOOP:  MVI M,20          clear
____ 23                 INX H             the screen
____ 7C                 MOV A,H           and
____ DE D0              SBI D0            restart
____ C2 __ __           JNZ CLOOP         Basic
____ C7                 RST 0             (may be a JMP) **
____ FE 20      WEED:   CPI 20          meaningless control character?
____ DA __ __           JC VDM7         yes. skip it.
____ 77                 MOV M,A         no,write it.
____ 23                 INX H           bump pointer
____ C3 __ __           JMP VDM5        go test for end of line
____ 36 0D      CR:     MVI M,'CR'      carriage return routine
____ 7D                 MOV A,L         reset pointer
____ E6 C0              ANI C0              to start
____ 6F                 MOV L,A             of current line
____ 7E                 MOV A,M         stick cursor
____ F6 80              ORI 80               over the first
____ 77                 MOV M,A              character on the line
____ C3 __ __           JMP VDM7A       and go finish
____ 7D         LF:     MOV A,L         linefeed routine
____ C6 40              ADI 40          bump
____ 6F                 MOV L,A            pointer
____ 3E 00              MVI A,0                by
____ 8C                 ADC H                    64 (decimal)
____ 67                 MOV H,A                    positions
____ C3 __ __           JMP VDM5A       go check for end of VDM memory
____ 3E 3F      VDM5:   MVI A,3F        check for end of line
____ A5                 ANA L              if not,
____ C2 __ __           JNZ VDM7           don't do anything
____ 7C         VDM5A:  MOV A,H         yes, end of line or screen
____ FE D0              CPI D0            if we overwrote the
____ C2 __ __           JNZ SCRL               1K of VDM memory, reset
____ 26 CC              MVI H,CC               the pointer to the start.
____ 3A __ __   SCRL:   LDA VDMS        compute
____ C6 01              ADI 01               new
____ E6 0F              ANI 0F                  scroll control
____ F5                 PUSH PSW                   byte.
____ DB 02      WAIT:   IN 02           check keyboard (b.2 is your data p.
____ FE 46              CPI 'F'         if F is pressed, don't
____ CA __ __           JZ ZOUT             waste any time
____ FE 20              CPI ' '         if space bar is pressed, wait
____ CA __ __           JZ WAIT         until it isn't anymore.
____ DB C8      W2:     IN C8           has 1/4 second passed by since
____ FE FF              CPI FF            last output to scroll port?
____ CA __ __           JZ W2           if not, go wait some more.
____ F1         ZOUT:   POP PSW         get scroll control byte
____ 32 __ __   VDM6A:  STA VDMS        save it.
____ D3 C8              OUT C8          output it to VDM
____ 36 8B      VDM7:   MVI M,8B        store a VT character at the new
____ 22 __ __   VDM7A:  SHLD VDMP          cursor position
____ F1                 POP PSW         restore registers
____ E1                 POP H
____ C9                 RET             and go home.
____ 00         VDMS:   DB 00           scroll control byte
____ C0 CF      VDMP:   DW CFC0         VDM memory pointer
```

** The RST 0 instruction may be changed to JMP VDM6A if this VDM driver is to be used with something other than Tiny Basic. Or, if you want to use this with Basic but don't want Basic restarted after every control Z, the above modification will eliminate the restart. If used with MITS 8K Basic, for example, you could then clear the screen by executing 'PRINT CHR$(31)', without effecting Basic.

"OUTC" ROUTINE, MODIFIED TO HANDLE ZVDM

```
* THIS ROUTINE CAN REPLACE THE OLD OUTC FROM
* TINY BASIC. THE ADDRESSES HERE ARE FILLED IN, FOR
* THE BENEFIT OF THOSE OF YOU WITH THE DR. DOBB'S
* VERSION OF T.B., BUT THE CODE APPLIES EQUALLY (EXCEPT
* FOR THE ADDRESSES) TO THE INTERFACE AGE VERSION.
*
* THE ZTTY ROUTINE WILL NOT FIT IN THE SPACE ALLOTED FOR
* OUTC, BUT MAY BE PUT ANYWHERE ELSE WHERE YOU ALLOCATE
* ROOM FOR IT.

0010 F5                 PUSH PSW
0011 3A 00 08           LDA OCSW
0014 B7                 ORA A

071A C2 __ __   OC2:    JNZ ZTTY        if switch is on, go do TTY
071D F1                 POP PSW         if not, don't
071E CD __ __ 00 DISP:  CALL ZVDM       output character to VDM
0721 00 00 00 00        NOP 8           not quite enough room for a
     00 00 00 00                        TTY routine, so we leaveit.
0729 FE 0D              CPI 'CR'        the rest of this is pretty
072B C0                 RNZ                much the same as it
072C 3E 0A              MVI A,'LF'         was before.
072E D7                 OUTC
072F 3E 0D              MVI A,'CR'
0731 C9                 RET


____ DB xx      ZTTY:   IN xx           your status port
____ E6 xx              ANI xx          mask off the bits that need it.
____ CA __ __           JZ TTY          or may be JNZ, who knows?
____ F1                 POP PSW         get data
____ D3 xx              OUT xx          output to your data port.
____ C3 __ __           JMP DISP        go back to within the OUTC
                                          routine. The address would
                                          be 071E if you use the OUTC
                                          version above.
                                        Those of you with the Dr.
                                          Dobb's version will find
                                          that ZVDM and ZTTY will fit
                                          together, as a unit, at
                                          locations 074F-07FF.
```

SOME USEFUL SYSTEM ADDRESSES

ROUTINE	DOBB'S VERSION ADDRESS	I.A. VERSION ADDRESS
PRTSTG	063C	0560
GETLIN	05D6	04FA
CRLF	000E	000E
ENDCHK	05A0	04C2
FNDLN	0614	0538
PARN	04FB	041A
AHOW	00A0	00A0

TWO S-100 GRAPHICS PRODUCTS COMPARED

Dear Sir: *Received: 78 May 12*

Since I have benefitted so frequently from the philosophy of *DDJ*'s communications I wish to pass along an evaluation which may help someone else decide between two expensive graphics interfaces for the S-100 bus. The Super Dense Merlin and the Matrox ALT-256**2 cost about the same but have different features which may not be obvious until you have gone to the expense of purchasing both. I already had an established computer system and needed a graphics interface as part of an economical prototype suited for self-instruction by simulations of physiological systems.

Initially the Merlin was the only alternative so I invested in the double board and a third tier to get 320 H by 200 V dot resolution graphics plus ASCII characters. The factory-assembled board was complete and met specifications but it was very exasperating to implement in any of my collection which includes two Altairs, a TDL ZPU, and a SOL. A major source of difficulty was associated with its DMA on the S-100 bus, a familiar problem to many these days. A major part of my memory investment was not compatible, the Z-80 was usable, but not the 8080A when the DMA was enabled. In the SOL (with no front switches) I could not disable the Merlin DMA to even initialize the display parameters.

In the meantime the ALT-256**2 became available, and to meet a pressing deadline I took a chance on a second investment in it. This factory-assembled board had a solder bridge which kept two of the port decoding bits high and made the screen respond to addresses as commands (how did that pass a "24-hour burn-in"?). Once this was found and removed it has been sheer joy to apply the graphics. Within a day I made up for several months of lost time.

The Merlin can be configured for keyboard and cassette tapes, may have an on-board ROM monitor and editor; thu. it can be a basic operating system. The Matrox, on the other hand, displays dots on a screen, and nothing more. A comparison of the features of each follows, viewed with a bias of my particular needs.

The Merlin DMA slows down the CPU to 43% normal operation speed as the video signal is being generated whereas the Matrox does not. If there is any flicker in either one this is not noticeable and not as important as the computation speed in a simulation application. The user must supply 8K of memory for the Super Dense graphics whereas the Matrox has 8K of dynamic RAM included on board (a costly component).

On the Merlin, dot information is in successive 8-bit memory locations, the addresses of which do not easily relate to the X and Y coordinates on the CRT screen; but since the Matrox has 256 x 256 dots the dot addresses correspond nicely with the position on the screen. The CPU communicates with Merlin by memory addresses (not moveable if the supplied monitor is used) whereas the Matrox uses 4 output ports and 1 status input port. For the latter one output sets the X, another sets the Y, and a third turns the dot off or on. A fourth output clears the screen within microseconds whereas the Merlin requires a software routine lasting seconds.

The Merlin has several display options including hardware ASCII of 16 lines of 40 characters; three different densities of graphics, and both mixed characters-graphics and splitscreen displays of the two. The Matrox requires software generated ASCII characters to be placed in memory, however this board may be synchronized with an ASCII board to give mixed displays. Also several graphics boards may be synchronized to give multiple levels of intensity to each dot.

If you want to get an integrated operating system and graphics package and expect no DMA problems, the Merlin is worth your consideration. If you want to add graphics to an existing system then the ALT-256**2 is apt to be more suitable. With the growing interest in graphics capabilities, competition is apt to produce packages at less cost than either of the ones described here.

Yours Truly,
James E. Randall

609 South Jordan
Bloomington, IN 47401

SELECTOR II REPORTING SYSTEM

News Release *Received: 78 April 28*

Selector II is a data entry and demand reporting system for users of CP/M and Microsoft Extended Disk BASIC. It provides timely information at the user's fingertips, as well as allowing on-line updating of data files.

Unique reports can be generated in minutes, with no programming experience required. The required information is simply selected with any variety of conditions desired and in the order wanted. Output can be used by programs, displayed or printed as a titled, columnized report.

One-time license price of $225 includes 1-yr warranty. Selector II is available on single-density 8" diskette complete with user's manual from Micro-Ap, 8939 San Ramon Road, Dublin, CA 94566; (415) 828-6697.

QUAY 80 F1 – S-100 DISK SYSTEM

News Release *Received: 78 May 15*

A floppy disk system for use in S-100 bus computers is available from Quay Corp., P.O. Box 386, Freehold, NJ 07728. The Quay 80 F1 system, priced at $795. includes the Q/80 FDC – floppy disk controller board (capable of supporting up to four disks), QDOS – disk based operating system, the Q/FD1 125 KB 5¼" band-driven disk drive with power regulator and interface cable, and the Q/80 FC – floppy disk cabinet. Add-on drives (Q/FD1) are priced at $395. each. In addition to the floppy disk support, the Q/FDC has available a programmable 8-bit, TTL compatible, parallel I/O port capable of supporting standard peripheral devices such as line printers, tape punches, keyboards, etc. Product availability is 30-60 days ARO.

A Z-80 Disassembler in TDL 8K BASIC

Dear Dr. Dobbs: *Received: 78 April 20*

In the March 1977 issue of *DDJ* you published an anonymous listing for an 8080 disassembler. For any student of BASIC this was a beautiful bit of work. I can't hope to come close to the original in compactness of code work but anyhow have tried to build on it as best as possible. Enclosed is the Z-80 version, written in TDL's 8K BASIC and generating TDL's mnemonics.

The line numbers are fairly well preserved from the original version but all of the comments have been omitted for memory compactness. You shouldn't have much trouble figuring out what was done from the original listing. Some changes were made for my own convenience. First, it was necessary to use "escape" characters to control the screen clearing, etc. on my relatively smart terminal. Second, the listing is controlled by a start and end address which are specified after the ">" sign. Third, the listing is continuous and I have eliminated the "J" routine. Fourth, the screen clearing activity is made to progress during the time that the files are being loaded. This goes with the smart terminal. Finally, the machine codes are printed along with the mnemonics. I still like to see the machine code since I grew up that way.

In the spirit of the original version, please omit my name and address from this article.

This is an example of how the disassembler works. It is disassembling a portion of the BASIC (TDL 8K) on which it is running, of course.

Note the limitation on memory address, so the BASIC had to be moved to a lower address for the disassembly.

```
40 DIM B(6)
50 GOSUB 9840:REM BASED ON 8080 DISASS.--DDJ MAR 1977
102 PRINT ES$;CH$;ES$;CS$:GOSUB 9030
110 PRINT "       Z-80 DISASSEMBLER":PRINT
160 PF=TRUE
210 PRINT ">";
215 GOSUB 220:GOSUB 5600
216 IF C$=" " OR C$="Z" OR C$="-" THEN GOTO 430
217 GOTO 210
220 GOSUB 5600
250 IF C$>="0" AND C$<="7" GOTO 300
260 IF PR THEN GOSUB 6200:GOTO 220
300 T=C-ASC("0"):N=1
310 GOSUB 5600
320 IF C$>="0" AND C$<="9" THEN D=C-ASC("0"):GOTO 400
330 IF C$>="A" AND C$<="F" THEN D=10+C-ASC("A"):GOTO 400
340 IF C=127 THEN PRINT CR$;ES$;EL$;:GOTO 210
350 IF PR THEN GOSUB 6200
360 GOTO 310
400 T=D+16*T:N=N+1
420 IF N<4 GOTO 310
425 RETURN
430 PC=T
440 IF PC<=32765 GOTO 450
445 GOSUB 530:GOTO 210
450 GOSUB 220:GOSUB 5600
452 IF C$=CR$ GOTO 455
454 GOTO 450
455 PE=T
456 IF PE<=32765 GOTO 900
530 PRINT "ADDRESS TOO HIGH":RETURN
900 PRINT CR$
1000 REM **MAIN LOOP**
1002 NB=0
1010 X=PC:GOSUB 5450
1020 PRINT ":  ";:PRINT TAB(15);
1030 GOSUB 5000:OP=X
1040 GOSUB 2060
1045 PRINT TAB(33);
1046 FOR I=1 TO NB:X=B(I):PRINT SPC(1);:GOSUB 5100:NEXT
1047 PRINT
1050 IF PC<PE GOTO 1000
1060 END
2000 L=INT(OP/BIT(6)):M=INT(OP/BIT(3)) AND 7:K=OP AND 7:RETURN
2060 GOSUB 2000:ON L GOTO 3010,3210,3510
2120 ON K GOTO 2210,2410,2510,2610,2610,2710,2810
2132 IF M=1 THEN S$="EXAF"
2133 IF M=0 THEN S$="NOP"
2134 IF M=1 OR M=0 GOTO 5700
2135 S$=JR$(M):GOSUB 6000
2136 GOSUB 5000:GOSUB 7000:GOSUB 5450:RETURN
2210 K=M/BIT(1)
2220 IF OP AND BIT(3) GOTO 2310
2240 S$="LXI ":GOSUB 6000
2250 S$=RP$(K):GOSUB 6000
2260 S$=", ":GOSUB 6000
2270 GOSUB 5400:RETURN
2310 S$="DAD ":GOSUB 6000
2320 S$=RP$(K):GOTO 6000
2410 S$=O2$(M):GOSUB 6000
2420 IF M<4 THEN RETURN
2440 GOSUB 6100:GOSUB 5400:RETURN
2450 GOSUB 5400:RETURN
2500 REM
2510 K=INT(M/BIT(1))
2520 M=M AND BIT(0)
2530 IF M=0 THEN S$="INX "
2540 IF M=1 THEN S$="DCX "
2550 GOSUB 6000
2560 S$=RP$(K):GOTO 6000
2610 IF K=4 THEN S$="INR "
2620 IF K=5 THEN S$="DCR "
2630 GOSUB 6000
2640 S$=REG$(M):GOTO 6000
2710 S$="MVI ":GOSUB 6000
2720 S$=REG$(M):GOSUB 6000
2730 S$=",":GOSUB 6000
2740 GOSUB 5200:RETURN
2810 S$=O7$(M):GOTO 6000
3010 IF M=6 AND K=6 GOTO 3160
3020 S$="MOV ":GOSUB 6000
3030 S$=REG$(M):GOSUB 6000
3040 S$="Z ":GOSUB 6000
3050 S$=REG$(K):GOTO 6000
3160 S$="HLT":GOTO 6000
3210 S$=AI$(M):GOSUB 6000
3220 GOSUB 6100:GOTO 3050:RETURN
3510 ON K GOTO 3570,3610,3805,3630,4005,4120,4210
3530 S$="R":GOSUB 6000
3540 S$=CC$(M):GOTO 6000
3570 S$=C1$(M):GOTO 6000
3610 S$="J":GOTO 3650
3630 S$="C"
3650 GOSUB 6000
3660 S$=CC$(M):GOSUB 6000
3680 GOSUB 6100
3690 GOSUB 5400
3695 JF=TRUE:RETURN
3805 IF M=1 GOTO 7200
3810 S$=C3$(M):GOSUB 6000
3820 IF M=0 GOTO 3850
3830 IF M=2 OR M=3 GOTO 3900
3840 RETURN
3850 GOSUB 6100:GOSUB 5400:JF=TRUE:RETURN
3900 GOSUB 6100:GOSUB 5200:RETURN
4005 IF M=5 GOTO 7400
4006 IF M=3 THEN I$="X"
4007 IF M=7 THEN I$="Y"
4008 IF M=3 OR M=7 GOTO 7700
4010 S$=C5$(M):GOSUB 6000
```

```
4020 IF M<>1 THEN RETURN
4040 GOSUB 6100:GOSUB 5400:JF=TRUE:RETURN
4120 S$=C6$(M):GOSUB 6000
4130 S$=" ":GOSUB 6000
4140 GOSUB 5200:RETURN
4210 S$="RST ":GOSUB 6000
4220 X=8*M:GOSUB 5100
4230 JF=TRUE:RETURN
5000 X=PEEK(PC):PC=PC+1:NB=NB+1:B(NB)=X:RETURN
5100 PRINT HEX$(INT(X/16));HEX$(X AND 15);:RETURN
5200 GOSUB 5000:GOSUB 5100:RETURN
5300 REM
5330 GOSUB 5000:XL=X
5340 GOSUB 5000:XH=X
5350 X=XL+256*XH:RETURN
5400 GOSUB 5300:GOSUB 5450:RETURN
5450 T=X:X=INT(T/256):GOSUB 5100
5470 X=T-256*X:GOSUB 5100
5480 X=T:RETURN
5500 WAIT TS,KB:C=INP(TD) AND 127
5525 IF C>=97 AND C<=122 THEN C=C-32
5530 C$=CHR$(C):RETURN
5600 GOSUB 5500
5640 PR=C>=32 AND C<=126
5650 IF PR THEN PRINT C$;:RETURN
5700 GOTO 6000
6000 IF PF THEN PRINT S$;" ";:RETURN
6100 IF PF THEN PRINT " ";:RETURN
6200 PRINT BS$;ES$;EL$:RETURN
7000 IF X<=127 GOTO 7040
7030 X=X-256
7040 X=PC+X:RETURN
7200 GOSUB 5000:OP=X
7212 L=INT(OP/BIT(6))
7214 M=INT(OP/BIT(3)) AND 7
7216 K=OP AND 7
7220 IF L=0 GOTO 7280
7230 S$=CB$(L):GOSUB 6000
7240 S$=CHR$(48+M):GOSUB 6000:S$=",":GOSUB 6000
7260 S$=REG$(K):GOTO 6000
7280 S$=RT$(M):GOSUB 6000
7300 S$=REG$(K):GOTO 6000
7400 GOSUB 5000:OP=X
7410 L=INT(OP/BIT(6)):M=INT(OP/BIT(3)) AND 7:K=OP AND 7
7430 ON L GOTO 7450,7690,7440
7440 S$="- -":GOSUB 6000:RETURN
7450 ONK GOTO 7480,7500,7550,7600,7620,7650,7680
7460 S$="INP":GOSUB 6000
7470 S$=REF$(M):GOTO 6000
7480 S$="OUTP":GOSUB 6000
7490 S$=REG$(M):GOTO 6000
7500 K=INT(M/BIT(1)):M=M AND BIT(0)
7520 IF M=1 THEN S$="DADC"
7530 IF M=0 THEN S$="DSBC"
7535 GOSUB 6000
7540 S$=RP$(K):GOTO 6000
7550 K=INT(M/BIT(1)):M=M AND BIT(0)
7560 IF M=0 THEN S$="S"
7570 IF M=1 THEN S$="L"
7575 GOSUB 6000
7580 IF K<>2 GOTO 7590 ELSE S$="-?-"
7585 GOTO 6000
7590 S$=RF$(K):GOSUB 6000
7592 GOSUB 5300:GOSUB 5450:RETURN
7600 IF M=0 GOTO 7610
7602 S$="--":GOTO 6000
7610 S$="NEG":GOTO 6000
7620 IF M<=1 GOTO 7630
7622 GOTO 7602
7630 IF M=1 THEN S$="RETI"
7632 IF M=0 THEN S$="RETN"
7640 GOTO 6000
7650 IF M=1 OR M>3 THEN S$="-!-"
7660 IF M=0 THEN S$="IM0"
7662 IF M=2 THEN S$="IM1"
7664 IF M=3 THEN S$="IM2"
7670 GOTO 6000
7680 S$=ED$(M):GOTO 6000
7690 S$=CP$(K):GOSUB 6000
7695 S$=CP$(M):GOTO 6000
7700 GOSUB 5000:OP=X
7705 IF OP=203 GOTO 7820
7710 L=INT(OP/BIT(6)):M=INT(OP/BIT(3)) AND 7
7712 K=OP AND 7
7714 ON L GOTO 7770,7790,7799
7720 ON K GOTO 7724,7736,7740,7750,7750,7760
7722 S$="==":GOTO 6000
7724 IF M=4 GOTO 7731
7725 K=INT(M/BIT(1)):M=M AND BIT(0)
7726 S$="DAD":GOSUB 6000:S$=I$:GOSUB 6000
7727 IF K=2 GOTO 7730
7728 S$=RP$(K):GOTO 6000
7730 S$=I$:GOTO 6000
7731 S$="LXI":GOSUB 6000:S$=I$:GOSUB 6000
7732 S$=",":GOSUB 6000:GOSUB 5330:GOSUB 5450:RETURN
7736 IF M=4 GOTO 7739
7737 S$="LI":GOSUB 6000:S$=I$:GOSUB 6000
7738 S$="D":GOSUB 6000:GOSUB 5330:GOSUB 5450:RETURN
7739 S$="SI":GOSUB 6000:S$=I$:GOSUB 6000:GOTO 7738
7740 IF M=4 THEN S$="INX"
7742 IF M=5 THEN S$="DCX"
7744 GOSUB 6000:S$=I$:GOTO 6000
7750 IF K=4 THEN S$="INR"
7752 IF K=5 THEN S$="DCR":GOSUB 7754:RETURN
7754 GOSUB 6000:GOSUB 5000:GOSUB 5100
7755 S$="(":GOSUB 6000:S$=I$:GOSUB 6000
7756 S$=")":GOTO 6000
7760 IF M<>6 GOTO 7722
7762 S$="MVI":GOSUB 7754:S$=" ":GOSUB 6000:GOSUB 5200:RETURN
7770 S$="MOV"
7772 IF K=6 GOTO 7780
7774 GOSUB 7754:S$=",":GOSUB 6000
7776 S$=REG$(K):GOTO 6000
7780 GOSUB 6000:S$=REG$(M):GOSUB 6000:S$="Z"
7782 GOSUB 7754:RETURN
7790 IF K<>6 GOTO 7794
7792 S$=AI$(M):GOSUB 7754:RETURN
7794 S$="--":GOTO 6000
7799 IF M<>4 GOTO 7810
7800 ON K GOTO 7804,7802,7805,7802,7808,7802,7802
7802 S$="--":GOTO 6000
7804 S$="POP":GOTO 7809
7805 S$="XTI":GOTO 7809
7806 S$="SPI":GOTO 7809
7808 S$="PUSH"
7809 GOSUB 6000:S$=I$:GOTO 6000
7810 IF M<>5 GOTO 7814:IF K<>1 GOTO 7802
7812 S$="PCl":GOTO 7809
7814 IF M=7 AND K=1 GOTO 7806
7820 GOSUB 5000:D=X:GOSUB 5000:OP=X
7822 GOSUB 2000
7826 IF L>0 GOTO 7840
7830 IF K<> 6 GOTO 7802
7832 S$=RT$(M):GOSUB 6000
7834 X=D:GOSUB 5100:S$="(":GOSUB 6000
7836 S$=I$:GOSUB 6000:S$=")":GOTO 6000
7840 S$=CB$(L):GOSUB 6000
7842 S$=CHR$(48+M):GOSUB 6000
7844 S$=",":GOSUB 6000:GOTO 7834
9030 DIM HEX$(15)
9040 FOR I=0 TO 15:READ HEX$(I):NEXT
9050 DATA "0","1","2","3","4","5","6","7","8","9",A,B,C,D,E,F
9080 DIM BIT(7)
9090 FOR I=0 TO 7:READ BIT(I):NEXT
9100 DATA 1,2,4,8,16,32,64,128
9130 DIM REG$(7)
9140 FOR I=0 TO 7:READ REG$(I):NEXT
9150 DATA B,C,D,E,H,L,M,A
9180 DIM RP$(3)
9190 FOR I=0 TO 3:READ RP$(I):NEXT
9200 DATA B,D,H,SP
9203 DIM RF$(3)
9204 FOR I=0 TO 3:READ RF$(I):NEXT
9205 DATA BCD,DED,HLD,SPD
9230 DIM AI$(7)
9240 FOR I=0 TO 7:READ AI$(I):NEXT
9250 DATA ADD,ADC,SUB,SBB,ANA,XRA,ORA,CMP
9280 DIM C6$(7)
9290 FOR I=0 TO 7:READ C6$(I):NEXT
9300 DATA ADI,ACI,SUI,SBI,ANI,XRI,ORI,CPI
9312 DIM JR$(7)
9314 FOR I=2 TO 7:READ JR$(I):NEXT
9316 DATA DJNZ,JMPR,JRNZ,JRZ,JRNC,JRC
9330 DIM CC$(7)
9340 FOR I=0 TO 7:READ CC$(I):NEXT
9350 DATA NZ,Z,NC,C,PO,PE,P,M
9390 DIM C1$(7)
9400 FOR I=0 TO 7:READ C1$(I):NEXT
9410 DATA POP B,RET,POP D,EXX,POP H,PCHL,POP PSW,SPHL
9440 DIM C3$(7)
```

```
9450 FOR I=0 TO 7:READ C3$(I): NEXT
9460 DATA JMP,- -,OUT,IN,XTHL,XCHG,DI,EI
9490 DIM C5$(7)
9500 FOR I=0 TO 7:READ C5$(I):NEXT
9510 DATA PUSH B,CALL,PUSH D,- -,PUSH H,- -,PUSH PSW,- -
9540 DIM O7$(7)
9550 FOR I=0 TO 7:READ O7$(I):NEXT
9560 DATA RLC,RRC,RAL,RAR,DAA,CMA,STC,CMC
9590 DIM O2$(7)
9600 FOR I=0 TO 7:READ O2$(I):NEXT
9610 DATA STAX B,LDAX B,STAX D,LDAX D,SHLD,LHLD,STA,LDA
9625 DIM CB$(3)
9630 FOR I=1 TO 3:READ CB$(I):NEXT
9640 DATA BIT,RES,SET
9655 DIM RT$(7)
9660 FOR I=0 TO 7:READ RT$(I):NEXT
9670 DATA RLCR,RRCR,RALR,RARR,SLAR,SRAR,--,SRLR
9680 DIM ED$(7)
9690 FOR I=0 TO 7:READ ED$(I):NEXT
9695 DATA STAI,STAR,LDAI,LDAR,RRD,RLD,--,--
9700 DIM CP$(7)
9702 FOR I=0 TO 7:READ CP$(I):NEXT
9704 DATA LD,CP,IN,OUT,I,D,IR,DR
9820 TRUE=-1:FALSE=0:RETURN
9840 TS=2:TD=3:KB=2:SB=2
9860 BS=8:BS$=CHR$(BS)
9870 CS=74:CS$=CHR$(CS)
9880 EL=75:EL$=CHR$(EL)
9890 CR=13:CR$=CHR$(CR)
9895 ES$=CHR$(27)
9897 CH$=CHR$(72):RETURN
```

```
READY>
RUN
HJ
       Z-80 DISASSEMBLER

>3000 301F
3000:          XTHL              E3
3001:          POP B             C1
3002:          XCHG              EB
3003:          XTHL              E3
3004:          INX  B            03
3005:          LD DR             ED B8
3007:          INX  H            23
3008:          INX  D            13
3009:          MOV  B ,  D       42
300A:          MOV  C ,  E       4B
300B:          POP D             D1
300C:          RET               C9
300D:          PUSH H            E5
300E:          LHLD  01B8        2A B8 01
3011:          MVI  B , 00       06 00
3013:          DAD  B            09
3014:          DAD  B            09
3015:          CALL  B11A        CD 1A B1
3018:          POP H             E1
3019:          RET               C9
301A:          PUSH D            D5
301B:          XCHG              EB
301C:          LXI  H ,  FFD8    21 D8 FF

READY>
```

COMPUTER NETWORK POLL

DDJ: ***Received: 78 April 28***

I read your letter by Jef Raskin, in Volume 2, Issue 10, about the D.C. Hayes modem board.

Randy Suess and I now have one on-line 24 hours a day in a computerlized call-in bulletin board system. We are quite pleased with it, and wanted to pass on our experiences, and also invite your readers to try it first hand. But first, some background on the system.

On January 16, 1978, I thought up the idea of a hobby computer-based bulletin board system, as a source of material for our club newsletter. I talked over the idea with Randy Suess, and we decided to go ahead and do it. I would do the software, and he would do the hardware.

We based it on a 24K 8080 system, using the Tarbell disk controller, and an Innovex floppy drive. The D.C. Hayes board, going thru a DAA, provided the phone link.

Within 30 days, we had the system designed, built, programmed, and tested. Shows what can be done if you really put your mind to it.

It has undergone very minor changes since going on the air in mid-February. A few resistor values were changed on the DC Hayes board, and a higher precision op amp was substituted, all based on work by Randy, and suggestions by Dennis Hayes.

We are now quite happy with the system. It is regularly called from California, Georgia, Ohio, as well as lots of local calls. It performs well with the possible exception of having trouble when the modem being used does not put out a strong signal. Very few people have had trouble receiving the signal we send out.

We are currently using a revision 3 modem board, and expect to be trying out a revision 4 board shortly (should be in place by the time this letter is printed).

We invite your readers to give it a try. We will take an informal poll of success or lack of it, and will report back to you.

Here is how to access the system: Any ASCII terminal, 110 or 300 baud, should work. There is no "protocol" associated with the system. It uses 2 stop bits at 110 baud, 1 at 300. It echoes everything typed, character by character, but this may be changed after signing on. For the time being, no password is required to access the system. We encourage its use, but may have to go to a password system if it gets abused.

Dial the system (312) 528-7141, and when you hear the answer modem tone, connect your modem. Then press return several times until the system responds. (C/R is used as the speed determining character). You will be prompted from then on.

NOTE: People who are successful will of course be known to us, but people who are unsuccessful won't. That's not right, as it gives a falsely biased view of the system's success. I therefore ask people who are unsuccessful in communicating to call me, Ward, at (312) 849-6279. I have a recorder on the phone in case I am not home. I would appreciate the callers having the following information ready to give in case the recorder answers the phone: Where they tried calling the system from; What type of modem was being used; What type of terminal; What speed; Any comments they think might help. I don't have the budget to return all the calls, but will try to handle those of interest and particularly those who might have useful suggestions.

Both Dennis Hayes and Dave Caulkins (PCNET) have received copies of the software, as they are very serious about setting up their own systems. We encourage other clubs to set up similar systems. The software is copyrighted, but something should be able to be worked out. The copyright is just to maintain control, not to limit its non-commercial use. A more detailed article on the system will be coming out in a Fall *BYTE*.

Happy Computing,
Ward Christensen

688 E. 154th St.
Dolton, IL 60419

MORE DOPE ON NIBL

Dear Dr. Dobb: *Received: 78 April 3*

Mike Fern's "nips at NIBL" in *DDJ* No. 22 contained several errors which deserve correction. The mysterious "JS" instruction he refers to is a pseudo-instruction recognized by the SC/MP cross assembler, and is documented in the SC/MP Assembly Language Programming Manual, publication number 420009C, January 1977. The JS instruction generates a sequence of five instructions occupying seven bytes of code. The actual sequence generated is: LDI, XPAH, LDI, XPAL, XPPC; the effect is to perform a long jump or subroutine call. In the NIBL source file, the assembler was directed (at line 2) to list no more than the first 2 bytes produced by any directive capable of generating more than 2 bytes of code, such as the JS instruction or ASCII directive. Thus, there are exactly five missing bytes of code in the listing at each occurrence of a JS, not 4 or -1 (!) as Mr. Fern claims. I suspect his hexadecimal arithmetic was a little rusty when he make those calculations.

By the way, SC/MP users will be pleased to note that the missing code appears in the load module produced by the assembler, and that the version of NIBL published in *Dr. Dobb's* actually runs, bugs and all.

The bug in NIBL (actually a design error) regarding the definitions of "true" and "false" has already been beaten pretty well into the ground by Jef Raskin. There are two possible fixes for the problem:

1. Change the definition of "true" in logical expressions to -1, rather than 1.
2. Change the code for the IF and UNTIL statements so that only the least significant bit of the logical expression is checked for 0 (false) or 1 (true).

At the time Jef reported the bug to me, NIBL had already been released and was being made into PROMs in great quantities for shipment (another way of saying the program was cast in concrete). If you don't like the way NIBL works, don't panic—change the rotten interpreter. I would implement fix #2 above by changing the OR instructions at addresses 5DB and 932 to ANI 1 (referring to the listing in *Dr. Dobb's* —the released version of NIBL will have slightly different addresses).

NIBL was not designed to sway followers of any particular Comp. Sci. guru (such as Dijkstra) away from their discipline of programming. It was built as a tool for helping a few hardware engineers debug their applications involving SC/MP, not as a language for writing business software or compilers. Admittedly, it is an atrocious language by most standards, but I would prefer that people compare it with languages in its class, namely the various dialects of Tiny BASIC. In all modesty, NIBL doesn't seem too bad by these lowered standards, aside from the fact that the interpreter is twice as large and 50 times as slow as Lichen Wang's excellent TB implementation (a result of having to use a hopelessly weak, slow processor).

I can't seem to find two people who will agree on some of the non-standard features in NIBL. Raskin hates the "@" notation for memory access, while Albrecht likes it; yet Raskin likes the DO/UNTIL structure and Albrecht condemns it as completely contrary to the spirit of BASIC. Personally, I find most dialects of BASIC (including NIBL) to be distasteful for writing large programs (i.e., programs longer than 10 lines).

NIBL should have remained in-house at National, but the marketing people (to whom all us peon programmers are slaves, more or less) saw it as a good gimmick to promote sales of SC/MP chips. Thus, I was pressured to publish both the listing and a quickie write-up full of errors before NIBL was really ready to be loosed upon the world. Judging by the generally unfavorable reaction from readers of *DDJ* this seems to have been a mistake.

Sincerely,
Mark Alexander

149 Jackson Ave. #200
Redwood City, CA 94061

'NUFF SAID

To All Multi-Micro Media Customers: *Received: 78 May 15*

Thank you for your order. It is most appreciated.

The purpose of this letter is two-fold: to acknowledge your order and to explain when, if ever, you can expect to receive it.

At present we have not received sufficient orders for some of the products to make it economically feasible to produce them at this time. The schedule below is our best guess as to when the order level will warrant producing the most *minimal* quantities for shipping.

PRODUCT	FORECAST	PROBABLE SHIPPING DATE
H8-BG1	Unclassifiable	Unknown
H8-BG2	Unclassifiable	Unknown
H8-MG1	Unclassifiable	Unknown
H8-MG2	Unclassifiable	Unknown
H8-MG3	Unclassifiable	Unknown
H8-MG4	Poor	July 1
H8-CODE	Unclassifiable	Unknown
H8-KING	Unclassifiable	Unknown
H8-DISM	Good	June 15
H8-EDUC	Good	June 1
R8-BG1	Excellent	May 1
R8-BG2	Very Good	May 15

As of May 1, 1978, we intend to re-evaluate the status of *ALL* orders for *ALL* products. At that time, any product whose forecast is still "Unclassifiable" will be cancelled and the orders for that product will be returned.

In the meantime, anyone who wishes to cancel an order is invited to to so. We will return your original order document and if payment was rendered by check, we will return your check. In the case of money orders made payable to Multi-Micro Media Corp., we will write a refund check for the same amount back to you.

Unless advised to the contrary, products will be shipped when produced and for credit card customers, you will be billed one at a time. In the case of orders for a combination of produced as well as cancelled products paid by check or money order, an appropriate refund will accompany the first product shipped.

Again, thank you for your order. We, like you, hope that interest will increase in our product line to make it possible to bring to you what we feel are good products. Your patience during this trial-time is appreciated very much.

Sincerely,
MULTI-MICRO MEDIA CORPORATION
Operations Department
(303) 425-5221

P.O. Box 1025
Arvada, CO 80001

SAM76 Language Update

Source Code Availability

The current position with respect to availability of the source code is very simple: it is available in machine readable form under the following conditions:

1. - A relatively complex test must be passed with flying honors to demonstrate the requestor's understanding of both the language philosophy and pragmatics; one of the ten test problems might be:

"Write a procedure named "FETCH" that precisely and completely mimics the "FT" function".

2. - The requestor must demonstrate skill and knowledge with respect to assembly language programming as it relates to the version of the source code desired; if the source code is desired as a model for implementation in a new machine then skill in the target machine must be demonstrated - team work is encouraged in this instance.

3. - The requestor must be prepared to agree to contribute free time towards the advancement of the language; this can be in a number of ways including improving the efficiency of existing source code, publishing algorithms and examples, giving talks to school and other educational groups &c.

4. - The requestor must agree to make available for publication at no charge any programs written either for other machines, or as additions to the source version requested. This does not, however, include any application programs, normally referred to as scripts or procedures written in the SAM76 language itself.

The foregoing requirements are an attempt at preserving the current integrity and portability of the language until enough implementations have been made that a "DE FACTO" standard is established; the benefits to users of such a position should be obvious.

I/O Vector Patch

Through an oversight all of the I/O calls were not made through the vector table created at ^h400 in early versions of the object code distributed at San Jose; this anomaly was brought to our attention the second day of the WEST COAST FAIRE, and the changes listed below will correct the problem for users with other than "ZAPPLE" monitors:

Version	location	from	to
Z80	80C6 & 8116	CD 15 F0	CD 21 04
	811F & 81CD	CD 18 F0	CD 24 04
8080	80E7 & 813B	CD 15 F0	CD 21 04
	8144 & 81F7	CD 18 F0	CD 24 04

Location of object code

The normal location for the interpreter object code was assembled to start at ^h8000; this was a compromise selection due to the desire to have all of the user work space in one continuous area starting at 0 for the Z80, and ^h400 for the 8080; the idea was to permit saving the user work space as an executable load module that would be portable from machine to machine, regardless of the location of the object code itself (be it below or above ^h8000).

Since then it has become painfully obvious that such hope could not be realized with regard to the POLYMORPHIC and HEATHKIT systems; consequently changes have been made in the object code to normalize certain pointers at the time the "EX" Exit function is executed, and restores these same pointers as appropriate to the machine in which the saved user work space might be reloaded.

version <0022>

Active and neutral execution of the "@N" function reveals the creation date and version number respectively; versions <0022> with a date equal to or greater than 14CC have been modified as required to permit portability of the actual work space. The creation date code may be decoded using the following procedure:

```
< date>
%ad,1,%su,%xd,%@@n//,%mu,31,%di,%xd,%@@n//,31////
[1]%ad,1,%su,%di,%xd,%@@n//,31/,
%mu,%di,%di,%xd,%@@n//,31/,12/,12///
[1]%ad,1964,%di,%di,%xd,%@@n//,31/,12//
```

Note that scripts output with the "OT" output text function, are not in question and that the new version is only required if the user desires to move his work space to machines where that space does not start at ^h400.

New function "XQS"

This function was added to delineate the limits of the user work space; it may be used at any time, but generally prior to executing the "EX" exit function; formal definition follows:

```
272  |xqs,s0|                    X Query Save
```

The value of this function are two numbers, each preceded by the string "s0" representing in the normal number base of the machine the starting and ending adresses of the user work space that should be saved.

Normally the "cws" change work space function would be first executed to reduce the work space to a minimum size, then the "xqs" function executed to ascertain the limits of the required save, prior to doing an "exit".

failure of "XI" in 8080 code

Version <0022> 14CC corrects the non functioning of the "XI" experimental input in the 8080 versions of the SAM76 language processor. The following shows the changes that can be made in early versions to correct this problem:

Early version coding:

```
XI:     MVI     A,KINP
        STA     CHA1
        MVI     A,KRET
        STA     CHA1+2
        CALL    MSXX
        STA     CHA1+1
        CALL    CHA1
        JMP     MSXRB
```

Corrected coding:

```
XI:     MVI     A,KINP
        STA     CHA1
        CALL    MSXX
        MOV     L,A
        MVI     H,KRET
        SHLD    CHA1+1
        CALL    CHA1
        MOV     C,A
        JMP     MSXRB
        NOP
```

CHA1= 04DE | KRET= C9 | KINP= DB
XI, MSXX, and MSXRB locations may vary between versions.

"CFC" superfluous argument

It was pointed out that the "cfc" change fill character funtion did not function as defined - requiring an extra argument; this may be readily corrected by replacing with NOP the call to the subroutine that moves the argument pointer in the expression being evaluated:

```
CFC:    MVI     A,NULCT-TIR     ; 3E 1E
        CALL    STORNO
        IJSAS   ; replace in z80 with one NOP, 8080 3 NOPs
        DCX     H
        CALL    MSR9
        MOV     M,A
        IJRQ
```

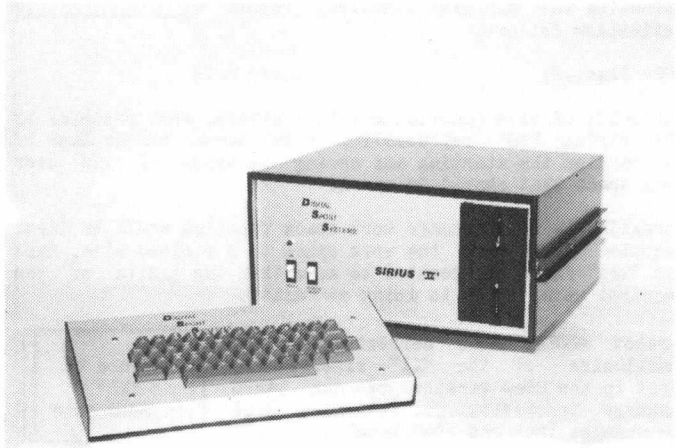
In the Z80 version IJSAS is an RST2, in the 8080 version IJSAS is assembled as a CALL to MSAS; the locations of CFC, STORNO, MSAS, and MSR9 may vary between versions.

SAM76 language afficionadoes might like to receive advance copies of material scheduled for publication. At this time it is suggested that those people wishing to receive preprints may send to SAM76 a self addressed envelope containing one dollar; this dollar will be replaced with preprints as available and the envelope mailed - then to continue if so desired %PREPRINTS/.

<PREPRINTS>

An object code version tailored to the POLYMORPHIC should be available by the time this is published. The price is $6.00 postpaid for this as well as for version <0022>. The 190 page textbook is priced at $12, and a system loader information booklet is $2; all of these are postpaid. Specify loose-leaf or bound preference for the text book.

SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534



THIS IS SIRIUS

News Release *Received: 78 April 13*

Sirius II comes complete, using a unique combination of two micro processors, the Mostek Z80 (for main computations), and a Fairchild 3870, that handles all keyboard and TV interface overhead. Also included are 32,768 characters of read/write memory, RS-232 interface for I/O, 8,192 character PROM board with 1,000 character monitor supplied, mini-floppy disc drive, a 64 key keyboard with alpha/numeric and graphic capabilities, and TV interface. Other features included are a full disc operating basic interpreter, monitor and software programs (ranging from home management, personal finance, educational learning programs, process control, to games and more games). The Sirius II is also capable of running the full complement of business software that its big brother (Sirius IV) runs.

Presently available hardware includes additional RS-232 I/O, additional parallel I/O, add-on 16,384 BYTE RAM cards, A/D-D/A converters, and additional mini-floppy disc drives.

The Sirius II is manufactured by Digital Sport Systems, and has a suggested retail price of $1,850. Contact Digital Sport Systems, 7th and Elm Streets, West Liberty, Iowa 52776; (319) 627-4211.

A BELL FOR YOUR SWTPC TERMINAL

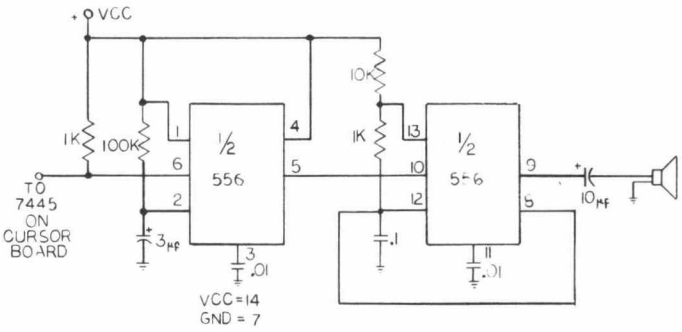
Dear Sir: *Received: 78 Feb 5*

Enclosed is a circuit for adding a "bell" function to a SWTPC CT-1024 terminal. The circuit consists of a 556 (dual 555) timer IC, four resistors, five capacitors, and a speaker. I built the entire circuit on the 14-pin Wire-wrap socket that the 556 is plugged into. The bell is triggered from an unused output pin of the 7445 decoder on the computer controlled cursor board in the CT-1024.

The left portion of the circuit is a one-shot with a duration of about ½ second. The right portion is an astable oscillator with a frequency of about 1Khz. The output of the one-shot is connected to the reset of the astable. When the one-shot is triggered by a low-going pulse from the 7445 decoder on the cursor board, its output goes high for ½ second. This high pulse allows the astable to oscillate at 1Khz for ½ second. The 1K resistor at the trigger for the one-shot is required to pull-up the output of the open-collector 7445. The astable is coupled to the speaker through a 10 mfd. capacitor. The circuit I built plugs in "piggy-back" onto another 14-pin IC (with Vcc=14, Gnd=7). The cost is less than $4.00.

Sincerely,
Stuart Brown

128½ Taber St.
Ft. Wayne, IN 46803



TIME-SHARE FOR NORTH STAR

News Release *Received: 78 April 6*

A Time-Share Disk BASIC System is now available for users of the North Star Floppy Disk System. Designed to operate with either 8080 or Z-80 processors, NORTHSHARE™ provides up to four independent users with selectable memory partitions and buffered terminal outputs.

Minimum memory requirements for operation are 24K bytes. There are no special hardware requirements outside of additional terminals and I/O ports to support the multiple users.

System includes one Diskette and Release 3 North Star Basic and DOS with NORTHSHARE™ Supervisor and Documentation Package. Price is $48. Byte Shop of Westminster, 14300 Beach Blvd., Westminster, CA 92683; (714) 894-9131.

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 27 | August 1978 | Volume 3, Issue 7

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

A Compact 8080 Disassembler & Dumper *Richard Greenlaw* 4
How it works and how to use it

A 1K Utilities Package for the Z-80 *Les Hancock* 22
A companion to the 1K Debugger published by Dr. Dobb's

A Shorter Printer Driver for PolyMorphic's BASIC *Richard B. Emerson* 27
With an explanation of circular buffers

Fast Cassette Interface for the 6502 *Carl Moser* 31

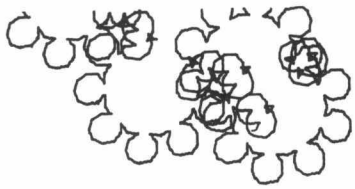
A BASIC Renumbering Program *John B. Trenholme* 34
Adaptable to various versions of BASIC

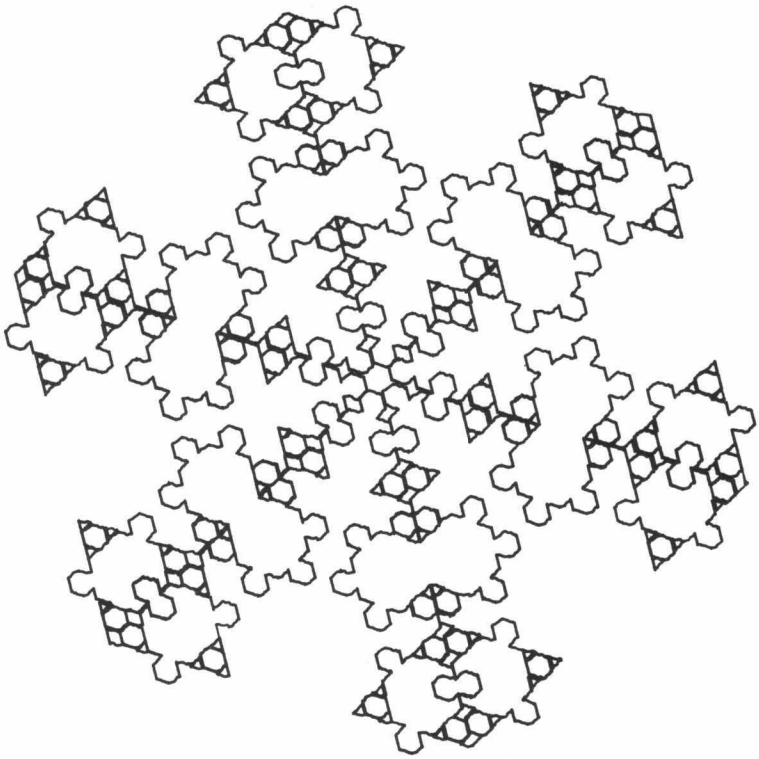
Custom File Routines for ALS-8 *Larry A. Marek* 42

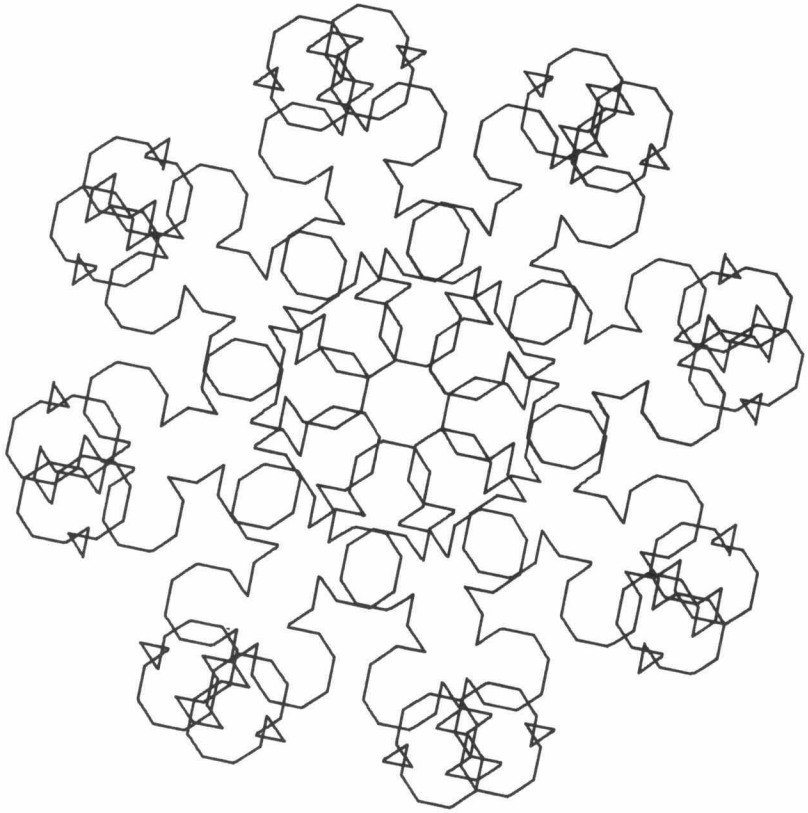
Common PILOT: Progress Report *Larry Kheriaty and George Gerhold* 46

HARDWARE

The Intel 8086 *Mike Gabrielson* 19
A look at Intel's new 16-bit microprocessor







© David Dameron 1979

Unix Dialogue Continues

Dear Dr. Dobb's: *Received: 78 June 28*

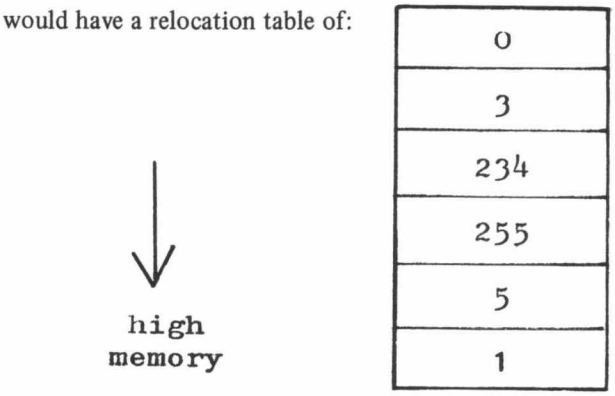
This is a follow-up letter to the UNIX™ letters that you have already published. As the chief initiators of the CHAOS project, we have faced many of the issues that Mr. duPont and Mr. Howell discuss. Specifically:

1. UNIX typically interrupts upon every input/output signal. For example, whenever a user types a character upon a terminal, an interrupt occurs. The system then stores the character in a buffer and, later, the user's program calls a system subroutine to retrieve the character from the buffer. This gives the "lookahead" feature. There is, however, considerable overhead in buffering characters in this fashion. Ignoring the memory required for the actual buffers, in a modular system such as UNIX or CHAOS, several modules must be invoked upon the interrupt. These modules must determine the source of the interrupt, locate the buffer associated with the device, read the character and place it into the buffer, and finally determine to which process control should be returned. On reasonably fast computers, such as the PDP 11's upon which UNIX runs, these operations can be handled quickly. On the 8080, where searches are slow, we have found that full character buffering entails a substantial percentage of processor time. Thus, in current versions of CHAOS, only disks and the real-time clock cause interrupts, and so eight levels of interrupt are sufficient. We do use a one-character buffer for input devices, as this permits the system to check for ⟨break signals⟩ for computer-bound processes.
2. We know of two reasonable techniques for handling relocation. One method entails a bit vector with one bit per byte of program. A bit is set in the bit vector if the address at the corresponding location must be relocated. Obviously, this technique will make a program 1/8 larger on a byte machine.

   In CHAOS, we use a somewhat similar technique. At the end of each program is a "relocation table" which contains the 'distances' between those addresses in the program that must be relocated. These distances are stored in a single byte if possible—when a distance is greater than 254 decimal, a 255 is placed in the byte and another two bytes are used to give a 16 bit distance. A zero byte terminates the relocation table. Our assembler has an option to generate the table, and all addresses are put into the table except for those preceded by a # sign. The relocation table is actually stored backwards so that the system can easily find the start of the table. As a short example, the program segment:

```
START:  LXI SP,TOP
        LXI H, 0
        JMP START
        DS  1000   ;1000(10) BYTES OF SPACE
TOP:    LDA 65000
```

   would have a relocation table of:



   There are a few important points here. One is that addresses needing to be relocated usually occur far less frequently than one in eight bytes. Also, examining a bit vector can be slow, as each bit must be tested individually. Finally, once CHAOS has used the relocation table to patch the program, the main memory space occupied by the table is reclaimed.
3. In our current version of CHAOS, we allocate time on a simple priority basis. Essentially, every process at level n is given time after a single process at level n+1 has used its slice of time. Processes that serve devices requiring frequent but short segments of time (such as a disk-head controller) are put at low levels while processes requiring infrequent slices of time (such as a line printer queuer) are put at high levels. Also, a process may be suspended by the system.
4. The size of CHAOS is only about one fifth to one third the size of the language (BASIC, PASCAL/S, PASCAL) that the users are working in. A prime consideration in choosing a processor is therefore the availability of suitable languages to build into the system and the compilation/interpretation time and space required by the languages. The amount of disk activity depends to a great extent upon the level of the users. At our high school, students learning BASIC use very little disk time. UNIX and CHAOS supplement disk space with sharable core buffers. This has a major impact on disk overhead, especially when several users are trying to access a file, such as the directory containing the system commands. Also, sharable buffers make filters easier to implement.
6. We believe that many people are interested in UNIX-type systems and that even more will be as the word gets out. UNIX is written primarily in C (see J Madden's article on page 130 of the October 1977 issue of *Byte* and any microcomputer with a C compiler should be able to run at least part of the system. There have already been some notable successes in transporting software, such as the UCSD PASCAL and the PASCAL-P compiler.

Our own efforts are directed towards producing a document of CHAOS detailing our algorithms and data structures so that other users can develop yet more advanced systems without having to start with the basics. UNIX and CHAOS offer great flexibility and promote rapid program —and have already spurred us and our students to create word processors, interactive debuggers, and even compilers.

Let's keep this discussion going in *Dr. Dobb's* (anyone for TINY-UNIX?) . . . .

Sincerely,
Jeff Levinsky
Robert Haas

Clairemont Computer Group
1541 Loring
San Diego, CA 92109

UNIX is a registered trademark of Bell Labs.

A Compact 8080 Disassembler & Dumper

BY RICHARD GREENLAW
251 Colony Ct.
Gahanna, Ohio 43230

Many important programs, such as BASIC interpreters, assemblers, editors and some of the most sophisticated games, are sold without source code listings. Others were written directly in machine code. When you want to modify or repair them this program will help you find and understand the desired routines and can make a source file which can be edited and reassembled.

This disassembler is compact enough to fit in memory with your assembler/editor and the program to be modified, unlike disassemblers written in BASIC. It operates on a program in memory to produce disassembled source code or a hex and ASCII dump.

The disassembled source code can be displayed or printed or can be appended to a file in memory. Labels are formed from their values in the original program, thus carrying some useful information and requiring only two bytes each in the disassembler symbol table. They can, of course, take on other values when reassembled.

The optional symbol table building pass and the output pass are independently controlled by the user. Either or both may omit or reorder parts of a program; thus labels can be generated on instructions referenced only from outside the area outputted.

Actual runs are shown in Figures 1 and 2.

Something for Everyone

Any 8080 system can use the disassembler and dumper (Figure 3, 1165_{10} bytes) with the command interpreter provided (Figure 4, 288_{10} bytes).

Users of Processor Technology's SOLOS™ and CUTER™ monitors don't need the command interpreter. Instead, they can use the routine in Figure 5 (81_{10} bytes) to define the disassembler and dump commands in their monitor.

The basic disassembler subroutine, DISL, is small enough (723_{10} bytes) to form the basis of a tiny disassembler in your own monitor.

How to Use it

A simple one-pass disassembly is shown in Figure 1. The ">" is a prompt from the command interpreter or your monitor. Commands end by carriage return, not shown.

First we define an empty symbol somewhere in RAM, at D500 in this case. This uses two bytes for now.

>ST D500

Now you are ready to disassemble something. Labels will be generated on all instructions because the symbol table is empty. The following command requests disassembly over a range of addresses with output to your display or other output device according to your version of the output subroutine.

>DS D9F1 DA06

During this output the input subroutine (SINP) is called at the end of each output line to see if you want to pause or stop the output. The space bar will cause a pause until a second key is operated. A control-@ (or MODE on a SOL) will terminate the output and return to the command prompter. Otherwise, output continues without pause.

If the second address had been omitted or was zero, it would have been defaulted to FFFF, which would never stop to output since it can't be exceeded. In such cases use control-@ to stop the output.

The next command in Figure 1 requests a hex and ASCII memory dump over a range of addresses:

>HA DA60 DA79

The hex part pairs the bytes to make the line fit on a 64 character display. There is no other significance, but it is convenient when counting across the dump to a particular address. In the ASCII part, values from space (20H) to, but not including, delete (7FH) are sent to the output subroutine unmodified. Others are converted to periods. There is one exception: the underscore (5FH) is converted because it would be converted to a backspace by my system monitor. This assumes your output device handles lower case alphabetic characters or converts them to upper case as my TTY does.

The method of Figure 1 is fine for a quick search for input/output drivers, etc., but you can't see which instructions receive control from calls and jumps and this format could

Figure 1 - A Simple Disassembly with ASCII/hex Dump

```
>ST D500
>DS D9F1 DA06
HD9F1 LXI  SP,HDAF0
HD9F4 CALL HD9F5
HD9F7 MVI  B,3FH
HD9F9 CALL HD9FB
HD9FC LXI  H,HDABF
HD9FF CALL HD9F8
HDA02 JZ   HD9FF
HDA05 MOV  M,A
HDA06 CPI  40H

>HA DA60 DA79
HDA60 F953 5AD3 D94B 41D6 D942 53D9 D924 53DC .SI..HA..BS..DS.
HDA70 D944 46DF D958 5BEF D94A 20DA D95F 21D0 .DF..X[...*... !.
```

overflow an assembler's symbol table if a fairly large program were reassembled.

To suppress unused labels, a symbol table is required to tell the disassembler which instructions are referenced. Actually, this table contains only the addresses of such instructions.

Figure 2 shows a two-pass disassembly. The symbol table is initialized as before by:

>ST D500

Then it is built by:

>BS D9D0 DABE

This runs the disassembler through the specified range without producing source code output. Instead it appends the last two bytes of each three byte instruction to the symbol table, unless they are already in the table. This assumes the value is an address. If it is a constant it could cause an unnecessary label to be generated on some instruction in the second pass.

```
Figure 2 - Two-Pass Disassembly

>ST D500
>BS D9D0 DABE
>DS D9E1
HD9E1 LXI  SP,HDABE
      CALL HD9E5
      MVI  B,3EH
      CALL HD9EB
      LXI  H,HDABE
HD9EE CALL HD9EB
      JZ   HD9EE
      MOV  M,A
      CPI  0DH
      JZ   HDA33
      MOV  B,A
      CPI  7EH
      JNZ  HDA1E
      LXI  D,HDABE
      CALL HD9E2
      JNC  HD9E1
      DCX  H
>XX
```

Note that the symbol table should represent all programs which could reference a label in the routines you intend to disassemble in pass two. If the range of addresses for the BS command is not continuous or contains data tables, simply use BS on each range separately. (Only the ST command empties the symbol table.) If you don't know where data tables are just include everything in the BS range. You may get some false symbol table entries and you could misinterpret a few instructions if a data area doesn't come out even when interpreted as instructions, but it's good enough for a first try.

Note that Figure 2 used the control-@ to stop the output and then shows the XX command exiting from the command interpreter to the system monitor.

If you want to build an assembler source file in memory instead of displaying the output, just use the DF command instead of DS. You must specify an end address since there's no visible display to interrupt. Each disassembled line is appended to the end of the file, so several DF ranges can be used.

Source files created by this disassembler exist entirely in RAM and are useable by several popular assemblers, such as Processor Technology's ALS-8. All line numbers are 0000, so ALS-8's RNUM command is needed before using a line number based editor. It renumbers the whole file without regard to the original numbers. The current end of file pointer is assumed to be in RAM at EOFP. This symbol is defined correctly for ALS-8. This disassembler doesn't use the beginning of file pointer and doesn't set the highest line number field in the file control block. RNUM does that.

Processor Technology's Software Package No. 1 uses the same EOFP address and file structure, but doesn't have the RNUM command unless you have patched it in. Some other assemblers are based on this package, or on an ancestor of it.

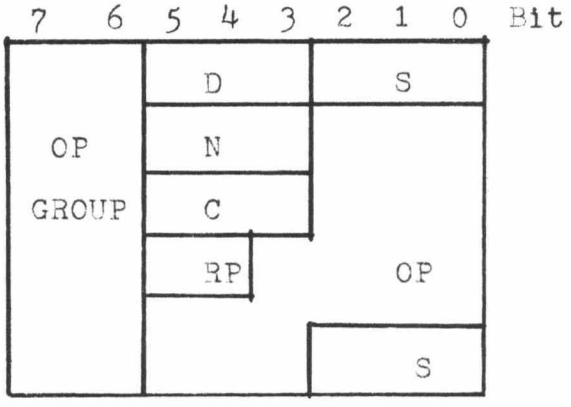
Operands of three byte instructions are disassembled as labels consisting of the letter H followed by the four hex digits of the value. If it is really a constant you must edit it before reassembly. Instructions referenced only by data tables or programs not represented in the symbol table may need labels not generated, and must be edited. Labels pointing at RAM areas and data tables must be defined by EQUates or other assembler pseudo-ops.

Operands of two byte instructions are disassembled as hex constants beginning with a decimal numeral and followed by the letter H, as required by the above assemblers. Any address-dependent values, such as one byte of an address, must be edited before reassembly.

How it Works

The functions of the command routines have already been described and they and the command interpreter are relatively straightforward, so this discussion and the flowcharts are limited to the process of disassembly itself.

The basic subroutine to disassemble one instruction begins at line 2435 in Figure 3. It can be entered at DISL to produce a label or at DISNL to omit the label. The HL registers point to the first byte of the instruction to be disassembled and the BC registers point to a RAM area to receive the disassembled source line followed by a carriage return byte. It advances these registers and if it is a three byte instruction the carry flag is set and the DE registers have the value of the last two bytes. One and two byte instructions reset the carry flag. No other registers are defined at exit from this subroutine. No other RAM is used except the stack.

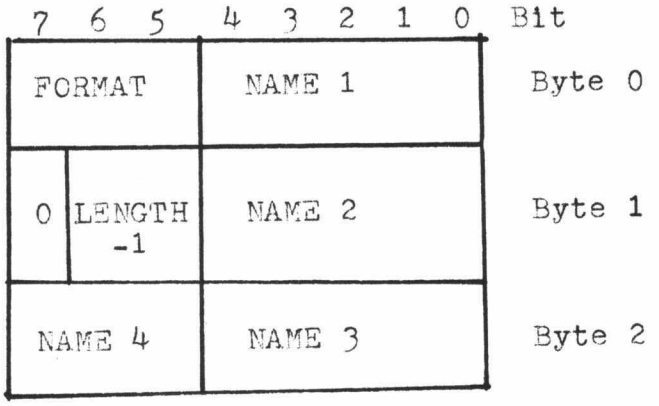


D Destination Register (A, B, C, D, E, H, L, M)
S Source Register (A, B, C, D, E, H, L, M)
N Restart number (0-7)
C Condition (NZ, Z, NC, C, PO, PE, P or M)
RP Register Pair (B, D, H or PSW/SP)
OP Group and Op determine the instruction name

Figure 6. Operation Byte Format for 8080

The 8080 operation (instruction) byte format is shown in Fig. 6. OP GROUP and OP represent the bits which are not dedicated to variable fields, such as register names, in a given instruction type. An instruction may or may not have a variable field or it may have both the D and S fields, as in MOV A, B. If an instruction has one or two additional bytes they simply contain a number which is interpreted by the disassembler as described above.

After generating or skipping the label field, disassembly continues by calling subroutine GTPTR which analyzes the operation byte of the instruction and produces a pointer to a 3-byte operation format block within the OPTAB table, as shown in Fig. 7. This block gives us, in tightly packed form, the name of the instruction, its length and its format. The



FORMAT CODES	NAME 1-3	Letters of instruction name coded as ASCII minus 'A'-1.
0 OP only		
1 RP-PSW		
2 RP-SP	NAME 4	Last letter of name coded randomly for table lookup.
3 S		
4 D		
5 DS	LENGTH	Number of bytes in instruction less one.
6 N		
7 C		

Figure 7. Operation Format Block Layout

DISL subroutine then converts the operation name to ASCII, using table lookup for the fourth letter. Conditional jumps, calls, and returns complete the name from a variable field.

Then the operand fields are generated from whatever variable fields are specified by the format code in the format block and from any additional bytes indicated by the length field of the format block. Small tables and simple algorithms are used in these conversions.

The only inner subroutine worthy of discussion is the GTPTR, at line 3650. This subroutine and the tables it uses make possible a compact disassembler by allowing OPTAB to have only 58 format blocks for the 256 operation byte values.

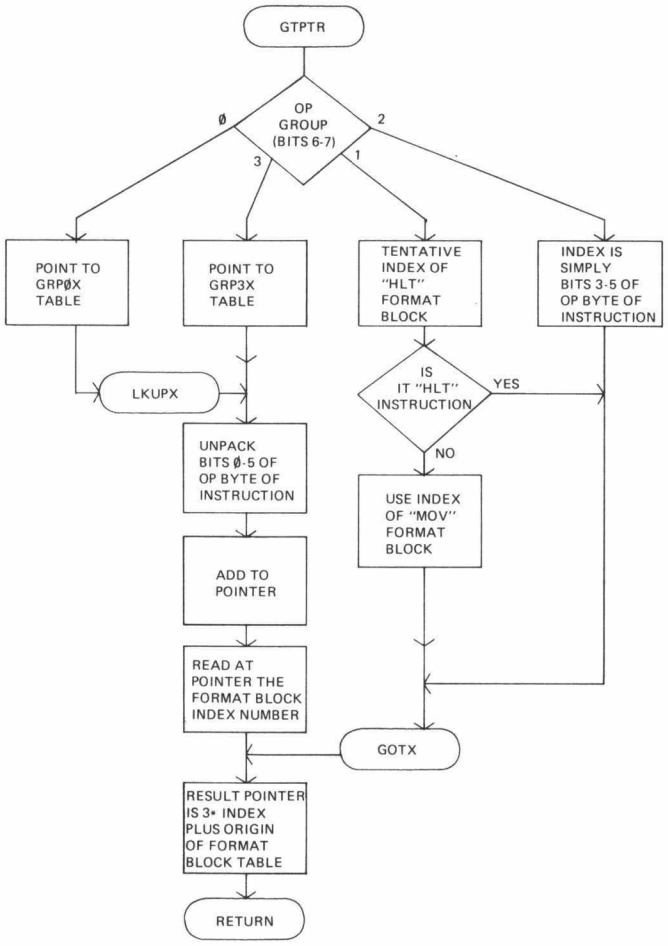
First, GTPTR determines which of four operation groups the instruction is in. This is simple because the group is simply the left two bits of the instruction, as shown in Fig. 6.

Group 0 consists of many instruction formats so the entire remaining six bits of the instruction are converted to an operation format block number by direct lookup in the GRP0X table.

Group 1 consists only of the MOV and HLT instructions so either format block is selected by simply checking for the HLT operation value, which has no variable fields.

Group 2 instructions all have the same format, with variable field S. OPTAB is arranged so that the three remaining instruction bits are the format block number.

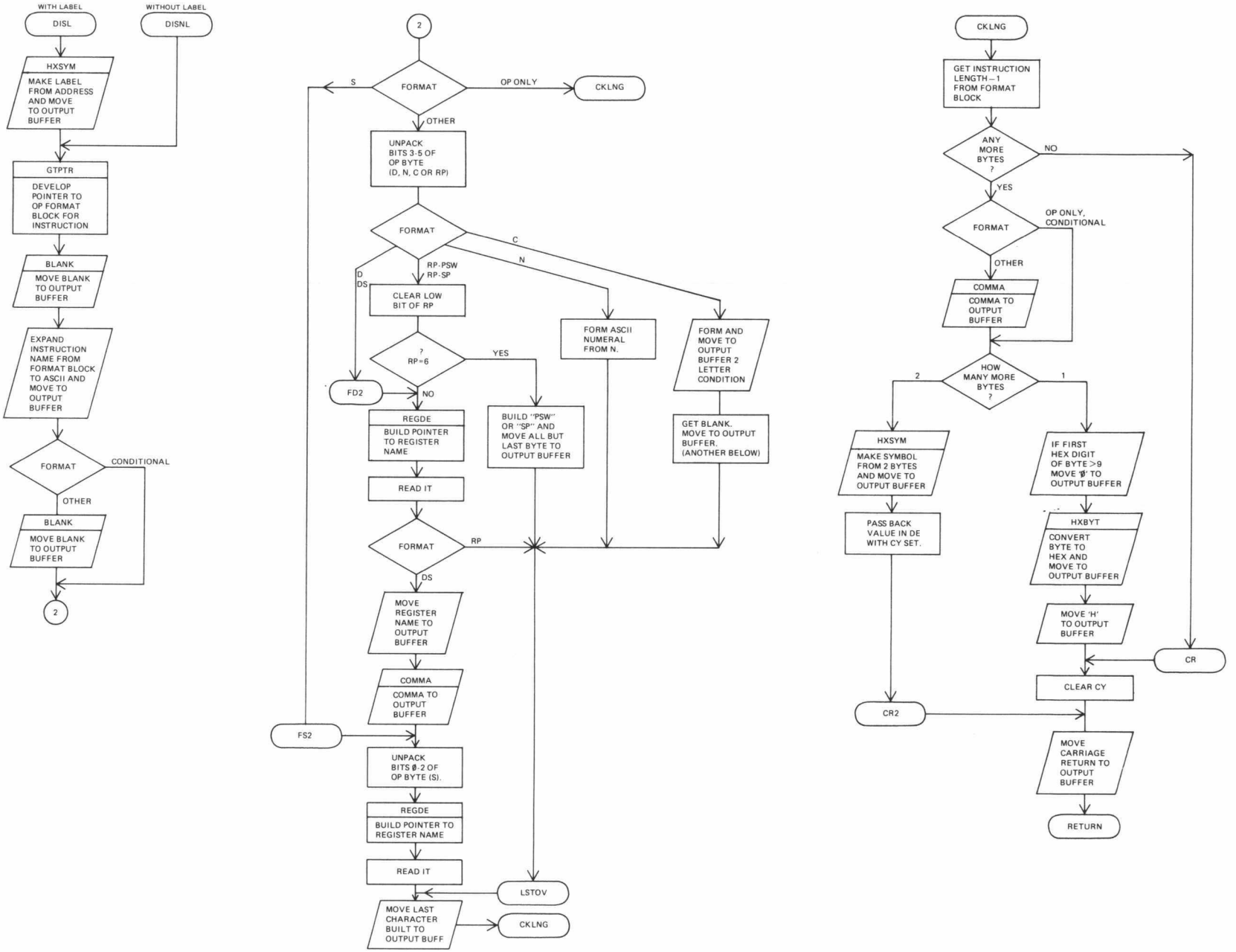
Group 3 is complex like group 0 and is treated the same using the GRP3X table.



SUBROUTINE TO BUILD POINTER TO FORMAT BLOCK

Interfaces with Your System

If you use SOLOS or CUTER you can use the command definition routine in Figure 5 with the program (Figure 3) and omit the command interpreter (Figure 4). Verify the addresses in the PSCN and SCONV jump vectors (lines 1150-1155 of Figure 3) and all references to CUTAB in Figure 5 with your monitor listing and correct them if necessary. Then simply execute the command definition routine from your monitor. It will return to the monitor and the disassembler and hex/ASCII dump commands may now be entered just like other monitor commands.



SUBROUTINE TO DISSASSEMBLE ONE INSTRUCTION

If you don't have SOLOS or CUTER, use the command interpreter (Figure 4) and omit the routine in Figure 5. Link them together and to your input and output routines as follows:

1. Check that the jump vectors in lines 90-150 of Figure 4 point to the corresponding symbols in Figure 3.
2. Point the RETRN vector in line 200 of Figure 4 back to your system monitor or, if you don't use a monitor, to the start of the command interpreter.
3. Point the RETRN vector in line 1135 of Figure 3 to the start of the command interpreter.
4. Point the SINP and SOUT sectors in *both* Figures 3 and 4 to input and output routines which do the following in your system:

   SINP: Check for keyboard input. If present, return it in the A register with the zero flag not set. If not present, set the zero flag and return (don't wait for input). Always preserve all registers except A and PSW.

   SOUT: Output the character in the B register to your display, TTY, or whatever. This is used by the command interpreter as well as the DS command. Wait until it can be done, then return with all registers except A and PSW unchanged. At the end of a line this will be called four times: for carriage return, line feed and two nulls (00H). This routine is not responsible for pause control or control-@ detection from the keyboard. The CRLF routine in Figure 3 does all that.
5. Point the PSCN and SCONV vectors in lines 1150-1155 of Figure 3 to the corresponding routines in Figure 4.
6. Give control to the start of the command interpreter. You are now ready to go. Try the examples in Figures 1 and 2.

Getting It Into Your System

The program start addresses were chosen to use RAM usually available in systems configured for the assembler systems described above. If you want to put them elsewhere I recommend getting or typing a source file so you can assemble it anywhere. Unfortunately, the source file that produced Figure 3 occupied 20K of RAM! You can divide it into three parts at lines 2405 and 3925 with a minimum of broken linkages. The first two parts would occupy more like 6K without comments and unnecessary blanks. The third part consists entirely of constants in tables and is not address dependent. The machine code relocater in *BYTE*, July, 1977, could also be useful.

If you can read my tapes and don't like to type I'll supply copies and special editions for a copying fee.

My system is a SOL-20 with 32K of RAM. My TTY does not have paper tape. I have a MICROPOLIS™ METAFLOPPY™ disk system but don't have it doing assembly language yet.

I can offer K.C. standard audio cassettes at 300 baud or at 1200 baud (uses lower tones). The recording format is compatible with SOLOS and CUTER monitors. I can also send the results of a LIST command to the tape without formatting or error (CRC) checks. The software to read this from an appropriate tape interface is trivial.

Prices: machine code of Figures 3 and 4 in normal 300 or 1200 baud format, $4; same at special address, $8; source file with comments (20K) in normal formats, $6; same in three parts, $8; unformatted LISTing at either speed, $10.

Specify speed and format. You'll get a cheap cassette unless you send one; they work for me. This isn't a business so there could be delays or rejected orders.

Figure 3

```
READY
ASSMX DAFA
DAFA                1000 *8000 DISASSEMBLER SYSTEM 3/23/78
DAFA                1005 *BY RICHARD GREENLAW
DAFA                1010 *   251 COLONY CT.
DAFA                1015 *   GAHANNA, OHIO 43230
DAFA                1020 *
DAFA                1025 *COMMAND DESCRIPTIONS:
DAFA                1030 *ADDR1-ADDR2 IS RANGE TO DISASSEMBLE
DAFA                1035 *ADDR2 DEFAULTS TO NO LIMIT. UNUSED LABELS
DAFA                1040 *ARE SUPPRESSED IF SYMBOL TABLE BUILT.
DAFA                1045 * ST ADDR  DEFINE # EMPTY SYMBOL TABLE
DAFA                1050 * RS ADDR1 [ADDR2] APPEND SYMBOLS TO TABLE
DAFA                1055 * DS ADDR1 [ADDR2] DISPLAY
DAFA                1060 * DF ADDR1 [ADDR2] APPEND TO FILE. (CALL
DAFA                1065 *   LINE NUMBERS=8000, NEEDS EQUF EQU.)
DAFA                1070 * HA ADDR1 [ADDR2] HEX AND ASCII DUMP
DAFA                1075 *
DAFA                1080 EQUF  0D807H  PTR TO END OF FILE (DF)
DAFA                1085 PSW   EQU     6
DAFA                1090 *
DAFA                1095 *THE DISASSEMBLER COMMANDS ARE DISPATCHED
DAFA                1100 *AND THE ADDRESSES ARE CONVERTED BY A
DAFA                1105 *SEPARATE MONITOR SUCH AS PROFESSOR
DAFA                1110 *TECHNOLOGY'S SOLOS(TM) OR CUTER(TM) OR BY
DAFA                1115 *THE SEPERATE COMMAND INTERPRETER PROVIDED.
DAFA                1120 *
DAFA                1125 *VECTORS TO COMMAND INTERPRETER AND I/O.
DAFA C3 8A CA       1130 *ADDRESSES SHOWN ARE FOR SOLOS(TM) MONITOR
DAFD C3 1F CB       1135 RETRN JMP   0CB1FH  RETURN TO MONITOR
DB00 C3 19 C9       1140 SINP  JMP   0C919H  INPUT A CHAR OR Z FLAG
DB03                1145 *
DB03                1150 *BUILD COMMON SYMBOL TABLE FOR USE
DB03                1155 *DURING ASSEMBLY OF OPTIONAL
DB03                1160 *COMMAND INTERPRETER.
DB03                1165 *
DB03 C3 19 C9       1170 SOUT  JMP   0C919H  OUTPUT CHAR FROM B
DB06 C3 3A C3       1175 SCONV JMP   0C33AH  GET ADDRESS 1 IN HL
DB09 C3 10 C3       1180 PSCN  JMP   0C310H  GET ADDRESS 2 IN HL
DB0C                1185 *
DB0C                1190 *
DB0C                1195 *
DB0C                1200 *
DB0C                1205 *
DB0C                1210 *
DB0C                1215 *
DB0C                1220 *COMMAND HA: HEX & ASCII DUMP
DB0C CD 9C DB       1225 HA    CALL  ADDRS HL->START, DE->END
DB0F CD 89 DB       1230       CALL  CRLF
DB12 01 39 DF       1235 HALP  LXI   B,-0C6H
DB15 D5             1240       PUSH  D
DB16 C5             1245       PUSH  B
DB17 CD 86 DD       1250       CALL  HXSYM
DB1A CD 72 DD       1255       CALL  BLANK
DB1D 16 08          1260       MVI   D,8       BYTE PAIRS/LINE
DB1F E5             1265       PUSH  H
DB20 7E             1270 HXLP  MOV   A,M
DB21 CD BF DD       1275       CALL  HXBYT
DB24 23             1280       INX   H
DB25 7E             1285       MOV   A,M
DB26 CD BF DD       1290       CALL  HXBYT
DB29 CD 72 DD       1295       CALL  BLANK
DB2C 23             1300       INX   H
DB2D 15             1305       DCR   D
DB2E C2 20 DB       1310       JNZ   HXLP
DB31 F1             1315       POP   H
DB32 16 10          1320       MVI   D,16      BYTES/LINE
DB34 7E             1325 AIP   MOV   A,M
```

```
0B2B FE 7F          1330        CPI     7FH
0B2D F2 37 0B       1335        JP      FILL
0B30 FE 5F          1340        CPI     5FH
0B32 CA 37 0B       1345        JZ      FILL    IF BACKSPACE
0B35 FE 20          1350        CPI     20H
0B37 F2 39 0B       1355        JP      ASOUT
0B3A 3E 2E          1360 FILL   MVI     A,2EH   NON-PRINTING BECOMES PER
                                I
0B3C 02             1365 ASOUT  STAX    B
0B3D 03             1370        INX     B
0B3E 23             1375        INX     H
0B3F 15             1380        DCR     D
0B40 C2 27 0B       1385        JNZ     ALP
0B43 3E 0D          1390        MVI     A,0DH
0B45 02             1395        STAX    B
0B46 C1             1400        POP     B
0B47 CD 72 0B       1405        CALL    OUTLN
0B4A D1             1410        POP     D
0B4B CD 97 0B       1415        CALL    CHLDE
0B4E D2 05 0B       1420        JNC     HALP
0B51 C9             1425        RET
0B52                1430 *
0B52                1435 *COMMAND DS: DISASSEMBLE TO DISPLAY
0B52 CD 9C 0B       1440 DS     CALL    ADDRS   HL->START, DE->END
0B55 CD 82 0B       1445        CALL    CRLF
0B58 01 39 0E       1450 DSLP   LXI     B,OBUF  OUTPUT BUFFER
0B5B D5             1455        PUSH    D
0B5C C5             1460        PUSH    B
0B5D                1465 *DISASSEMBLE 1 INSTR W/ OR W/O LABEL
0B5D CD 4E 0C       1470        CALL    DISA1
0B60                1475 *HL->ADDR OF NEXT INSTR TO DISS.
0B60                1480 *OUT BUFF HAS LABEL, OP, VAR AND CR
0B60 C1             1485        POP     B       OUT BUFF START
0B61                1490 *IF NO LABEL, SUBSTITUTE BLANKS
0B61 0A             1495        LDAX    B       C(COL 1)
0B62 C2 67 0B       1500        JNZ     DISPL
0B65 01 34 0E       1505        LXI     B,OBENL START OUTPUT AT BLANKS

0B68 CD 72 0B       1510 DISPL  CALL    OUTLN   OUTPUT LINE
0B6B D1             1515        POP     D
0B6C CD 97 0B       1520        CALL    CHLDE   COMPARE HL:DE
0B6F D2 58 0B       1525        JNC     DSLP
0B72 C9             1530        RET
0B73                1535 *
0B73                1540 *SUBR TO OUTPUT THRU CR AND ADD LF.
0B73                1545 *BC->BUFF START
0B73                1550 *CHANGES A, PSW, BC
0B73 0A             1555 OUTLN  LDAX    B       CHAR TO OUTPUT
0B74 FE 0D          1560        CPI     0DH CR
0B76 CA 82 0B       1565        JZ      CRLF
0B79 C5             1570        PUSH    B
0B7A 47             1575        MOV     B,A
0B7B CD F6 0A       1580        CALL    SOUT    AFFECTS A, PSW
0B7E C1             1585        POP     B
0B7F 03             1590        INX     B
0B80 C3 73 0B       1595        JMP     OUTLN
0B83                1600 *
0B83 CD A9 0B       1605 CRLF   CALL    CKWT    PAUSE/EXIT CK (A/PSW)
0B86 06 0D          1610        MVI     B,0DH   CR
0B88 CD F6 0A       1615        CALL    SOUT
0B8B 06 0A          1620        MVI     B,0AH   LF
0B8D CD F6 0A       1625        CALL    SOUT
0B90 06 00          1630        MVI     B,0     NULL
0B92 CD F6 0A       1635        CALL    SOUT
0B95 C3 F6 0A       1640        JMP     SOUT    ANOTHER NULL, RETURN
0B98                1645 *
0B98                1650 *SUBR TO COMPARE HL:DE
0B98 7B             1655 CHLDE  MOV     A,E
0B99 95             1660        SUB     L
0B9A 7A             1665        MOV     A,D
0B9B 9C             1670        SBB     H
0B9C C9             1675        RET     .       CY SET IF HL>DE
0B9D                1680 *
0B9D                1685 *SUBR TO GET PARAMETERS FROM COMMAND
0B9D CD E9 0A       1690 ADDRS  CALL    SCONV   GET 1ST ADDR OR EXIT
0BA0 E5             1695        PUSH    H
0BA1 21 FF FF       1700        LXI     H,0FFFFH  DEFAULT
0BA4 CD EC 0A       1705        CALL    PSCN    GET 2ND OR DEFAULT
0BA7 D1             1710        POP     D
0BA8 EB             1715        XCHG
0BA9                1720 *HL=START, DE=END ADDRESS
0BA9 C9             1725        RET
0BAA                1730 *
0BAA                1735 *SUBR TO CONTROL PAUSE/ EXIT VIA KBD
0BAA CD F3 0A       1740 CKWT   CALL    SINP    GET CHAR IF ANY
0BAD C8             1745        RZ      .       IF NONE
0BAE E6 7F          1750        ANI     7FH
0BB0 CA F0 0A       1755        JZ      RETRN   IF MODE/CTL-@
0BB3 FE 20          1760        CPI     ' '
0BB5 C0             1765        RNZ     .       IGNORE OTHERS
0BB6                1770 *SPACE BAR=PAUSE
0BB6 CD F3 0A       1775 PAUSE  CALL    SINP
0BB9 CA B6 0B       1780        JZ      PAUSE
0BBC E6 7F          1785        ANI     7FH
0BBE CA F0 0A       1790        JZ      RETRN   IF MODE/CTL-@
0BC1 C9             1795        RET
0BC2                1800 *
0BC2                1805 *COMMAND BS: BUILD SYMBOL TABLE
0BC2 CD 9C 0B       1810 BS     CALL    ADDRS   HL=START, DE=END
0BC5 01 39 0E       1815 BSLP   LXI     B,OBUF  SAFE PLACE FOR OUTPUT
0BC8 D5             1820        PUSH    D
0BC9 CD 61 0C       1825        CALL    DISL    DISASSEMBLE 1 INSTR
0BCC                1830 *IF CY=1 DE=ADDRESS FROM 3 BYTE INSTR
0BCC D2 E1 0B       1835        JNC     BSECK   IF NOT 3 BYTE
0BCF E5             1840        PUSH    H       SAVE DIS. PC
0BD0 EB             1845        XCHG    .       HL=ADDR FROM INSTR
0BD1 CD E9 0B       1850        CALL    LKUPA
0BD4                1855 *CY SET IF MATCH. ELSE DE->EOF
0BD4 DA E0 0B       1860        JC      POPH    IF ALREADY IN TABLE
0BD7                1865 *PUT IN TABLE.
0BD7 EB             1870        XCHG
0BD8                1875 *HL->OLD EOF, DE->ADDR FROM INSTR
0BD8 73             1880        MOV     M,E
0BD9 23             1885        INX     H
0BDA 72             1890        MOV     M,D
0BDB 23             1895        INX     H
0BDC 36 FF          1900        MVI     M,0FFH  EOF
0BDE 23             1905        INX     H
0BDF 36 FF          1910        MVI     M,0FFH
0BE1 E1             1915 POPH   POP     H       DIS. PC
0BE2 D1             1920 BSECK  POP     D       DIS END ADDR
0BE3 CD 97 0B       1925        CALL    CHLDE
0BE6 D2 C5 0B       1930        JNC     BSLP    IF NOT DONE
0BE9 C9             1935        RET
0BEA                1940 *
0BEA                1945 LKUPA  EQU     $
0BEA                1950 *LOOKUP ADDRESS IN SYMBOL TABLE
0BEA                1955 *HL=ADDR TO MATCH.
0BEA                1960 *SET CY=1 IF MATCH OR DE->1ST BYTE OF EOF
0BEA EB             1965        XCHG
0BEB 2A 79 0E       1970        LHLD    SYMTB   A(SYM TAB)
0BEE EB             1975        XCHG
0BEF                1980 *DE->TABLE, HL->ADDR FROM INSTR
0BEF 1A             1985 CMPLP  LDAX    D
0BF0 FE FF          1990        CPI     0FFH    END TABLE MARK
0BF2 CA 05 0C       1995        JZ      CKEOT
0BF5 BD             2000 CMPR1  CMP     L
0BF6 13             2005        INX     D
0BF7 C2 01 0C       2010        JNZ     SKP2
0BFA 1A             2015        LDAX    D       2ND BYTE
0BFB 13             2020        INX     D
0BFC BC             2025        CMP     H
```

```
DBEC C2 FE DB          2035          JNZ     CMPLP   NEXT ENTRY
DBEF 37                2040          STC     .       INDICATE MATCH
DBF0 C9                2045          RET
DBF1 13                2050 SKP2     INX     D
DBF2 C3 FE DB          2055          JMP     CMPLP
DBF5                   2060 *
DBF5                   2065 CKEOT    EQU     $       CK FOR 2ND FF (EOF)
DBF5 13                2070          INX     D
DBF6 1A                2075          LDAX    D       2ND BYTE
DBF7 FE FF             2080          CPI     0FFH
DBF9 1B                2085          DCX     D
DBFA 1A                2090          LDAX    D       1ST BYTE AGAIN
DBFB C2 F4 DB          2095          JNZ     CMPRT
DBFE C9                2100          RET     .       CY=0 FOR EOT
DBFF                   2105 *
DBFF                   2110 *COMMAND ST: DEFINE SYMBOL TABLE
DBFF CD F9 DA          2115 ST       CALL    SCONV   HL FROM COMMAND OR ERROR

DC12 22 79 DF          2120          SHLD    SYMTB   DEFINE ORIGIN
DC15 3E FF             2125          MVI     A,0FFH  END OF TABLE MARK
DC17 77                2130          MOV     M,A
DC18 23                2135          INX     H
DC19 77                2140          MOV     M,A
DC1A C9                2145          RET
DC1B                   2150 *
DC1B                   2155 *COMMAND DF: DISASSMEBLE TO FILE
DC1B CD 9C DB          2160 DF       CALL    ADDRS
DC1E E5                2165 DFLP     PUSH    H
DC1F 2A 07 D0          2170          LHLD    EOFP    ->EOF MARK
DC22 44                2175          MOV     B,H
DC23 4D                2180          MOV     C,L
DC24 E1                2185          POP     H
DC25                   2190 *HL->NEXT INSTR, DE->STOP ADDR, BC->EOF
DC25 D5                2195          PUSH    D
DC26 C5                2200          PUSH    B
DC27 03                2205          INX     B
DC28 3E 30             2210          MVI     A,'0'   LINE NUMBER 0000
DC2A 02                2215          STAX    B
DC2B 03                2220          INX     B
DC2C 02                2225          STAX    B
DC2D 03                2230          INX     B
DC2E 02                2235          STAX    B
DC2F 03                2240          INX     B
DC30 02                2245          STAX    B
DC31 03                2250          INX     B
DC32 3E 20             2255          MVI     A,' '
DC34 02                2260          STAX    B       BYTE BEFORE LABEL FIELD

DC35 03                2265          INX     B
DC36                   2270 *DISASSEMBLE 1 INSTR TO END OF FILE
DC36 CD 4F DC          2275          CALL    DISA1
DC39 3E 01             2280          MVI     A,1     MARK
DC3B 02                2285          STAX    B
DC3C 78                2290          MOV     A,B     UPDATE EOF PTR
DC3D 32 08 D0          2295          STA     EOFP+1
DC40 79                2300          MOV     A,C
DC41 32 07 D0          2305          STA     EOFP
DC44 C1                2310          POP     B       START OF LINE
DC45 91                2315          SUB     C       LENGTH
DC46 02                2320          STAX    B
DC47 D1                2325          POP     D       STOP ADDR
DC48 CD 97 DB          2330          CALL    CHLDE
DC4B D2 1E DC          2335          JNC     DFLP IF NOT DONE
DC4E C9                2340          RET
DC4F                   2345 *
DC4F                   2350 *DISASSEMBLE 1 INSTRUCTION
DC4F                   2355 *BUILD LABEL IF IN TABLE OR TABLE EMPTY
DC4F                   2360 * BC->LABEL FIELD IN FILE, HL->INSTR.
DC4F CD F9 DB          2365 DISA1    CALL    LKUPA   IS ADDR IN TABLE
DC52 DA 61 DC          2370          JC      DIS1    IF SO INCLUDE LABEL
DC55 E5                2375          PUSH    H

DC56 2A 79 DF          2380          LHLD    SYMTB
DC59 EB                2385          XCHG
DC5A CD 97 DB          2390          CALL    CHLDE   CY IF HL>DE
DC5D E1                2395          POP     H
DC5E DA 64 DC          2400          JC      DISNL   IF NO LABEL
DC61                   2405 *
DC61                   2410 *SUBROUTINE TO DISSASEMBLE ONE INSTRUCTION
DC61                   2415 *SETS CY IF DE=ADDR FIELD OF 3 BYTE INSTR
DC61                   2420 *BC->OUTPUT BUFFER (ADVANCES)
DC61                   2425 *HL->INSTRUCTION (ADVANCES)
DC61                   2430 *
DC61                   2435 DISL     EQU     $       DISASSEMBLE INSTR W/ LAB
                                       F
DC61 CD B6 DD          2440          CALL    HXSYM   PACK HL AS SYMBOL
DC64                   2445 DISNL    EQU     $       DISASSEMBLE W/O LABEL
DC64 C5                2450          PUSH    B
DC65 CD A6 DD          2455          CALL    GTPTR   GET DE->OP FMT BLK
DC68 C1                2460          POP     B
DC69 CD 72 DD          2465          CALL    BLANK
DC6C                   2470 *BUILD OP FIELD
DC6C 1A                2475          LDAX    D       OPR(0)
DC6D E6 1F             2480          ANI     1FH     1ST LETTER CODE
DC6F C6 40             2485          ADI     'A'-1   MAKE ASCII
DC71 02                2490          STAX    B       TO OUT BUFFER
DC72 03                2495          INX     B
DC73 13                2500          INX     D
DC74 1A                2505          LDAX    D       OPR(1)
DC75 E6 1F             2510          ANI     1FH
DC77                   2515 *IF ZERO, OP DONE EXCEPT COND OF C,J,R
DC77 CA A0 DC          2520          JZ      BACK1
DC7A C6 40             2525          ADI     'A'-1
DC7C 02                2530          STAX    B
DC7D 03                2535          INX     B
DC7E 13                2540          INX     D
DC7F 1A                2545          LDAX    D       OPR(2)
DC80 E6 1F             2550          ANI     1FH
DC82 C2 87 DC          2555          JNZ     OPCH3
DC85 3E F0             2560          MVI     A,' '-'A'+1  FUDGE BLANK
DC87 C6 40             2565 OPCH3    ADI     'A'-1
DC89 02                2570          STAX    B
DC8A 03                2575          INX     B
DC8B 1A                2580          LDAX    D       OPR(2)
DC8C E6 E0             2585          ANI     0E0H    HI 3 BITS
DC8E 07                2590          RLC
DC8F 07                2595          RLC
DC90 07                2600          RLC
DC91                   2605 *LOOKUP 4TH OP LETTER
DC91 C5                2610          PUSH    B
DC92 01 E8 DD          2615          LXI     B,OL4TB  TABLE START
DC95 81                2620          ADD     C
DC96 4F                2625          MOV     C,A
DC97 3E 00             2630          MVI     A,0     W/O CHANGING CARRY
DC99 88                2635          ADC     B
DC9A 47                2640          MOV     B,A     BC->BYTE IN TABLE
DC9B 0A                2645          LDAX    B       4TH OP LETTER ASCII
DC9C C1                2650          POP     B       BC->OUT BUFF
DC9D 02                2655          STAX    B
DC9E 03                2660          INX     B
DC9F 1B                2665          DCX     D
DCA0 1B                2670 BACK1    DCX     D
DCA1                   2675 *BUILD OP BYTE DEPENDENT VARIABLE FIELDS
DCA1                   2680 *HL->OP, DE->OPR(0), BC->OUT BUFF
DCA1 1A                2685          LDAX    D
DCA2 E6 F0             2690          ANI     0F0H    FMT CODE IN 5-7
DCA4                   2695 *FORMAT CODES NAMED FOR VARIABLES
DCA4                   2700 *1=RP(SP),2=RP(PSW),3=S,4=D,5=DS,6=N,7=COND
DCA4 F5                2705          PUSH    PSW
DCA5 FE E0             2710          CPI     0E0H
DCA7 C4 72 DD          2715          CNZ     BLANK   IF NOT FMT 7 (COND)
DCAA F1                2720          POP     PSW
```

```
DCAB CA 2C DD    2725          JZ      CKLNG   IF NO FIELDS FROM OP
DCAE D5          2730          PUSH    D       DURING OP-VAR BUILD
DCAF FE 60       2735          CPI     60H     TYPE 3?
DCB1 F5          2740          PUSH    PSW
DCB2 CA 21 DD    2745          JZ      FS2
DCB5             2750 *REMAINING FMTS NEED BITS 3-5 OF OP BYTE
DCB5 7E          2755          MOV     A,M     OP BYTE
DCB6 0F          2760          RRC
DCB7 0F          2765          RRC
DCB8 0F          2770          RRC
DCB9 E6 07       2775          ANI     7
DCBB 57          2780          MOV     D,A
DCBC F1          2785          POP     PSW     RESTORE
DCBD F5          2790          PUSH    PSW
DCBE             2795 *A=FMT CODE, PSW SET,
DCBE             2800 *HL->OP BYTE, D=OP BITS 3-5, BC->OUT B
DCBE D2 E7 DC    2805          JNC     CKN     IF FMT>3
DCC1             2810 *RP FMTS
DCC1 7A          2815          MOV     A,D
DCC2 E6 06       2820          ANI     6       RP IS 2 BITS
DCC4 57          2825          MOV     D,A
DCC5 FE 06       2830          CPI     6
DCC7 C2 10 DD    2835          JNZ     FD2     IF D HAS REG CODE
DCCA             2840 *BUILD SP OR PSW
DCCA F1          2845          POP     PSW     A=FMT
DCCB F5          2850          PUSH    PSW
DCCC FE 40       2855          CPI     FRPP
DCCE CA DA DC    2860          JZ      RPPSW
DCD1 3E 53       2865          MVI     A,'S'
DCD3 02          2870          STAX    B
DCD4 03          2875          INX     B
DCD5 3E 50       2880          MVI     A,'P'
DCD7 C3 28 DD    2885          JMP     LSTOV   LAST OP VAR
DCDA 3E 50       2890 RPPSW    MVI     A,'P'
DCDC 02          2895          STAX    B
DCDD 03          2900          INX     B
DCDE 3E 53       2905          MVI     A,'S'
DCE0 02          2910          STAX    B
DCE1 03          2915          INX     B
DCE2 3E 57       2920          MVI     A,'W'
DCE4 C3 28 DD    2925          JMP     LSTOV
DCE7             2930 *
DCE7             2935 *A=FMT, HL->OP BYTE, D=OP BITS 3-5,
DCE7             2940 *BC->OUT BFFR, (D,PSW PUSHED)
DCE7 FE C0       2945 CKN      CPI     0C0H    FMT 6?
DCE9 C2 F2 DC    2950          JNZ     CKCND
DCEC             2955 *FMT N (RST)
DCEC 7A          2960          MOV     A,D
DCED C6 30       2965          ADI     '0'     ASCII
DCEF C3 28 DD    2970          JMP     LSTOV
DCF2             2975 *
DCF2 FE E0       2980 CKCND    CPI     0E0H    FMT 7?
DCF4 C2 10 DD    2985          JNZ     FD2     IF FD OR FDS
DCF7             2990 *CONDITION PART OF OP FOR J,C,R
DCF7 7A          2995          MOV     A,D
DCF8 07          3000          RLC     .       DOUBLE FOR 2 BYTE LOOKUP

DCF9 11 FF DD    3005          LXI     D,CNDTB  TABLE START
DCFC 83          3010          ADD     E
DCFD 5F          3015          MOV     E,A
DCFE 3E 00       3020          MVI     A,0
DD00 8A          3025          ADC     D
DD01 57          3030          MOV     D,A
DD02 1A          3035          LDAX    D       1ST LETTER
DD03 02          3040          STAX    B
DD04 03          3045          INX     B
DD05 13          3050          INX     D
DD06 1A          3055          LDAX    D
DD07 02          3060          STAX    B
DD08 03          3065          INX     B
DD09 3E 20       3070          MVI     A,' '   END OP FIELD
DD0B 02          3075          STAX    B
DD0C 03          3080          INX     B
DD0D C3 28 DD    3085          JMP     LSTOV
DD10             3090 *
DD10             3095 FD2      EQU     $       FD, FDS OR PREPROCESSED
P                        R
DD10 7A          3100          MOV     A,D     REG
DD11 CD 7C DD    3105          CALL    REGDE   DE->REG LETTER
DD14 F1          3110          POP     PSW     A=FMT
DD15 F5          3115          PUSH    PSW
DD16 FE A0       3120          CPI     0A0H
DD18 1A          3125          LDAX    D       LETTER
DD19 C2 28 DD    3130          JNZ     LSTOV   IF NOT FDS
DD1C             3135 *FDS
DD1C 02          3140          STAX    B
DD1D 03          3145          INX     B
DD1E CD 77 DD    3150          CALL    COMMA
DD21             3155 FS2      EQU     $
DD21 7E          3160          MOV     A,M     OP BYTE
DD22 E6 07       3165          ANI     7       S FIELD
DD24 CD 7C DD    3170          CALL    REGDE   DE->REG LETTER
DD27 1A          3175          LDAX    D       LETTER
DD28             3180 LSTOV    EQU     $       LAST VAR FIELD FROM OP
DD28 02          3185          STAX    B       TO BFFR
DD29 03          3190          INX     B
DD2A F1          3195          POP     PSW     A=FMT
DD2B D1          3200          POP     D
DD2C 23          3205 CKLNG    INX     H       ->AFTER OP BYTE
DD2D 13          3210          INX     D
DD2E             3215 *A=FMT, DE->OBK(1), HL->AFTER OP
DD2E             3220 *BC->OUT BFFR
DD2E F5          3225          PUSH    PSW
DD2F 1A          3230          LDAX    D       OBB(1)
DD30 D1          3235          POP     D       D=FMT
DD31 07          3240          RLC
DD32 07          3245          RLC
DD33 07          3250          RLC
DD34 E6 03       3255          ANI     3 INSTR LNGTH-1
DD36 CA 6C DD    3260          JZ      CR IF DONE
DD39 5F          3265          MOV     E,A
DD3A             3270 *FMTS 0,7 NEED NO COMMA
DD3A 7A          3275          MOV     A,D FMT
DD3B B7          3280          ORA     A
DD3C CA 44 DD    3285          JZ      RYT?
DD3F FE E0       3290          CPI     0E0H FMT7?
DD41 C4 77 DD    3295          CNZ     COMMA
DD44 7B          3300 RYT?     MOV     A,E LNGTH-1
DD45 5E          3305          MOV     E,M NXT RYT
DD46 23          3310          INX     H
DD47 FE 02       3315          CPI     2
DD49 FA 57 DD    3320          JM      LNG2    IF 2 BYTE INSTR
DD4C             3325 *3 BYTE INSTR
DD4C 56          3330          MOV     D,M     3D BYTE
DD4D 23          3335          INX     H
DD4E EB          3340          XCHG    .       DE<->HL
DD4F CD 86 DD    3345          CALL    HXSYM
DD52 EB          3350          XCHG
DD53 37          3355          STC     .       CARRY MEANS 3 BYTE INSTR

DD54             3360 * DE=ADDRESS VALUE FROM INSTR
DD54 C3 6D DD    3365          JMP     CR2
DD57             3370 *
DD57 3E 30       3375 LNG2     MVI     A,'0'
DD59 02          3380          STAX    B       TENTATIVE 1ST CHAR
DD5A 7B          3385          MOV     A,E     2ND INSTR BYTE
DD5B 0F          3390          RRC
DD5C E6 7F       3395          ANI     7FH
DD5E FE 50       3400          CPI     050H
DD60 7B          3405          MOV     A,E     DATA BYTE AGAIN
DD61 FA 65 DD    3410          JM      PREFXD  IF 1ST HEX IS 0-9
DD64 03          3415          INX     B       USE 0 PREFIX
```

```
DD65 CD 8F DD       3420 PRFXD  CALL   HXBYT
DD68 3E 48          3425        MVI    A,'H'
DD6A 02             3430        STAX   B
DD6B 03             3435        INX    B
DD6C AF             3440 CR     XRA    A
DD6D                3445 *CY=0 MEANS NOT A 3 BYTE INSTR
DD6D 3E 0D          3450 CR2    MVI    A,0DH
DD6F C3 74 DD       3455        JMP    LOUT
DD72                3460 *
DD72 3E 20          3465 BLANK  MVI    A,' '
DD74 02             3470 LOUT   STAX   B
DD75 03             3475        INX    B
DD76 C9             3480        RET
DD77                3485 *
DD77 3E 2C          3490 COMMA  MVI    A,','
DD79 C3 74 DD       3495        JMP    LOUT
DD7C                3500 *
DD7C 11 FE DD       3505 REGDE  LXI    D,REGTB
DD7F 83             3510        ADD    E
DD80 5F             3515        MOV    E,A
DD81 3E 00          3520        MVI    A,0
DD83 8A             3525        ADC    D
DD84 57             3530        MOV    D,A
DD85 C9             3535        RET    .       DE->REG LETTER
DD86                3540 *
DD86                3545 *CONVERT HL TO SYMBOL. USES A,PSW. ADV BC.
DD86 3E 48          3550 HXSYM  MVI    A,'H'  MAKES IT A SYMB
DD88 02             3555        STAX   B
DD89 03             3560        INX    B
DD8A 7C             3565        MOV    A,H
DD8B CD 8F DD       3570        CALL   HXBYT
DD8E 7D             3575        MOV    A,L
DD8F F5             3580 HXBYT  PUSH   PSW
DD90 0F             3585        RRC
DD91 0F             3590        RRC
DD92 0F             3595        RRC
DD93 0F             3600        RRC
DD94 CD 98 DD       3605        CALL   HXNBL
DD97 F1             3610        POP    PSW
DD98 E6 0F          3615 HXNBL  ANI    0FH
DD9A C6 30          3620        ADI    '0'
DD9C FE 3A          3625        CPI    '9'+1
DD9E DA 74 DD       3630        JC     LOUT   IF <=9
DDA1 C6 07          3635        ADI    'A'-'9'-1  MAKE A-F
DDA3 C3 74 DD       3640        JMP    LOUT
DDA6                3645 *
DDA6                3650 *SUBR. GETS PTR TO OP FORMAT BLOCK
DDA6                3655 *GROUP IS BITS 7-6 OF OP BYTE
DDA6                3660 * HL->OP BYTE (PRESERVED)
DDA6                3665 * DE->OP BLOCK ON EXIT
DDA6                3670 * USES C. PRESERVES B.
DDA6                3675 GTPTR  EQU    $       GET BLK PTR
DDA6 7E             3680        MOV    A,M    OP BYTE
DDA7 FE 40          3685        CPI    40H
DDA9 D2 B2 DD       3690        JNC    CKG1   IF NOT GRP 0
DDAC                3695 *GROUP 0
DDAC 11 B4 DE       3700        LXI    D,GRP0X  ->GROUP 0 INDEX TABLE

DDAF C3 D2 DD       3705        JMP    LKUPX  LOOK UP INDEX
DDB2                3710 *
DDB2 FE 80          3715 CKG1   CPI    80H
DDB4 D2 C2 DD       3720        JNC    CKG2
DDB7                3725 *GROUP 1
DDB7 FE 76          3730        CPI    76H
DDB9 3E 08          3735        MVI    A,8    INDX FOR HLT
DDBB CA DB DD       3740        JZ     GOTX   IF HLT
DDBE 3C             3745        INR    A      INDX FOR MOV
DDBF C3 DB DD       3750        JMP    GOTX
DDC2                3755 *
DDC2 FE C0          3760 CKG2   CPI    0C0H
DDC4 D2 CF DD       3765        JNC    GRP3
DDC7                3770 *GROUP 2
DDC7 0F             3775        RRC
DDC8 0F             3780        RRC
DDC9 0F             3785        RRC
DDCA E6 07          3790        ANI    7      INDX IS BITS 5-3 OF OP
DDCC C3 DB DD       3795        JMP    GOTX
DDCF                3800 *
DDCF                3805 GRP3   EQU    $      GROUP 3
DDCF 11 E4 DE       3810        LXI    D,GRP3X  ->GROUP 3 INDEX TABLE

DDD2 E6 3F          3815 LKUPX  ANI    3FH BITS 5-0 OF OP
DDD4 83             3820        ADD    E
DDD5 5F             3825        MOV    E,A
DDD6 3E 00          3830        MVI    A,0
DDD8 8A             3835        ADC    D
DDD9 57             3840        MOV    D,A
DDDA 1A             3845        LDAX   D      GET INDEX
DDDB                3850 *
DDDB                3855 GOTX   EQU    $
DDDB                3860 * A=INDEX OF OP BLOCK
DDDB                3865 * HL->OP BYTE
DDDB 11 06 DE       3870        LXI    D,OPTAB  ->OP FORMAT BLOCK TABL
                                   E
DDDE                3875 *TRIPLE INDEX (3 BYTES PER OP BLOCK)
DDDE 4F             3880        MOV    C,A
DDDF 81             3885        ADD    C
DDE0 81             3890        ADD    C      3*INDX (CAN'T CARRY)
DDE1 83             3895        ADD    E
DDE2 5F             3900        MOV    E,A
DDE3 3E 00          3905        MVI    A,0
DDE5 8A             3910        ADC    D
DDE6 57             3915        MOV    D,A    DE->BLOCK FOR THIS OP
DDE7 C9             3920        RET
DDE8                3925 *
DDE8                3930 OL4TB  EQU    $      4TH LETTER OF OPS
DDE8 20 44          3935        DW     'D '
DDEA 47 48          3940        DW     'HG'
DDEC 4C 58          3945        DW     'XL'
DDEE                3950 CNDTB  EQU    $      COND CODES FOR R,C,J
DDEE 4E 5A          3955        DW     'ZN'
DDF0 5A 20          3960        DW     ' Z'
DDF2 4E 43          3965        DW     'CN'
DDF4 43 20          3970        DW     ' C'
DDF6 50 4F          3975        DW     'OP'
DDF8 50 45          3980        DW     'EP'
DDFA 50 20          3985        DW     ' P'
DDFC 4D 20          3990        DW     ' M'
DDFE                3995 REGTB  EQU    $      REG NAMES
DDFE 42 43          4000        DW     'CB'
DE00 44 45          4005        DW     'ED'
DE02 48 4C          4010        DW     'LH'
DE04 4D 41          4015        DW     'AM'
DE06                4020 *OPERATION CODE FORMAT AND MNEMONIC TABLE
DE06                4025 *INDEX IS DERIVED ELSEWHERE AND USED *3.
DE06                4030 *SYMBOLS FOR DEBIASED LETTERS
DE06                4035 AA     EQU    1
DE06                4040 BB     EQU    2
DE06                4045 CC     EQU    3
DE06                4050 DD     EQU    4
DE06                4055 EE     EQU    5
DE06                4060 HH     EQU    8
DE06                4065 II     EQU    9
DE06                4070 JJ     EQU    10
DE06                4075 LL     EQU    12
DE06                4080 MM     EQU    13
DE06                4085 NN     EQU    14
DE06                4095 PP     EQU    16
DE06                4100 RR     EQU    18
DE06                4105 SS     EQU    19
```

```
DE06                    4110 TT      EQU    20
DE06                    4115 UU      EQU    21
DE06                    4120 VV      EQU    22
DE06                    4125 XX      EQU    24
DE06                    4130 *LENGTH CODES
DE06                    4135 L2      EQU    20H
DE06                    4140 L3      EQU    40H
DE06                    4145 *CODES FOR 4TH OP LETTER
DE06                    4150 OL4D    EQU    32
DE06                    4155 OL4G    EQU    64
DE06                    4160 OL4H    EQU    96
DE06                    4165 OL4L    EQU    128
DE06                    4170 OL4X    EQU    160
DE06                    4175 *FORMAT TYPES
DE06                    4180 FRPS    EQU    32      RP(SP)=1
DE06                    4185 FRPP    EQU    64      RP(PSW)=2
DE06                    4190 FS      EQU    96      S=3
DE06                    4195 FD      EQU    128     D=4
DE06                    4200 FDS     EQU    160     DS=5
DE06                    4205 FN      EQU    192     NUMERAL=6
DE06                    4210 FC      EQU    224     C=7
DE06                    4215 *
DE06                    4220 OPTAB   EQU    $
DE06                    4225 *3 BYTE BLOCK PER OP CODE FORMAT
DE06                    4230 *RIGHT 5 BITS OF BYTE IS CODED OP LETTER
DE06                    4235 *LEFT 3 BITS OF BYTE 1 IS FORMAT CODE
DE06                    4240 *BITS 5-6 OF BYTE 2 IS INSTR LNGTH (1,2,3)
DE06                    4245 *LEFT 3 BITS OF BYTE 3 IS CODED OP LETTER 4
DE06                    4250 *
DE06                    4255 *GROUP 2 INDEX=THIS+BITS 5-3 OF INSTR.
DE06 61                 4260         DB     AA+FS   ADD=0
DE07 04                 4265         DB     DD
DE08 04                 4270         DB     DD
DE09 61                 4275         DB     AA+FS   ADC=1
DE0A 04                 4280         DB     DD
DE0B 03                 4285         DB     CC
DE0C 73                 4290         DB     SS+FS   SUB=2
DE0D 15                 4295         DB     UU
DE0E 02                 4300         DB     BB
DE0F 73                 4305         DB     SS+FS   SBB=3
DE10 02                 4310         DB     BB
DE11 02                 4315         DB     BB
DE12 61                 4320         DB     AA+FS   ANA=4
DE13 0E                 4325         DB     NN
DE14 01                 4330         DB     AA
DE15 78                 4335         DB     XX+FS   XRA=5
DE16 12                 4340         DB     RR
DE17 01                 4345         DB     AA
DE18 6F                 4350         DB     OO+FS   ORA=6
DE19 12                 4355         DB     RR
DE1A 01                 4360         DB     AA
DE1B 63                 4365         DB     CC+FS   CMP=7
DE1C 0D                 4370         DB     MM
DE1D 10                 4375         DB     PP
DE1E                    4380 *GROUP 1
DE1E 08                 4385         DB     HH      HLT=8
DE1F 0C                 4390         DB     LL
DE20 14                 4395         DB     TT
DE21 AD                 4400         DB     MM+FDS  MOV=9
DE22 0F                 4405         DB     OO
DE23 16                 4410         DB     VV
DE24                    4415 *GROUPS 0 AND 3
DE24                    4420 *INDEXES FROM GRP0X AND GRP3X TABLES
DE24 8D                 4425         DB     MM+FD   MVI=10
DE25 36                 4430         DB     VV+L2
DE26 09                 4435         DB     II
DE27 84                 4440         DB     DD+FD   DCR=11
DE28 03                 4445         DB     CC
DE29 12                 4450         DB     RR
DE2A 89                 4455         DB     II+FD   INR=12
DE2B 0E                 4460         DB     NN
DE2C 12                 4465         DB     RR
DE2D 2C                 4470         DB     LL+FRPS  LXI=13
DE2E 58                 4475         DB     XX+L3
DE2F 09                 4480         DB     II
DE30 0C                 4485         DB     LL      LDA=14
DE31 44                 4490         DB     DD+L3
DE32 01                 4495         DB     AA
DE33 13                 4500         DB     SS      STA=15
DE34 54                 4505         DB     TT+L3
DE35 01                 4510         DB     AA
DE36 0C                 4515         DB     LL      LHLD=16
DE37 48                 4520         DB     HH+L3
DE38 2C                 4525         DB     LL+OL4D
DE39 13                 4530         DB     SS      SHLD=17
DE3A 48                 4535         DB     HH+L3
DE3B 2C                 4540         DB     LL+OL4D
DE3C 2C                 4545         DB     LL+FRPS  LDAX=18
DE3D 04                 4550         DB     DD
DE3E A1                 4555         DB     AA+OL4X
DE3F 33                 4560         DB     SS+FRPS  STAX=19
DE40 14                 4565         DB     TT
DE41 A1                 4570         DB     AA+OL4X
DE42 18                 4575         DB     XX      XCHG=20
DE43 03                 4580         DB     CC
DE44 48                 4585         DB     HH+OL4G
DE45 01                 4590         DB     AA      ADI=21
DE46 24                 4595         DB     DD+L2
DE47 09                 4600         DB     II
DE48 01                 4605         DB     AA      ACI=22
DE49 23                 4610         DB     CC+L2
DE4A 09                 4615         DB     II
DE4B 13                 4620         DB     SS      SUI=23
DE4C 35                 4625         DB     UU+L2
DE4D 09                 4630         DB     II
DE4E 13                 4635         DB     SS      SBI=24
DE4F 22                 4640         DB     BB+L2
DE50 09                 4645         DB     II
DE51 29                 4650         DB     II+FRPS  INX=25
DE52 0F                 4655         DB     NN
DE53 18                 4660         DB     XX
DE54 24                 4665         DB     DD+FRPS  DCX=26
DE55 03                 4670         DB     CC
DE56 18                 4675         DB     XX
DE57 24                 4680         DB     DD+FRPS  DAD=27
DE58 01                 4685         DB     AA
DE59 04                 4690         DB     DD
DE5A 04                 4695         DB     DD      DAA=28
DE5B 01                 4700         DB     AA
DE5C 01                 4705         DB     AA
DE5D 01                 4710         DB     AA      ANI=29
DE5E 2F                 4715         DB     NN+L2
DE5F 09                 4720         DB     II
DE60 18                 4725         DB     XX      XRI=30
DE61 32                 4730         DB     RR+L2
DE62 09                 4735         DB     II
DE63 0F                 4740         DB     OO      ORI=31
DE64 32                 4745         DB     RR+L2
DE65 09                 4750         DB     II
DE66 03                 4755         DB     CC      CPI=32
DE67 30                 4760         DB     PP+L2
DE68 09                 4765         DB     II
DE69 12                 4770         DB     RR      RLC=33
DE6A 0C                 4775         DB     LL
DE6B 03                 4780         DB     CC
DE6C 12                 4785         DB     RR      RRC=34
DE6D 12                 4790         DB     RR
DE6E 03                 4795         DB     CC
DE6F 12                 4800         DB     RR      RAL=35
DE70 01                 4805         DB     AA
DE71 0C                 4810         DB     LL
DE72 12                 4815         DB     RR      RAR=36
```

```
DE73 01          4820          DB      AA
DE74 12          4825          DB      RR
DE75 03          4830          DB      CC      CMA=37
DE76 0D          4835          DB      MM
DE77 01          4840          DB      AA
DE78 03          4845          DB      CC      CMC=38
DE79 0D          4850          DB      MM
DE7A 03          4855          DB      CC
DE7B 13          4860          DB      SS      STC=39
DE7C 14          4865          DB      TT
DE7D 03          4870          DB      CC
DE7E 0A          4875          DB      JJ      JMP=40
DE7F 4D          4880          DB      MM+L3
DE80 10          4885          DB      PP
DE81 FA          4890          DB      JJ+FC   J(COND)=41
DE82 40          4895          DB      L3
DE83 00          4900          DB      0
DE84 03          4905          DB      CC      CALL=42
DE85 41          4910          DB      AA+L3
DE86 8C          4915          DB      LL+0L4L
DE87 F3          4920          DB      CC+FC   C(COND)=43
DE88 40          4925          DB      L3
DE89 00          4930          DB      0
DE8A 12          4935          DB      RR      RET=44
DE8B 05          4940          DB      EE
DE8C 14          4945          DB      TT
DE8D F2          4950          DB      RR+FC   R(COND)=45
DE8E 00          4955          DB      0
DE8F 00          4960          DB      0
DE90 D2          4965          DB      RR+FN   RST=46
DE91 13          4970          DB      SS
DE92 14          4975          DB      TT
DE93 10          4980          DB      PP      PCHL=47
DE94 03          4985          DB      CC
DE95 88          4990          DB      HH+0L4L
DE96 50          4995          DB      PP+FRPP  PUSH=48
DE97 15          5000          DB      UU
DE98 73          5005          DB      SS+0L4H
DE99 50          5010          DB      PP+FRPP  POP=49
DE9A 0E          5015          DB      00
DE9B 10          5020          DB      PP
DE9C 18          5025          DB      XX      XTHL=50
DE9D 14          5030          DB      TT
DE9E 88          5035          DB      HH+0L4L
DE9F 13          5040          DB      SS      SPHL=51
DEA0 10          5045          DB      PP
DEA1 88          5050          DB      HH+0L4L
DEA2 09          5055          DB      II      IN=52
DEA3 2E          5060          DB      NN+L2
DEA4 00          5065          DB      00
DEA5 0E          5070          DB      00      OUT=53
DEA6 35          5075          DB      UU+L2
DEA7 14          5080          DB      TT
DEA8 05          5085          DB      EE      EI=54
DEA9 09          5090          DB      II
DEAA 00          5095          DB      00
DEAB 04          5100          DB      DD      DI=55
DEAC 09          5105          DB      II
DEAD 00          5110          DB      00
DEAE 0E          5115          DB      NN      NOP=56
DEAF 0F          5120          DB      00
DEB0 10          5125          DB      PP
DEB1 15          5130          DB      UU      UND=57 (UNDEFINED)
DEB2 0E          5135          DB      NN
DEB3 04          5140          DB      DD
DEB4             5145 *THE FOLLOWING TABLES CONVERT THE REMAINING
DEB4             5150 *6 (LOW) BITS OF INSTRS IN GROUPS 0 AND 3
DEB4             5155 *TO INDEXES TO OP BLOCKS
DEB4             5160 GRP0X   EQU     $       GROUP 0
DEB4 38          5165         DB      56      NOP
DEB5 0D          5170         DB      13      LXI
DEB6 13          5175         DB      19      STAX
DEB7 19          5180         DB      25      INX
DEB8 0C          5185         DB      12      INR
DEB9 0B          5190         DB      11      DCR
DEBA 0A          5195         DB      10      MVI
DEBB 21          5200         DB      33      RLC
DEBC 39          5205         DB      57      UNDEF
DEBD 1B          5210         DB      27      DAD
DEBE 12          5215         DB      18      LDAX
DEBF 1A          5220         DB      26      DCX
DEC0 0C          5225         DB      12      INR
DEC1 0B          5230         DB      11      DCR
DEC2 0A          5235         DB      10      MVI
DEC3 22          5240         DB      34      RRC
DEC4 39          5245         DB      57      UNDEF
DEC5 0D          5250         DB      13      LXI
DEC6 13          5255         DB      19      STAX
DEC7 19          5260         DB      25      INX
DEC8 0C          5265         DB      12      INR
DEC9 0B          5270         DB      11      DCR
DECA 0A          5275         DB      10      MVI
DECB 23          5280         DB      35      RAL
DECC 39          5285         DB      57      UNDEF
DECD 1B          5290         DB      27      DAD
DECE 12          5295         DB      18      LDAX
DECF 1A          5300         DB      26      DCX
DED0 0C          5305         DB      12      INR
DED1 0B          5310         DB      11      DCR
DED2 0A          5315         DB      10      MVI
DED3 24          5320         DB      36      RAR
DED4 39          5325         DB      57      UNDEF
DED5 0D          5330         DB      13      LXI
DED6 11          5335         DB      17      SHLD
DED7 19          5340         DB      25      INX
DED8 0C          5345         DB      12      INR
DED9 0B          5350         DB      11      DCR
DEDA 0A          5355         DB      10      MVI
DEDB 1C          5360         DB      28      DAA
DEDC 39          5365         DB      57      UNDEF
DEDD 1B          5370         DB      27      DAD
DEDE 10          5375         DB      16      LHLD
DEDF 1A          5380         DB      26      DCX
DEE0 0C          5385         DB      12      INR
DEE1 0B          5390         DB      11      DCR
DEE2 0A          5395         DB      10      MVI
DEE3 25          5400         DB      37      CMA
DEE4 39          5405         DB      57      UNDEF
DEE5 0D          5410         DB      13      LXI
DEE6 0F          5415         DB      15      STA
DEE7 19          5420         DB      25      INX
DEE8 0C          5425         DB      12      INR
DEE9 0B          5430         DB      11      DCR
DEEA 0A          5435         DB      10      MVI
DEEB 27          5440         DB      39      STC
DEEC 39          5445         DB      57      UNDEF
DEED 1B          5450         DB      27      DAD
DEEE 0E          5455         DB      14      LDA
DEEF 1A          5460         DB      26      DCX
DEF0 0C          5465         DB      12      INR
DEF1 0B          5470         DB      11      DCR
DEF2 0A          5475         DB      10      MVI
DEF3 26          5480         DB      38      CMC
DEF4             5485 *
DEF4             5490 GRP3X   EQU     $       GROUP 3
DEF4 2D          5495         DB      45      R
DEF5 31          5500         DB      49      POP
DEF6 29          5505         DB      41      J
DEF7 28          5510         DB      40      JMP
DEF8 2B          5515         DB      43      C
```

```
DEF9 30              5520        DB      48      PUSH
DEFA 15              5525        DB      21      ADI
DEFB 2F              5530        DB      46      RST
DEFC 2D              5535        DB      45      R
DEFD 2C              5540        DB      44      RET
DEFE 29              5545        DB      41      J
DEFF 39              5550        DB      57      UNDEF
DF00 2B              5555        DB      43      C
DF01 2A              5560        DB      42      CALL
DF02 16              5565        DB      22      ACI
DF03 2E              5570        DB      46      RST
DF04 2D              5575        DB      45      R
DF05 31              5580        DB      49      POP
DF06 29              5585        DB      41      J
DF07 35              5590        DB      53      OUT
DF08 2B              5595        DB      43      C
DF09 30              5600        DB      48      PUSH
DF0A 17              5605        DB      23      SUI
DF0B 2E              5610        DB      46      RST
DF0C 2D              5615        DB      45      R
DF0D 39              5620        DB      57      UNDEF
DF0E 29              5625        DB      41      J
DF0F 34              5630        DB      52      IN
DF10 2B              5635        DB      43      C
DF11 39              5640        DB      57      UNDEF
DF12 18              5645        DB      24      SBI
DF13 2E              5650        DB      46      RST
DF14 2D              5655        DB      45      R
DF15 31              5660        DB      49      POP
DF16 29              5665        DB      41      J
DF17 32              5670        DB      50      XTHL
DF18 2B              5675        DB      43      C
DF19 30              5680        DB      48      PUSH
DF1A 1D              5685        DB      29      ANI
DF1B 2E              5690        DB      46      RST
DF1C 2D              5695        DB      45      R
DF1D 2F              5700        DB      47      PCHL
DF1E 29              5705        DB      41      J
DF1F 14              5710        DB      20      XCHG
DF20 2B              5715        DB      43      C
DF21 39              5720        DB      57      UNDEF
DF22 1E              5725        DB      30      XRI
DF23 2E              5730        DB      46      RST
DF24 2D              5735        DB      45      R
DF25 31              5740        DB      49      POP
DF26 29              5745        DB      41      J
DF27 37              5750        DB      55      DI
DF28 2B              5755        DB      43      C
DF29 30              5760        DB      48      PUSH
DF2A 1F              5765        DB      31      ORI
DF2B 2F              5770        DB      46      RST
DF2C 2D              5775        DB      45      R
DF2D 33              5780        DB      51      SPHL
DF2E 29              5785        DB      41      J
DF2F 36              5790        DB      54      EI
DF30 2B              5795        DB      43      C
DF31 39              5800        DB      57      UNDEF
DF32 20              5805        DB      32      CPI
DF33 2E              5810        DB      46      RST
DF34                 5815 *
DF34                 5820 *BLANKS TO PAD UNLABELLED INSTRS (DS)
DF34 20 20 20 20     5825 OBENL   ASC     '       '
     20 20
DF3A                 5830 *SCRATCH MEMORY
DF3A                 5835 OBUF    DS      64      OUTPUT AND WORK BUFFER
DF7A                 5840 SYMTB   DS      2       ->TOP OF SYMBOL TBL
DF7C                 5845 END     EQU     $-1
DF7C                 5850         COM     END

AA       0001      4260 4275 4320 4330 4345 4360 4495 4510 4555 4570 4590
                   4605 4685 4700 4705 4710 4805 4820 4840 4910
ADDRS    DF9C      1225 1440 1815 2160
ALP      DB27      1385
ASOUT    DB39      1355
BACK1    DCA0      2520
BB       0002      4300 4310 4315 4640
BLANK    DD72      1255 1295 2465 2715
BS       DBC1
BSECK    DBE1      1840
BSLP     DBC4      1935
BYT2     DD44      3285
CC       0003      4285 4365 4445 4580 4610 4670 4755 4780 4795 4830 4845
                   4855 4870 4905 4920 4985
CHLDF    DB97      1415 1525 1930 2330 2390
CKCND    DCF2      2950
CKEOT    DC05      2000
CKG1     DDB2      3690
CKG2     DDC2      3720
CKLNG    DD2C      2725
CKN      DCF7      2805
CKWT     DBA9      1610
CMPLP    DBEE      2035 2055
CMPR1    DBF4      2095
CNDTB    DDEE      3005
COMMA    DD77      3150 3295
CR       DD6C      3260
CR2      DD6D      3365
CRLF     DB82      1230 1445 1570
DD       0004      4265 4270 4280 4440 4490 4550 4595 4665 4680 4690 4695
                   5100 5140
DF       DC1B
DFLP     DC1E      2335
DISA1    DC4F      1470 2275
DISL     DC41      1830 2370
DISNL    DC64      2400
DISPL    DB67      1505
DS       DB4F
DSLP     DB55      1530
EE       0005      4940 5085
END      DF7A
EOFP     D007      2170 2295 2305
FC       00E0      4890 4920 4950
FD       0080      4425 4440 4455
FD2      DD10      2835 2985
FDS      00A0      4400
FILL     DB37      1335 1345
FN       00C0      4965
FRPP     0040      2855 4995 5010
FRPS     0020      4470 4545 4560 4650 4665 4680
FS       0060      4260 4275 4290 4305 4320 4335 4350 4365
FS2      DD21      2745
GOTX     DDDB      3740 3750 3795
GRP0X    DEB4      3700
GRP3     DDCF      3765
GRP3X    DEF4      3810
GTPTR    DDA6      2455
HA       DAFF
HALP     DB05      1420
HH       0008      4385 4520 4535 4585 4990 5035 5050
HXRYT    DD8F      1275 1290 3420 3570
HXLP     DB13      1310
HXNBL    DD98      3605
HXSYM    DD86      1250 2440 3345
II       0009      4435 4455 4480 4600 4615 4630 4645 4650 4720 4735 4750
                   4765 5055 5090 5105
JJ       000A      4875 4890
L2       0020      4430 4595 4610 4625 4640 4715 4730 4745 4760 5060 5075
```

```
L3        0040      4475 4490 4505 4520 4535 4880 4895 4910 4925
LKUPA     DBE9      1855 2365
LKUPX     DDD2      3705
LL        000C      4390 4470 4485 4515 4525 4540 4545 4775 4810 4915
LNG2      DD57      3320
LOUT      DD74      3455 3495 3630 3640
LSTOV     DD28      2885 2925 2970 3085 3130
MM        000D      4370 4400 4425 4835 4850 4880
NN        000E      4325 4460 4655 4715 5060 5115 5135
OBFNL     DF34      1510
OBUF      DF39      1235 1450 1820
OL4D      0020      4525 4540
OL4G      0040      4585
OL4H      0060      5005
OL4L      0080      4915 4990 5035 5050
OL4TB     DDF8      2615
OL4X      00A0      4555 4570
OO        000F      4350 4405 4740 5015 5070 5120
OPCH3     DC87      2555
OPTAB     DE06      3870
OUTLN     DB72      1405 1515 1600
PAUSE     DBB5      1785
POPH      DBE0      1865
PP        0010      4375 4760 4885 4980 4995 5010 5020 5045 5125
PREXD     DD65      3410
PSCN      DAFC      1710
PSW       0006      2705 2720 2740 2785 2790 2845 2850 3110 3115 3195 3225
                    3580 3610
REGDE     DD7C      3105 3170
REGTB     DDFF      3505
RETRN     DAF0      1760 1795
RPPSW     DCDA      2860
RR        0012      4340 4355 4450 4465 4730 4745 4770 4785 4790 4800 4815
                    4825 4935 4950 4965
SCONV     DAF9      1695 2115
SINP      DAF3      1745 1780
SKP2      DC01      2015
SOUT      DAF6      1585 1620 1630 1640 1645
SS        0013      4290 4305 4500 4530 4560 4620 4635 4860 4970 5005 5040
ST        DC0F
SYMTB     DF79      1975 2120 2380
TT        0014      4395 4505 4565 4865 4945 4975 5030 5080
UU        0015      4295 4625 5000 5075 5130
VV        0016      4410 4430
XX        0018      4335 4475 4575 4660 4675 4725 5025
```

```
DUMP DAF0 DF39
DAF0: C3 04 C0 C3 1F C0 C3 19 C0 C3 3A C3 C3 10 C3 CD
DB00: 9C DB CD 82 DB 01 39 DF D5 C5 CD 86 DD CD 72 DD
DB10: 16 08 E5 7E CD 8F DD 23 7E CD 8F DD CD 72 DD 23
DB20: 15 C2 13 DB E1 16 10 7E FE 7F F2 37 DB FE 5E CA
DB30: 37 DB FE 20 F2 39 DB 3E 2E 02 03 23 15 C2 27 DB
DB40: 3E 0D 02 C1 CD 72 DB D1 CD 97 DB D2 05 DB C9 CD
DB50: 9C DB CD 82 DB 01 39 DF D5 C5 CD 4F DC C1 0A FE
DB60: 20 C2 67 DB 01 34 DF CD 72 DB D1 CD 97 DB D2 55
DB70: DB C9 0A FE 0D CA 82 DB C5 47 CD F6 DA C1 03 C3
DB80: 72 DB CD A9 DB 06 0D CD F6 DA 06 0A CD F6 DA 06
DB90: 00 CD F6 DA C3 F6 DA 7B 95 7A 9C C9 CD F9 DA E5
DBA0: 21 FF FF CD FC DA D1 EB C9 CD F3 DA C8 E6 7F CA
DBB0: F0 DA FE 20 C0 CD F3 DA CA B5 DB E6 7F CA F0 DA
DBC0: C9 CD 9C DB 01 39 DF D5 CD 61 DC D2 E1 DB E5 EB
DBD0: CD F9 DB DA F0 DB EB 73 23 72 23 36 FF 23 36 FF
```

```
DBE0: F1 D1 CD 97 DB D2 C4 DB C9 EB 2A 79 DF EB 1A FE
DBF0: FF CA 05 DC BD 13 C2 01 DC 1A 13 BC C2 FF DB 37
DC00: C9 13 C3 FF DB 13 1A FE FF 1B 1A C2 F4 DB C9 CD
DC10: F9 DA 22 79 DF 3F FF 77 23 77 C9 CD 9C DB E5 2A
DC20: 07 D0 44 4D E1 D5 C5 03 3E 30 02 03 02 03 02 03
DC30: 02 03 3F 20 02 03 CD 4F DC 3E 01 02 78 32 0B D0
DC40: 79 32 07 D0 C1 91 02 D1 CD 97 DB D2 1F DC C9 CD
DC50: F9 DB DA 61 DC E5 2A 79 DF EB CD 97 DB F1 DA 64
DC60: DC CD 86 DD C5 CD A6 DD C1 CD 72 DD 1A E6 1F CA
DC70: A0 02 03 13 1A E6 1F CA A0 DC C6 A0 02 03 13 1A
DC80: E6 1F C2 87 DC 3E E0 C6 A0 02 03 1A E6 E0 07 07
DC90: 07 C5 01 E8 DD 81 4F 3E 00 88 47 0A C1 02 03 1B
DCA0: 1B 1A E6 E0 E5 EE E0 C4 72 DD F1 CA 2C DD D5 EE
DCB0: 60 F5 CA 21 DD 7E 0F 0F 0F E6 07 57 F1 F5 D2 E7
DCC0: DC 7A E6 06 57 EE 06 C2 10 DD F1 F5 EE A0 CA DA
DCD0: DC 3F 53 02 03 3F 50 C3 28 DD 3E 50 02 03 3E 53
DCE0: 02 03 3E 57 C3 28 DD FE C0 C2 F2 DC 7A C6 30 C3
DCF0: 28 DD EE E0 C2 10 DD 7A 07 11 EE DD 83 5F 3E 00
DD00: 8A 57 1A 02 03 13 1A 02 03 3E 20 02 03 C3 28 DD
DD10: 7A CD 7C DD F1 F5 FE A0 1A C2 28 DD 02 03 CD 77
DD20: DD 7E E6 07 CD 7C DD 1A 02 03 F1 D1 23 13 F5 1A
DD30: D1 07 07 07 E6 03 CA 6C DD 5F 7A B7 CA 44 DD FE
DD40: E0 C4 77 DD 7B 5F 23 FE 02 FA 57 DD 5A 23 EB CD
DD50: 86 DD EB 37 C3 6D DD 3E 30 02 7B 0F E6 7F FE 50
DD60: 7B FA 65 DD 03 CD 8F DD 3E 48 02 03 AF 3E 0D C3
DD70: 74 DD 3E 20 02 03 C9 3E 2C C3 74 DD 11 FE DD 83
DD80: 5F 3E 00 8A 57 C9 3E 48 02 03 7C CD 8F DD 7D F5
DD90: 0F 0F 0F 0F CD 98 DD F1 E6 0F C6 30 FE 3A DA 74
DDA0: DD C6 07 C3 74 DD 7E FE 40 D2 B2 DD 11 B4 DE C3
DDB0: D2 DD FE 80 D2 C2 DD FE 76 3E 08 CA DB DD 3C C3
DDC0: DB DD FE C0 D2 CF DD 0F 0F 0F E6 07 C3 DB DD 11
DDD0: F4 DE E6 3F 83 5F 3E 00 8A 57 1A 11 06 DE 4F 81
DDE0: 81 83 5F 3E 00 8A 57 C9 20 44 47 48 4C 58 4E 5A
DDF0: 5A 20 4E 43 43 20 50 4F 50 45 50 20 4D 20 42 43
DE00: 44 45 48 4C 4D 41 61 04 04 61 04 03 73 15 02 73
DE10: 02 02 61 0E 01 78 12 01 6F 12 01 63 0D 10 08 0C
DE20: 14 AD 0F 16 8D 36 09 84 03 12 89 0E 12 2C 58 09
DE30: 0C 44 01 13 54 01 0C 48 2C 13 48 2C 2C 04 A1 33
DE40: 14 A1 18 03 48 01 24 09 01 23 09 13 35 09 13 22
DE50: 09 29 0E 18 24 03 18 24 01 04 04 01 01 01 2E 09
DE60: 18 32 09 0F 32 09 03 30 09 12 0C 03 12 12 03 12
DE70: 01 0C 12 01 12 03 0D 01 03 0D 03 13 14 03 0A 4D
DE80: 10 EA 40 00 03 41 8C E3 40 00 12 05 14 F2 00 00
DE90: D2 13 14 10 03 88 50 15 73 50 0F 10 18 14 88 13
DEA0: 10 88 09 2E 00 0F 35 14 05 09 00 04 09 00 0E 0F
DEB0: 10 15 0E 04 38 0D 13 19 0C 0B 0A 21 39 1B 12 1A
DEC0: 0C 0B 0A 22 39 0D 13 19 0C 0B 0A 23 39 1B 12 1A
DED0: 0C 0B 0A 24 39 0D 11 19 0C 0B 0A 1C 39 1B 10 1A
DEE0: 0C 0B 0A 25 39 0D 0F 19 0C 0B 0A 27 39 1B 0E 1A
DEF0: 0C 0B 0A 26 2D 31 29 2B 2B 30 15 2E 2D 2C 29 39
DF00: 2B 2A 16 2E 2D 31 29 35 2B 30 17 2E 2D 39 29 34
DF10: 2B 39 18 2E 2D 31 29 32 2B 30 1D 2E 2D 2F 29 14
DF20: 2B 39 1E 2E 2D 31 29 37 2B 30 1F 2E 2D 33 29 36
DF30: 2B 39 20 2E 20 20 20 20 20 20
```

```
ASSMX D9D0
```

Figure 4

```
D9D0                    0010 *COMMAND INTERPRETER
D9D0                    0020 *BY RICHARD GREENLAW   3/23/78
D9D0                    0030 *
D9D0 C3 F1 D9           0040         JMP    NEWCM  ENTRY POINT
D9D3                    0050 *VECTORS TO DISASSEMBLER ROUTINES
D9D3                    0060 *ADDRESSES SHOWN ARE FROM A COMMON
D9D3                    0070 *SYMBOL TABLE BUILT BY "COM"S DURING
D9D3                    0080 *ASSEMBLY OF THE DISASSEMBLER PROGRAM.
D9D3 C3 0F DC           0090 TST    JMP    ST
D9D6 C3 FF DA           0100 THA    JMP    HA
D9D9 C3 C1 DB           0110 TBS    JMP    BS
```

```
D9DC C3 4F DB    0120 TDS    JMP    DS
D9DF C3 1B DC    0130 TDF    JMP    DF
D9E2 C3 97 DB    0140 TCHLDE JMP    CHLDE
D9E5 C3 B2 DB    0150 TCRLF  JMP    CRLF
D9E8             0160 *VECTORS TO INPUT/OUTPUT ROUTINES
D9E8             0170 *(REQUIREMENTS IN ARTICLE)
D9E8 C3 1E C0    0180 SINP   JMP    0C01EH  INPUT TO A OR ZERO FLAG
D9EB C3 19 C0    0190 SOUT   JMP    0C019H  OUTPUT FROM B
D9EE C3 04 C0    0200 RETRN  JMP    0C004H  USER'S MONITOR (XX)
D9F1             0000 *
D9F1             0000 *DEFINE DELETE/BACKSPACE CODES
D9F1             0000 DEL    EQU    7FH     INPUT CODE
D9F1             0000 BSP    EQU    5FH     CODE ECHOED
D9F1             0210 *
D9F1             0220 *DEFINE ENTRY POINTS IN COMMON
D9F1             0230        COM    SCONV   GET ADDRESS IN HL
D9F1             0240        COM    PSCN    GET ADDRESS/DEFAULT
D9F1             0250 *
D9F1 31 F0 DA    0260 NEWCM  LXI    SP,STACK  JUST PAST STACK
D9F4 CD F5 D9    0270        CALL   TCRLF   NEW OUTPUT LINE
D9F7 06 3E       0280        MVI    B,'>'   PROMPT
D9F9 CD EB D9    0290        CALL   SOUT    OUTPUT
D9FC             0300 *LOAD COMMAND BUFFER
D9FC 21 BE DA    0310        LXI    H,CBUFF COMMAND BUFFER START
D9FF CD E8 D9    0320 GCMLP  CALL   SINP    GET A=INPUT OR Z FLAG
DA02 CA FF D9    0330        JZ     GCMLP   WAIT FOR INPUT
DA05 77          0340        MOV    M,A     TO BUFFER
DA06 FE 0D       0350        CPI    0DH
DA08 CA 33 DA    0360        JZ     OBEY    IF CARRIAGE RETURN
DA0B 47          0370        MOV    B,A     SAVE FOR ECHO
DA0C FE 7F       0380        CPI    DEL
DA0E C2 1F DA    0390        JNZ    ECHO    IF NOT DELETE
DA11             0400 *BACKSPACE UNLESS AT START
DA11 11 BE DA    0410        LXI    D,CBUFF
DA14 CD E2 D9    0420        CALL   TCHLDE COMPARE. USES A,PSW
DA17 D2 F1 D9    0430        JNC    NEWCM   IF PTR<=START
DA1A 2B          0440        DCX    H       BACKSPACE
DA1B 2B          0450        DCX    H       TO BALANCE INX H BELOW
DA1C 06 5F       0460        MVI    B,BSP   ECHO BACKSPACE
DA1E CD EB D9    0470 ECHO   CALL   SOUT    OUTPUT B. USES A, PSW
DA21 23          0480        INX    H       ADVANCE BUFFER POINTER
DA22 11 D0 DA    0490        LXI    D,CBUFF+1  END OF BUFFER
DA25 CD E2 D9    0500        CALL   TCHLDE  COMPARE TO POINTER
DA28 D2 FF D9    0510        JNC    GCMLP   IF ROOM FOR MORE
DA2B             0520 *
DA2B             0530 *STACK CAN BE UNBALANCED HERE
DA2B 06 3E       0540 ERROR  MVI    B,'?'
DA2D CD EB D9    0550        CALL   SOUT    OUTPUT ERROR MARK
DA30 C3 F1 D9    0560        JMP    NEWCM   START OVER
DA33             0570 *
DA33 21 C1 DA    0580 OBEY   LXI    H,CBUFF+2  BEYOND NAME
DA36 22 D0 DA    0590        SHLD   SPTR    BUFFER SCAN POINTER
DA39 2A BE DA    0600        LHLD   CBUFF   GET 2-LETTER COMMAND
DA3C 11 61 DA    0610        LXI    D,CTABL  COMMAND TABLE
DA3F 1A          0620 ENDCM  LDAX   D
DA40 13          0630        INX    D
DA41 B7          0640        ORA    A
DA42 CA 2B DA    0650        JZ     ERROR   IF END OF TABLE
DA45 BD          0660        CMP    L
DA46 C2 5A DA    0670        JNZ    NXT1    IF NO MATCH
DA49 1A          0680        LDAX   D
DA4A 13          0690        INX    D
DA4B BC          0700        CMP    H
DA4C C2 5B DA    0710        JNZ    NXT2    IF NO MATCH
DA4F             0720 *COMMAND FOUND: GET ADDRESS
DA4F 1A          0730        LDAX   D
DA50 13          0740        INX    D
DA51 6F          0750        MOV    L,A
DA52 1A          0760        LDAX   D
DA53 67          0770        MOV    H,A
DA54 CD 60 DA    0780        CALL   GO      DO THE WORK
DA57 C3 F1 D9    0790        JMP    NEWCM   NEXT COMMAND
DA5A             0800 *
DA5A             0810 *THIS TABLE ENTRY DOESN'T MATCH
DA5A 13          0820 NXT1   INX    D       SKIP 2ND LETTER
DA5B 13          0830 NXT2   INX    D SKIP ADDRESS
DA5C 13          0840        INX    D
DA5D C3 3F DA    0850        JMP    ENDCM   TRY NEXT TABLE ENTRY
DA60             0860 *
DA60 E9          0870 GO     PCHL   GO TO ROUTINE IN HL
DA61             0880 *
DA61             0890 *COMMAND TABLE
DA61             0900 *
DA61 53 54       0910 CTABL  ASC    'ST'    COMMAND NAME
DA63 D3 D9       0920        DW     TST     ADDRESS OF ROUTINE
DA65 48 41       0930        ASC    'HA'
DA67 D6 D9       0940        DW     THA
DA69 42 53       0950        ASC    'BS'
DA6B D9 D9       0960        DW     TBS
DA6D 44 53       0970        ASC    'DS'
DA6F DC D9       0980        DW     TDS
DA71 44 46       0990        ASC    'DF'
DA73 DF D9       1000        DW     TDF
DA75 58 58       1010        ASC    'XX'    EXIT
DA77 EE D9       1020        DW     RETRN
DA79 00          1030        DB     0       END
DA7A             1040 *
DA7A             1050 *SUBROUTINES TO GET AND CONVERT HEX
DA7A             1060 *ADDRESSES FROM COMMAND BUFFER TO HL
DA7A             1070 *
DA7A 2A D0 DA    1080 SCONV  LHLD   SPTR    SCAN POINTER
DA7D EB          1090        XCHG
DA7E 21 00 00    1100        LXI    H,0     INITIALIZE RESULT
DA81 1A          1110 ENDLP  LDAX   D       FROM COMMAND BUFFER
DA82 13          1120        INX    D
DA83 FE 20       1130        CPI    ' '
DA85 CA 81 DA    1140        JZ     ENDLP   IGNORE BLANKS
DA88             1150 *FOUND SOMETHING
DA88 1B          1160        DCX    D       POINT TO IT
DA89 EB          1170 HEXLP  XCHG   UPDATE SCAN POINTER
DA8A 22 D0 DA    1180        SHLD   SPTR
DA8D EB          1190        XCHG
DA8E 1A          1200        LDAX   D       FROM BUFFER
DA8F FE 0D       1210        CPI    0DH
DA91 C8          1220        RZ     .       IF CARRIAGE RETURN
DA92 FE 20       1230        CPI    ' '
DA94 C8          1240        RZ     .       IF END OF ADDRESS
DA95 D6 30       1250        SUI    '0'
DA97 FA 2B DA    1260        JM     ERROR IF <ASCII 0
DA9A FE 0A       1270        CPI    10
DA9C FA AB DA    1280        JM     PACK    IF 0-9
DA9F D6 11       1290        SUI    'A'-'0'  TRY A-F
DAA1 FA 2B DA    1300        JM     ERROR   IF <'A'
DAA4 C6 0A       1310        ADI    10      MAKE A-F INTO 10-15
DAA6 FE 10       1320        CPI    16
DAA8 F2 2B DA    1330        JP     ERROR   >F
DAAB 29          1340 PACK   DAD    H       SHIFT OLD VALUE LEFT
DAAC 29          1350        DAD    H
DAAD 29          1360        DAD    H
DAAE 29          1370        DAD    H
DAAF B5          1380        ORA    L
DAB0 6F          1390        MOV    L,A     PACK NEW DIGIT
DAB1 13          1400        INX    D       ->NEXT BUFFER CHAR
DAB2 C3 89 DA    1410        JMP    HEXLP   GET MORE HEX DIGITS
DAB5             1420 *
DAB5             1430 *SUBROUTINE TO GET (SECOND) ADDRESS
DAB5             1440 *IN HL BUT LEAVE HL UNCHANGED IF NONE
DAB5             1450 *OR ADDRESS IS ZERO.
DAB5             1460 *
DAB5 E5          1470 PSCN   PUSH   H       SAVE FOR DEFAULT
```

```
DA86 CD 7A DA           1480        CALL    SCONV   GET HL=ADDRESS OR ZERO
DA89 D1                 1490        POP     D       DEFAULT VALUE
DA8A 7D                 1500        MOV     A,L
DA8B B4                 1510        ORA     H
DA8C C0                 1520        RNZ     .       IF ADDRESS NONZERO,
DA8D EB                 1530        XCHG    .       USE DEFAULT
DA8E C9                 1540        RET
DA8F                    1550 *
DA8F                    1570 SP     EQU     6       STACK POINTER REGISTER
DA8F                    1580 *
DA8F                    1593 CBUFF  DS      17      COMMAND BUFFER
DAD0                    1600 CBUFE  EQU     $-1
DAD0                    1610 SPTR   DS      2       SCAN POINTER
DAD2                    1620        DS      30      GENEROUS STACK
DAF0                    1633 STACK  EQU     $

BSP     005E    0460
CBUFF   DACF    0490
CBUFE   DADF    0310 0410 0580 0600
CTABL   DA61    0610
DEL     007F    0380
ECHO    DA1F    0390
ERROR   DA2B    0650 1260 1300 1330
ENDCM   DA3F    0850
ENDLP   DA81    1140
GCMLP   D9FF    0330 0510
GO      DA60    0780
HEXLP   DA89    1410
NEWCM   D9F1    0040 0430 0560 0790
NXT1    DA5A    0670
NXT2    DA5B    0710
OBEY    DA33    0360
PACK    DAAB    1280
PSCN    DAB5
RETRN   D9FF    1020
SCONV   DA7A    1480
SINP    D9EB    0320
SOUT    D9FB    0290 0470 0550
SP      0006    0260
SPTR    DAD0    0590 1080 1180
STACK   DAF0    0240
TBS     D9D9    0960
TCHLD   D9F2    0420 0500
TCRLF   D9F5    0270
TDF     D9DF    1000
TDS     D9DC    0980
THA     D9D6    0940
TST     D9D3    0920

DUMP D9D0 DAFF
D9D0: C3 F1 D9 C3 0F DC C3 FF DA C3 C1 DB C3 4F DB C3
D9E0: 1B DC C3 97 DB C3 82 DB C3 1F C0 C3 19 C0 C3 04
D9F0: C0 31 F0 DA CD F5 D9 06 3E CD FB D9 21 BF DA CD
DA00: FB D9 CA FF D9 77 FF 0D CA 33 DA 47 FF 7F C2 1F
DA10: DA 11 BF DA CD F2 D9 D2 F1 D9 2B 2B 06 5F CD FB
DA20: D9 23 11 D0 DA CD E2 D9 D2 FF D9 06 3E CD FB D9
DA30: C3 F1 D9 21 C1 DA 22 D0 DA 2A BF DA 11 61 DA 1A
DA40: 13 B7 CA 2B DA BD C2 5A DA 1A 13 B8 C2 5B DA 1A
DA50: 13 6F 1A 67 CD 60 DA C3 F1 D9 13 13 13 C3 3E DA
DA60: F9 53 54 D3 D9 48 41 D6 D9 42 53 D9 D9 44 53 DC
DA70: D9 44 46 DF D9 58 58 FF D9 00 2A D0 DA FB 21 00
DA80: 00 1A 13 FE 20 CA 81 DA 1B FB 22 D0 DA FB 1A FE
DA90: 0D C8 FE 20 C8 D6 30 FA 2B DA FE 0A FA AB DA D6
DAA0: 11 FA 2B DA C6 0A FF 10 F2 2F DA 29 29 29 29 B5
DAB0: 6F 13 C3 89 DA E5 CD 7A DA D1 7D B4 C0 EF C9
```

```
READY
DUMP 154720 155276
154720: 303 361 331 303 017 334 303 377 332 303 301 333 303 117 333 303
154740: 033 334 303 227 333 303 202 333 303 037 300 303 031 300 303 004
154760: 300 061 360 332 315 345 331 006 076 315 353 331 041 277 332 315
155000: 350 331 312 377 331 147 376 015 312 063 332 107 376 177 302 036
155020: 332 021 277 332 315 342 331 322 361 331 053 053 006 137 315 353
155040: 331 043 021 320 332 315 342 331 322 377 331 006 077 315 353 331
155060: 303 361 331 041 301 332 042 320 332 052 277 332 021 141 332 032
155100: 023 267 312 053 332 275 302 132 332 032 023 274 302 133 332 032
155120: 023 157 032 147 315 140 332 303 361 331 023 023 023 303 077 332
155140: 351 123 124 323 331 110 101 326 331 102 123 331 331 104 123 334
155160: 331 104 106 337 331 130 130 356 331 000 052 320 332 353 041 000
155200: 000 032 023 376 040 312 201 332 033 353 042 320 332 353 032 376
155220: 015 310 376 040 310 326 060 372 053 332 376 012 372 253 332 326
155240: 021 372 053 332 306 012 376 020 362 053 332 051 051 051 051 265
155260: 157 023 303 211 332 345 315 172 332 321 175 264 300 353 311
```

```
P
READY
ASSM DAAF
```

Figure 5

```
DAAF                    0010 *ROUTINE TO DEFINE DISASSEMBLER COMMANDS
DAAF                    0020 *IN SOLOS(TM) OR CUTER(TM) CUSTOM COMMAND
DAAF                    0030 *TABLE, ERASING ANY EXISTING
DAAF                    0040 *CUSTOM COMMAND DEFINITIONS.
DAAF                    0050 *
DAAF                    0060 *BY RICHARD GREENLAW 3/25/78
DAAF                    0070 *
DAAF                    0080 *THE FOLLOWING VALUE IS FOR SOLOS(TM) 1.3
DAAF                    0090 CUTAB  EQU    0C83CH  CUSTOM COMMAND TABLE
DAAF                    0100 *
DAAF                    0110 *THE COMMAND ADDRESSES WERE DEFINED IN
DAAF                    0120 *THE COMMON SYMBOL TABLE WHEN THE
DAAF                    0130 *DISASSEMBLER WAS ASSEMBLED.
DAAF                    0140 *
DAAF 21 53 54           0150        LXI    H,'TS'  'ST' COMMAND NAME
DAB2 22 3C C8           0160        SHLD   CUTAB
DAB5 21 0F DC           0170        LXI    H,ST    COMMAND ADDRESS
DAB8 22 3E C8           0180        SHLD   CUTAB+2
DABB 21 42 53           0190        LXI    H,'SB'  'BS'
DABE 22 40 C8           0200        SHLD   CUTAB+4
DAC1 21 C1 DB           0210        LXI    H,BS
DAC4 22 42 C8           0220        SHLD   CUTAB+6
DAC7 21 44 53           0230        LXI    H,'SD'  'DS'
DACA 22 44 C8           0240        SHLD   CUTAB+8
DACD 21 4F DB           0250        LXI    H,DS
DAD0 22 46 C8           0260        SHLD   CUTAB+10
DAD3 21 44 46           0270        LXI    H,'FD'  'DF'
DAD6 22 48 C8           0280        SHLD   CUTAB+12
DAD9 21 1B DC           0290        LXI    H,DF
DADC 22 4A C8           0300        SHLD   CUTAB+14
DADF 21 48 41           0310        LXI    H,'AH'  'HA'
DAE2 22 4C C8           0320        SHLD   CUTAB+16
DAE5 21 FF DA           0330        LXI    H,HA
DAE8 22 4E C8           0340        SHLD   CUTAB+18
DAEB AF                 0350        XRA    A       ZERO
DAEC 32 50 C8           0360        STA    CUTAB+20  END OF LIST
DAEF C9                 0370        RET
READY
```

THE INTEL 8086

BY MIKE GABRIELSON
Box 2692
Stanford, Calif. 94305

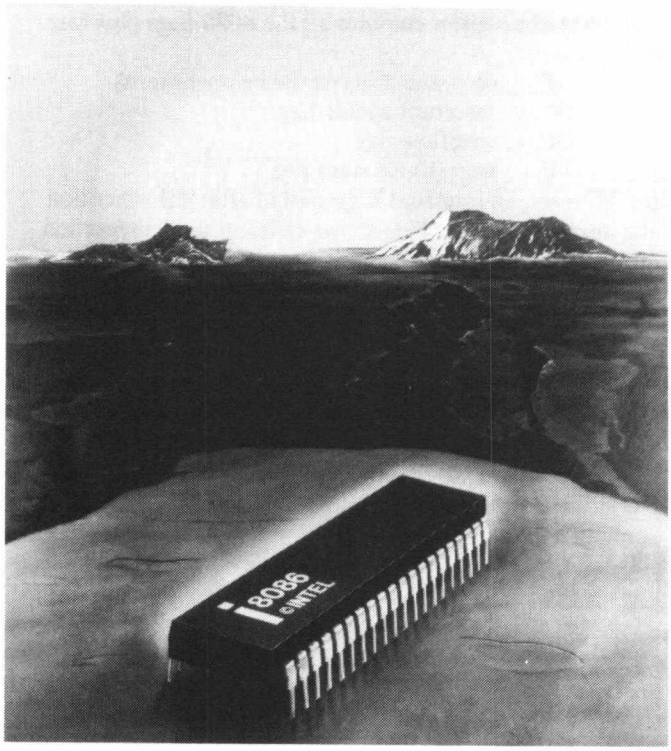
Intel has been slowly leaking information on its new 16 bit microprocessor, the 8086. Except for the concluding paragraph, the following is intended to be an objective report describing the various features of this new device. I have purposely skipped most of the hardware details (such as pinouts and interfacing), since that is not my area of expertise. Of course, all specifications are subject to change by Intel, and no endorsement of this article by them is implied.

Memory Organization

The 8086 can address one million bytes of memory, using a set of "segment" registers. Every byte has an address. Locations $FFFF0_{16}$ through $FFFFF_{16}$ are reserved for the initial program load routine (the CPU jumps to location $FFFF0_{16}$ on reset). Locations 0 through $3FF_{16}$ are reserved for interrupt pointers (described later). Addresses are 16 bits and stored byte-reversed in the 8080 fashion, but a 20 bit address can be constructed before a memory access by offsetting a 16-bit effective address four bits to the right of a segment address contained in one of four segment registers, filling in the four low order bits of the segment address with zeros and adding the two. The segment registers define the base address of four segments (a "code" or instruction segment, a stack segment, a data segment, and an extra data segment) of 64K bytes each, with each segment falling on 16-bit boundaries. The segment registers are named:

CS	code
SS	stack
DS	data
ES	extra data

and each is 16 bits in length. As described above, they are extended with four low order zero bits to form a 20 bit segment base address. The entire megabyte of memory can be divided into these four segments and is dynamically allocatable by changing the contents of the segment registers. In addition, an entire memory block up to 64 bytes, can be moved from one group of memory locations into a completely different group. Word (16 bit) operands can reside at even or odd addresses. Referencing word data (but not instruction fetches) at odd addresses does require an extra memory cycle, so it is worth it to locate word data (especially the stack) on even address boundaries.



Other Registers

In addition to the segment registers, the 8086 includes all the 8080 registers (but with different names), plus a few new additions. The four 16-bit general registers have high and low order bytes which are independently addressable. Thus, the 16-bit accumulator, AX, contains the bytes AH and AL. The other 16-bit registers include four pointer and index registers as well as the instruction pointer (program counter) and flag register. A breakdown and comparison to 8080 register structure looks like this:

8086 Name	8080 Name	Meaning
		General Registers
AX	–	16 bit accumulator
AH	–	high order byte of AX
AL	A	low order byte of AX
BX	HL	16 bit base register
BH	H	8 bit register, high order byte of BX
BL	L	8 bit register, low order byte of BX
CX	BC	16 bit count register
CH	B	8 bit register, high order byte of CX
CL	C	8 bit register, low order byte of CX
DX	DE	16 bit data register
DH	D	8 bit register, high order byte of DX
DL	E	8 bit register, low order byte of DX
		Pointers and Index Registers
SP	SP	16 bit stack pointer
BP	–	16 bit base pointer
SI	–	16 bit source index register
DI	–	16 bit destination index register
		Instruction Pointer and Flags
IP	PC	16 bit instruction pointer (program counter)
flags	–	16 bit status register

The 8086 status register contains all the 8080 flags plus four new flag bits:

DF	direction flag (for string operations)
IF	interrupt enable flag
OF	overflow flag
TF	trap (single step) flag

When TF is set, an interrupt is generated after the execution of the next instruction. This allows efficient implementation of software to allow single stepping, such as in a debugger.

Addressing Modes

8086 instructions can be from one to five bytes long, with a large number of addressing modes, but the basic two-byte, two-operand instruction allows byte or word operations from register to register, register to memory, or memory to register, and is formatted as follows:

Field Name	# Bits	Effect
opc	6	opcode, selects instruction
d	1	defines direction: operand 1 "to" or "from" operand 2
w	1	selects operand width: word or byte
mod	2	sets addressing mode (whether r/m is a reg)
reg	3	selects an 8 or 16 bit register with w
r/m	3	same as reg for one mode, else selects which registers compute the effective operand address

The various combinations allow a memory operand to be addressed in the following ways:

- as an 8 or 16 bit immediate operand
- relative to a segment register (CS, SS, DS, ES) using a 16 bit offset
- with an index register (BX, SI, or DI)
- relative to a base or index register (BX, BP, SI, or DI) using an 8 or 16 bit offset
- with a base register (BX or BP) plus an index register (SI or DI)
- relative to a base register (BX or BP) plus an index register (SI or DI) using an 8 or 16 bit offset

Instruction Set

For every 8080 instruction, there is a corresponding 8086 instruction (or, in some cases, a short sequence of instructions) to perform the same function. Nearly all instructions can operate on either 8 or 16 bit quantities in either registers or memory (push and pop are notable exceptions). And of course, several additions and extensions have been made . . .

- An XLAT (translate) instruction searches a table to find the corresponding value of an incoming variable. Once found, this corresponding value is substituted for the original value. Therefore, using the iteration control and string primitives with the translate instruction, a task such as translating a string of EBCDIC characters into ASCII characters can easily be performed using only seven instruction bytes.
- An LEA (load effective address) instruction automatically computes the effective address of a memory operand.
- There are single instructions to add, subtract, multiply, or divide signed or unsigned 8 or 16 bit operands in binary, BCD, or ASCII, carried out in two's complement form. This allows easy implementation of multiple precision algorithms.
- The CBW and CWD instructions convert (sign extend) bytes to words and words to double words.
- Variable length shifts and rotates (using a count in CL) are implemented.
- A set of one byte instructions use SI and DI as pointers to a pair of byte or word strings (the pointers are optionally automatically incremented or decremented after each operation). These are the "string primitives" and during their execution, the SI and DI pointers as well as the CX count register are updated after each operation:

Move	move source string to destination string
Compare	compare strings
Search	search destination string
Load	load from source string
Store	store (fill) destination string

- A REP (repeat) instruction automatically repeats a following string instruction (move, compare, etc.) for a specified count in CS, or while or until the Z bit is set or reset.
- CALL, JMP, and RET instructions can transfer within the current segment or to another segment. Calls and jumps use relative addresses to allow position independent (relocatable) code. Conditional transfers must be within ±128 bytes of the instruction. A return can automatically pop a number of parameters off the stack.
- Various LOOP instructions perform a decrement of CX and jump ±128 bytes depending on the result (and optionally testing the Z flag as set by instructions in the loop).
- Software interrupt instructions provide efficient breakpoint and overflow handling.
- A WAIT for external interrupt instruction is automatically re-executed at the end of interrupt handling to allow synchronization with external devices.
- An ESC (escape) instruction places memory operands on the bus so that other processors can receive instruction streams and use 8086 addressing modes.
- A LOCK instruction can precede any other instruction so that while the instruction executes, other processors in a multiple processor configuration are locked out and cannot interfere. This allows easy implementations of test-and-set or semaphore operations in software.

Speed

Both five megaHertz and eight megaHertz clocks are supported. At 5 MHz, one clock cycle is 200 nanoseconds; at 8 MHz, it is 125 nanoseconds. The fastest instructions, such as register to register moves, take two cycles. The slowest instructions, such as a signed 16 bit divide, take over 150 cycles.

Instruction execution is overlapped with the fetch of the next instruction. In the 8086 up to six instruction bytes can be accessed and queued prior to actual execution. The queue forms the pipeline of instructions and time is saved because the instructions are accessed while the instruction execution portion of the microprocessor is busy, thus resulting in significantly improved operating efficiency in the microprocessor. "Dead time" on the memory bus is greatly reduced, thus

allowing moderate speed, less expensive memories to be used while still achieving high CPU performance. Intel consistently claims ten times the "performance" of the 8080.

Interrupts

The reserved memory locations 0 through $3FF_{16}$ are divided into 256_{10} 4 byte interrupt pointers, where each pointer corresponds to an interrupt "type". Every interrupt supplies an 8 bit type number, which is used to select the appropriate 4 byte pointer. The pointer is interpreted as a 16 bit segment address and a 16 bit offset in the segment, and control is transferred to that address. The first 32 interrupt pointers are dedicated:

type 0: divide error
1: single step
2: non-maskable interrupt
3: breakpoint
4: overflow
5-31: reserved

The 8086 also provides software maskable interrupts. The total overhead at each interrupt (for pushing flags, disabling interrupts, transferring control, returning from the interrupt, and so on), is 88 cycles.

Another feature afforded by the architectural enchancements of the microprocessor addressing capability is improved input/output addressing. The improvement permits a maximum of 65K bytes of I/O port space. The ports may be either 8-bit or 16-bit ports and the 16-bit ports may be located at either odd or even addresses. Moreover, indirect I/O addressing is now possible with the 8086, a feature not available on the 8080 or 8085.

Conclusion

The 8086 is a sophisticated device with a very large instruction set. At one time, the various types of CPUs (micros, minis, midis, and maxis) could be differentiated by speed and addressing capabilities, but the trend is to minimize those differences. The 8086 may be "micro" in size or price, but certainly not in performance.

For $360, Intel is marketing the SDK-86 System Design Kit, a complete single board system including CPU, 8K ROM, 2K RAM, and I/O. The board also contains an interactive LED display and keyboard. The system monitor is contained in ROM and there is a wire-wrap area for custom interfacing. All bus and parallel I/O signals are TTL compatible and serial I/O is 20μA current loop TTY or RS232. The system comes with a comprehensive design library.

'TINY-C' INTERPRETER FOR 8080 AND PDP-11

News Release *Received: 78 June 22*

An interpreter for a subset of the C structured programming language which runs on both DEC PDP-11 and Intel 8080 processors is available from Tiny-c Associates. The tiny-c Owner's Manual includes a complete reference description of the language, a tutorial walkthrough of a training program, lots of sample programs including comments on their programming style, and a description of the Program Preparation System. It also includes commented source code listings of both the 8080 and the PDP-11 interpreters, and a chapter on how the interpreter works.

Tiny-c is intended primarily for the education and hobbyist markets. The tiny-c language handles integer and character data, and arrays of either type. Other features include compound statements, if-else and while statements, global and local variables, pointer variables, and functions. Functions may have arguments and may return results. Recursion is allowed. The interpreter also recognizes calls to functions written in machine language. These, too, may have arguments and return results. A minimum of 16K bytes of memory is recommended to run tiny-c.

The package includes a Program Preparation System with which the user can write, edit, run, debug, store, recall, and link tiny-c programs. The PPS includes a standard library of tiny-c software tools. The PPS is written in tiny-c. Thus it serves as an example of a significant use of tiny-c, and is also easily adapted to a user's or operating system's requirements.

The Owner's Manual is available for $40 from Tiny-c Associates, P.O. Box 269, Holmdel, New Jersey 07733. Machine readable copies of the interpreter are available separately on several formats of tape and disk.

ERROR MESSAGE MANUAL FOR NORTH STAR DOS

News Release *Received: 78 June 15*

The NORTH STAR DISK SYSTEM ERROR MESSAGE SUMMARY is for users of the North Star Computers floppy disk system. This manual is a compilation of all DOS and BASIC error messages given by the North Star system. The manual gives the meanings of these messages in easily understood terms. Numerous examples extracted from programs serve to illustrate what program errors cause these messages so they can be avoided. The manual serves as a reference source in debugging programs when occurrence of these messages appears. Also illustrated are error messages resulting from improper DOS input commands as well as BASIC commands. It is available from HSC COMPUTER SERVICES, LTD., P.O. Box 43, Brooklyn, NY 11236, by sending check or money order in amount of $5.00 plus 50 cents shipping and handling, available from stock.

MKB-2 KEYBOARD

News Release *Received: 78 June 26*

The MKB-2 Keyboard by MicroAge, is designed for use with the new 64 and 80 character display video boards.

Included as standard in the MKB-2 are a numeric key pad, upper and lower case, cursor control keys, 2-key rollover, and auto repeat on all keys. The MKB-2 is assembled in a heavy duty steel case with parallel interface, strobe or pulse, on-board regulation (5v, 12v), complete with standard DB25S connector, and black double-injection molded keys. List: $149.00 Contact MicroAge, 1425 W 12th Place #101, Tempe, AZ 85281; (602) 967-1421.

A 1K UTILITIES PACKAGE FOR THE Z-80

BY LES HANCOCK
305 Lexington Ave. 7-C
New York, NY 10016

This is a companion piece to the 1K Debugger, written to run on a Digital Group Z-80 system with ZE audio ROM, TVC-64 interface and TVC-64 monitor (see *DDJ* no. 24, p. 12). Starting the program causes it to clear the TV screen and print this menu:

```
                TVC-64 UTILITIES

      Ø MONITOR
            1 SEARCH
                  2 COMPARE
                        3 MOVE
                              4 HEX MODE
```

You can pick any item on the menu by keying in a number from 0 through 4. (Anything else is ignored.) 0 returns control to the monitor and 4 sets hex mode for routines 1, 2 or 3. (Default mode is octal.)

As soon as one of the three utilities is chosen, the screen is erased and a new banner appears. You'll be prompted for the appropriate addresses and data entries. Carriage returns are *not* used to signal the end of an entry, except in the case of the target string for item 1. When a utility has all the parameters it needs, it runs to completion, then waits for you to press a key. The space bar restarts the utility; any other key sends you back to the menu, reinitializing the program and resetting octal mode. There's no way to recover from typing errors. You'll have to use the RESET switch to abort, then start over.

Item 1 is a string search utility. It searches any range of memory for a string up to 9 bytes in length. Bytes are specified in octal/hex notation or in ASCII, and any byte in the string (except the first, of course) can be left "wild"—in other words, you can search for discontinuous strings.

This last feature is really handy. Suppose you're looking for jumps to page zero. There are 256 possible 3-byte strings to hunt for: C3 00 00, C3 01 00, . . . C3 FF 00. You can find all of them in one go by specifying the target string as C3 // 00, where // is a wild byte.

The syntax of item 1 is as follows:

```
SEARCH A-B FOR STRING
      A?aa
      B?bb
STRING?
```

where aa and bb are two 16-bit addresses entered by the user. As for the target string itself, you can define it in either of two entry modes:

1. *Octal or hex notation.* Simply type in a string of bytes in octal or hex. Don't try to separate them; the program prints a blank after each byte entered. To specify a wild byte, press the slash key (/). The slash will be echoed twice or three times, depending on whether you're using hex or octal. To terminate the string enter a carriage return. If your string is 9 bytes long no c/r is necessary; the program will take off as soon as the last number is typed.
2. *Text strings.* There is provision for entering text directly in ASCII, but it's necessary to distinguish between the two ASCIIs supported by the Digital Group. One is the code we all know, with the high bit off. The letter A is hex 41, the numeral 1 is hex 31. The other ASCII, with bit 7 set for use as a strobe, codes the letter A as hex C1, the numeral 1 as hex B1. Where you want to search for text with the high bit on, enter a single quotation mark ('), then type in the characters. For straight ASCII, enter a double quotation mark ("), then the text. No wild bytes permitted in either case.

The address of the first byte of each occurrence of the string will be displayed at the console. If there are enough addresses to fill the screen, output ends and the program waits for a keypress. Then it erases the screen and goes ahead.

Some examples:

```
SEARCH A - B FOR STRING
      A?ØØØØØØ
      B?ØØ6ØØØ
 STRING?315 372 ØØØ (c/r)
 ØØØ35Ø ØØ1237 ØØ2Ø16 ØØ2Ø73 ØØ21Ø1 ØØ3232 ØØ3324 ØØ4176
 ØØ4262 ØØ4277 ØØ4316 ØØ4361


SEARCH A - B FOR STRING
      A?ØØØØØØ
      B?ØØØ377
 STRING?315 /// ØØØ (c/r)
 ØØØ114 ØØØ13Ø ØØØ133 ØØØ153 ØØØ156 ØØØ162 ØØØ17Ø ØØØ214
 ØØØ223 ØØØ226 ØØØ35Ø ØØØ357


 SEARCH A - B FOR STRING
       A? Ø5 ØØ <---------- hex mode
       B? Ø5 FF
  STRING?'Assembler <------ 9 bytes, no c/r needed
  Ø5 9A


 SEARCH A - B FOR STRING
       A?ØØØØØØ
       B?Ø1ØØØØ
  STRING?"Assemble (c/r)
  NONE
```

Item 2 is a routine that compares corresponding bytes in two ranges of memory. It reports out every mismatch, giving addresses, the bytes themselves in octal or hex, and the ASCII

equivalent of each byte. (Note: the Digital Group's TVC-64 board uses an MCM6571L character generator with special symbols and Greek letters below the blank.) Output is slanted across the screen for better legibility. As with item 1, you can clear a filled screen by touching anything on the keyboard.

An example:

```
COMPARE A - B TO C - D
  A?Ø1ØØ33
  B?Ø1ØØ45
  C?ØØ5255

          (erases screen)

Ø1ØØ33=311='I  ØØ5255=343'c
  Ø1ØØ34=311='I  ØØ5256=357='o
    Ø1ØØ35=311='I  ØØ5257=36Ø='p
      Ø1ØØ36=3Ø1='A  ØØ526Ø=371='y
        Ø1ØØ37=377=    ØØ5261=ØØØ="<
          Ø1ØØ4Ø=31Ø='H  ØØ5262=ØØØ="<
            Ø1ØØ41=31Ø='H  ØØ5263=ØØØ="<
              Ø1ØØ42=31Ø='H  ØØ5264=ØØØ="<
                Ø1ØØ43=3ØØ='@  ØØ5265=ØØØ="<
                  Ø1ØØ44=276='>  ØØ5266=ØØØ="<
                    Ø1ØØ45=3Ø1='A  ØØ5267=ØØØ="<
```

Item 3 copies a block of bytes from one range of memory to another. You supply the starting and ending addresses that define the source block, then the starting address for the new block. The utility transfers the data and prints the new ending address.

```
MOVE A - B TO C - D
    A?ØØØØØØ
    B?ØØ1ØØØ
    C?1ØØØØØ
    D?1Ø1ØØØ <------- this line printed by the
                      program itself after
                      the move is completed
```

Footnotes To The Code

1. Line 180: The TVC-64 monitor contains a routine, here called TVE, for printing a string of high-bit ASCII characters at the TV console. Before calling it the user sets HL to point to the first byte in the string. A rubout in the string (hex FF) will erase the screen; a value *n* from 1 through hex 7F will print *n* spaces; an empty byte (0) ends the string.
2. Line 200: The monitor subroutine NUMIN gets a byte from the keyborad, entered by the user in octal or hex notation. (See note #3.) But what if we already have the first of the two hex or three octal digits that represent a byte? In that case a call to FINISH will dive into the middle of NUMIN and come back with the rest of the byte.
3. Lines 310, 5910: By Digital Group convention, the byte at hex address 01A7 is a mode code. If it's ASCII 'H', the current mode is hex; if not, octal.
4. Line 1570: SEARCH keeps its not-found flag and its paging counter in alternate registers E and B respectively. The counter is initialized to 96 because the first page of addresses is printed below the routine's headers and parameters. If the first page overflows, the counter is reset to 127 for subsequent pages. (Space is left for a slash to indicate that there's more to come; hence the odd number.)

The Code Listing

```
67E9                    0100 * * * * * * * * * * * * * * * * *
67E9                    0110 * 1K UTILITIES, BY LES HANCOCK   *
67E9                    0120 *                                *
67E9                    0130 *   VERSION OF MAY 12, 1978      *
67E9                    0140 * * * * * * * * * * * * * * * * *
67E9                    0150 *
7000                    0160         ORG  7000H
7000                    0170 *
7000                    0180 TVE     EQU  0200H       SEE NOTE #1
7000                    0190 MONTR   EQU  0500H       MONITOR ENTRY ADDR
7000                    0200 FINISH  EQU  04A9H       SEE NOTE #2
7000                    0210 KEYBD   EQU  01A8H       GETS 1 BYTE FROM KEYBD
7000                    0220 ERASE   EQU  00E6H       ERASES TV SCREEN
7000                    0230 NUMIN   EQU  04A6H       GETS HEX OR OCT BYTE
7000                    0240 *                          FROM KEYBD
7000                    0250 NUMOUT  EQU  0246H       PRINTS BYTE AS HEX OR OCT
7000                    0260 *
7000                    0270 * INITIALIZATION
7000                    0280 *
7000 31 00 70           0290 START   LD   SP,START    SET STACK POINTER
7003 3E CF              0300         LD   A,'O'       SET MODE TO OCTAL
7005 32 A7 01           0310         LD   (01A7H),A     SEE NOTE #3
7008 21 76 73           0320 START2  LD   HL,UTIL     CLEAR SCREEN AND PRINT
700B CD 00 02           0330         CALL TVE           MENU OF ROUTINES
700E 3E 81              0340         LD   A,'1'
7010 CD A7 70           0350         CALL NUMS
7013 21 36 73           0360         LD   HL,SRCH
7016 CD 00 02           0370         CALL TVE
7019 06 3E              0380         LD   B,62D
701B CD 81 70           0390         CALL LINE
701E 3E 82              0400         LD   A,'2'
7020 CD A7 70           0410         CALL NUMS
7023 21 3D 73           0420         LD   HL,CMPR
7026 CD 00 02           0430         CALL TVE
7029 06 3D              0440         LD   B,61D
702B CD 81 70           0450         CALL LINE
702E 3E 83              0460         LD   A,'3'
7030 CD A7 70           0470         CALL NUMS
7033 21 45 73           0480         LD   HL,MV
7036 CD 00 02           0490         CALL TVE
7039 06 40              0500         LD   B,64D
703B CD 81 70           0510         CALL LINE
703E 3E 84              0520         LD   A,'4'
7040 CD A7 70           0530         CALL NUMS
7043 21 4A 73           0540         LD   HL,HX
7046 CD 00 02           0550         CALL TVE
7049                    0560 *
7049                    0570 * SELECT ROUTINE
7049                    0580 * TO BE USED
7049                    0590 *
7049 CD A8 01           0600 INPUT0  CALL KEYBD       GET A BYTE
704C FE B0              0610         CP   0B0H        IF NOT AN ASCII
704E 38 F9              0620         JR   C,INPUT0      NUMBER 0<=N<5,
7050 FE B5              0630         CP   0B5H          GET ANOTHER
7052 30 F5              0640         JR   NC,INPUT0
7054 E6 0F              0650         AND  0FH         CONVERT TO BINARY VALUE
7056 07                 0660         RLCA             TIMES 2
7057 6F                 0670         LD   L,A         GET RESULT INTO HL
7058 26 00              0680         LD   H,0
705A 11 65 70           0690         LD   DE,TABLE
705D 19                 0700         ADD  HL,DE       ADD TABLE START ADDR
705E 22 62 70           0710         LD   (IMMED),HL  STORE AT IMMED
7061                    0720         DB   2AH         INSTRUCTION: LOAD
  2A
7062                    0730 *                          HL EXTENDED
7062                    0740 IMMED   DW   TABLE       OPERAND FOR INSTRUCTION
  65 70
7064 E9                 0750         JP   (HL)        JUMP TO SELECTED ROUTINE
7065                    0760 *
7065                    0770 * TABLE OF ENTRY POINTS
7065                    0780 * FOR UTILITY ROUTINES
7065                    0790 *
7065                    0800 TABLE   DW   MONTR
  00 05
7067                    0810         DW   SEARCH
  B4 70
7069                    0820         DW   COMPAR
  56 72
706B                    0830         DW   MOVE
  FD 71
706D                    0840         DW   HEXMOD
  26 73
706F                    0850 *
706F                    0860 * SUBRS AT THE FOLLOWING
706F                    0870 * LABELS: "SPACE" PRINTS
706F                    0880 * 1 BLANK ON TV SCREEN;
706F                    0890 * "QUEST" PRINTS CHARACTER
706F                    0900 * IN ACCUMULATOR, THEN "?";
706F                    0910 * "TV" PRINTS CHARACTER IN
706F                    0920 * ACCUMULATOR
706F                    0930 *
706F F5                 0940 SPACE   PUSH AF
7070 3E A0              0950         LD   A,0A0H
7072 18 06              0960         JR   SPC
7074 CD 79 70           0970 QUEST   CALL TV
7077 3E BF              0980 QUEST1  LD   A,'?'
7079 F5                 0990 TV      PUSH AF
707A D3 00              1000 SPC     OUT  0
707C AF                 1010         XOR  A
707D D3 00              1020         OUT  0
707F F1                 1030         POP  AF
7080 C9                 1040         RET
7081                    1050 *
```

```
7081                    1060 * SUBR TO PRINT # OF BLANK
7081                    1070 * SPACES INDICATED BY VALUE
7081                    1080 * IN B
7081                    1090 *
7081 CD 6F 70           1100 LINE   CALL SPACE
7084 10 FB              1110        DJNZ LINE
7086 C9                 1120        RET
7087                    1130 *
7087                    1140 * SUBR TO PROMPT FOR TWO
7087                    1150 * 2-BYTE VALUES, RETURNING
7087                    1160 * THEM IN HL AND DE
7087                    1170 *
7087 3E C1              1180 GETAB  LD   A,'A'
7089 CD 74 70           1190        CALL QUEST
708C CD A6 04           1200        CALL NUMIN
708F 67                 1210        LD   H,A
7090 CD A6 04           1220        CALL NUMIN
7093 6F                 1230        LD   L,A
7094 06 38              1240        LD   B,56D
7096 CD 81 70           1250        CALL LINE
7099 3E C2              1260        LD   A,'B'
709B CD 74 70           1270        CALL QUEST
709E CD A6 04           1280        CALL NUMIN
70A1 57                 1290        LD   D,A
70A2 CD A6 04           1300        CALL NUMIN
70A5 5F                 1310        LD   E,A
70A6 C9                 1320        RET
70A7                    1330 *
70A7                    1340 * SUBR TO PRINT ASCII
70A7                    1350 * NUMBER IN A, THEN ONE
70A7                    1360 * BLANK SPACE
70A7                    1370 *
70A7 CD 79 70           1380 NUMS   CALL TV
70AA CD 6F 70           1390        CALL SPACE
70AD C9                 1400        RET
70AE                    1410 *
70AE                    1420 * SUBR TO WAIT FOR KEYPRESS,
70AE                    1430 * THEN COMPARE INPUT WITH ASCII BLANK
70AE                    1440 *
70AE CD A8 01           1450 KEYPR  CALL KEYBD
70B1 FE A0              1460        CP   0A0H
70B3 C9                 1470        RET
70B4                    1480 * * * * * * * * * * * * * * * * * *
70B4                    1490 * ROUTINE TO SEARCH A
70B4                    1500 * SPECIFIED RANGE OF MEMORY
70B4                    1510 * FOR A GIVEN STRING
70B4                    1520 *
70B4                    1530 * INITIALIZATION
70B4                    1540 *
70B4 AF                 1550 SEARCH XOR  A          CLEAR WILD-BYTE FLAG
70B5 32 FC 71           1560        LD   (MASK),A
70B8 06 60              1570        LD   B,96D      SEE NOTE #4
70BA CB C3              1580        SET  0,E        SET NOT-FOUND FLAG
70BC D9                 1590        EXX
70BD CD E6 00           1600        CALL ERASE      CLEAR SCREEN AND
70C0 21 36 73           1610        LD   HL,SRCH      PRINT BANNER
70C3 CD 00 02           1620        CALL TVE
70C6 21 53 73           1630        LD   HL,AB
70C9 CD 00 02           1640        CALL TVE
70CC 21 5C 73           1650        LD   HL,FOR
70CF CD 00 02           1660        CALL TVE
70D2 21 61 73           1670        LD   HL,STRNG
70D5 CD 00 02           1680        CALL TVE
70D8 06 2E              1690        LD   B,46D
70DA CD 81 70           1700        CALL LINE
70DD                    1710 *
70DD                    1720 * GET PARAMETERS
70DD                    1730 *
70DD CD 87 70           1740        CALL GETAB      GET SEARCH RANGE
70E0 E5                 1750        PUSH HL         SEE NEXT6
70E1 E5                 1760        PUSH HL
70E2 D5                 1770        PUSH DE
70E3 06 33              1780        LD   B,51D
70E5 CD 81 70           1790        CALL LINE
70E8 21 61 73           1800        LD   HL,STRNG   PROMPT FOR STRING
70EB CD 00 02           1810        CALL TVE
70EE CD 77 70           1820        CALL QUEST1
70F1 11 00 09           1830        LD   DE,0900H   SET D TO 9 (MAX BYTE
70F4                    1840 *                        COUNT) AND SET E TO
70F4                    1850 *                        ZERO
70F4 DD 21 A5 73        1860        LD   IX,INBUF   GET INITIAL INPUT
70F8 DD E5              1870        PUSH IX           BUFFER ADDRESS
70FA FD E1              1880        POP  IY
70FC                    1890 *
70FC                    1900 * INPUT OF STRING BEGINS.  FLAGS AS FOLLOWS.
70FC                    1910 * BIT 7 OF E IS SET IF INPUT = ASCII STRING.
70FC                    1920 * OTHERWISE INPUT IS ASSUMED TO BE A SERIES
70FC                    1930 * OF BYTES IN HEX OR OCTAL REPRESENTATION.
70FC                    1940 *
70FC                    1950 * IF THE FIRST BYTE INPUT IS " THEN THE
70FC                    1960 * USER WANTS TO SEARCH FOR ASCII BYTES
70FC                    1970 * WITH THE HIGH BIT ON.  IN THAT CASE
70FC                    1980 * BIT 1 OF E IS SET.
70FC                    1990 *
70FC                    2000 * IF THE FIRST BYTE INPUT IS ' THEN THE
70FC                    2010 * USER WANTS TO SEARCH FOR STRAIGHT
70FC                    2020 * ASCII BYTES, WITH THE HIGH BIT OFF.
70FC                    2030 * IN THAT CASE BIT 1 OF E IS ZERO.
70FC                    2040 *
70FC CD A8 01           2050 INPUT  CALL KEYBD      GET BYTE
70FF FE 8D              2060        CP   8DH        IS IT C/R?
7101 28 60              2070        JR   Z,ENDPUT   YES, END INPUT
7103 CB 7B              2080        BIT  7,E        IS STRING LITERAL?
7105 28 0B              2090        JR   Z,NOTLIT
7107 CD 79 70           2100        CALL TV         YES, ECHO IT
710A CB 4B              2110        BIT  1,E        IS IT STRAIGHT ASCII?
710C 28 4D              2120        JR   Z,LOAD     NO, LOAD AS IS
710E E6 7F              2130        AND  7FH        ELSE KILL HIGH BIT
7110 18 49              2140        JR   LOAD
7112 FE A7              2150 NOTLIT CP   0A7H       BYTES THAT ARRIVE HERE
7114                    2160 *                        ARE EITHER NUMERIC OR
7114                    2170 *                        ELSE 1ST BYTE INPUT.
7114                    2180 *                        IS BYTE ' ?
7114 28 06              2190        JR   Z,SETLIT   YES; STRING WILL BE
7116                    2200 *                        STRAIGHT ASCII
7116 FE A2              2210        CP   0A2H       IS IT " ?
7118 20 09              2220        JR   NZ,NUMBER
711A CB C8              2230        SET  1,E        YES; STRING IS ASCII
711C                    2240 *                        WITH HIGH BIT ON
711C CB FB              2250 SETLIT SET  7,E
711E CD 79 70           2260        CALL TV         ECHO ' OR "
7121 18 D9              2270        JR   INPUT      GET 1ST BYTE OF STRING
7123 FE A0              2280 NUMBER CP   0A0H        BYTES THAT ARRIVE HERE
7125                    2290 *                        ARE NUMERIC; IF BLANK,
7125 28 D5              2300        JR   Z,INPUT      IGNORE THEM
7127 FE AF              2310        CP   '/'        IS THIS A "WILD" BYTE?
7129 20 2A              2320        JR   NZ,NTWILD
7128                    2330 *
7128                    2340 * BEGIN WILD-BYTE ROUTINE
7128                    2350 *
7128 7A                 2360        LD   A,D        GET BYTE COUNT
712C FE 09              2370        CP   9D         IS THIS 1ST BYTE?
712E CA 00 70           2380        JP   Z,START    YES, ABORT
7131 3D                 2390        DEC  A          DEC BYTE COUNT
7132 07                 2400        RLCA
7133 07                 2410        RLCA
7134 07                 2420        RLCA
7135 F6 C6              2430        OR   0C6H
7137 32 3E 71           2440        LD   (BIT0),A   A NOW CONTAINS OPERAND
713A                    2450 *                      OF FORTHCOMING BIT-SET
713A                    2460 *                      INSTRUCTION
713A 21 FC 71           2470        LD   HL,MASK
713D                    2480        DB   0CBH       INSTRUCTION: SET (HL),N
  CB
713E                    2490 BIT0   DS   1          WHERE THIS BYTE IS N
713F                    2500 *
713F                    2510 * WHEN BYTE COUNT IS 8, THE ABOVE INSTRUCTION
713F                    2520 * WILL BE SET (HL),7; WHEN 7, IT WILL BE
713F                    2530 * SET (HL),6; UNDOSWEITER
713F                    2540 *
713F 3E AF              2550        LD   A,'/'      ECHO SLASH TWICE
7141 CD 79 70           2560        CALL TV
7144 CD 79 70           2570        CALL TV
7147 3A A7 01           2580        LD   A,(01A7H)  IF MODE IS OCTAL, ECHO
714A FE C8              2590        CP   'H'          ANOTHER
714C 28 05              2600        JR   Z,HEX
714E 3E AF              2610        LD   A,'/'
7150 CD 79 70           2620        CALL TV
7153 18 03              2630 HEX    JR   NEXT5
7155 CD A9 04           2640 NTWILD CALL FINISH     COMPLETE THE HEX OR
7158                    2650 *                      OCTAL BYTE & ECHO IT
7158 CD 6F 70           2660 NEXT5  CALL SPACE
7158 DD 77 00           2670 LOAD   LD   (IX+0),A   LOAD BYTE INTO BUFFER
715E DD 2B              2680        DEC  IX         DECR POINTER
7160 15                 2690        DEC  D          DECR COUNTER
7161 20 99              2700        JR   NZ,INPUT   IF 9 BYTES NOT USED
7163                    2710 *                      YET, GET ANOTHER
7163                    2720 *
7163                    2730 * SPACE TO NEXT LINE
7163                    2740 *
7163 3E 09              2750 ENDPUT LD   A,9D      GET NO. OF BYTES
7165 92                 2760        SUB  D          USED INTO D
7166 57                 2770        LD   D,A
7167 CB 7B              2780        BIT  7,E        WAS INPUT ASCII?
7169 20 04              2790        JR   NZ,SHORT
716B 07                 2800        RLCA            TIMES 4
716C 07                 2810        RLCA
716D D6 01              2820        SUB  1          MINUS 1
716F C6 08              2830 SHORT  ADD  8D         PLUS 8
7171 ED 44              2840        NEG
7173 C6 40              2850        ADD  64D        SUBTRACTED FROM 64
7175 47                 2860        LD   B,A        PRINT THAT MANY
7176 CD 81 70           2870        CALL LINE         BLANKS
7179                    2880 *
7179                    2890 * SET UP FOR SEARCH
7179                    2900 *
7179 E1                 2910 NEXT6  POP  HL         GET TOP OF RANGE
717A C1                 2920        POP  BC           & START OF RANGE
717B B7                 2930        OR   A          RESET CARRY FLAG
717C                    2940 *                        (OR NEXT INSTRUCTION
717C                    2950 *                        MIGHT BLOW UP)
717C ED 42              2960        SBC  HL,BC      GET SIZE OF RANGE
717E E5                 2970        PUSH HL           (WHICH GOES
717F C1                 2980        POP  BC           INTO BC)
7180 03                 2990        INC  BC         PLUS 1 (IN CASE THE
7181                    3000 *                        STRING FOUND IS
7181                    3010 *                        COTERMINOUS WITH
7181                    3020 *                        THE RANGE SET)
7181 E1                 3030        POP  HL         START ADDR INTO HL
7182 5A                 3040        LD   E,D        SAVE BYTE COUNT IN E
7183                    3050 *
7183                    3060 * SEARCH TILL FIRST BYTE OF
7183                    3070 * STRING IS FOUND, OR TILL
7183                    3080 * SEARCH RANGE IS EXHAUSTED
7183                    3090 *
7183 53                 3100 LOOK2  LD   D,E        BYTE COUNT TO D
7184 FD E5              3110        PUSH IY         RESTORE POINTER
7186 DD E1              3120        POP  IX           TO START OF BUFF
7188 DD 7E 00           3130        LD   A,(IX+0)   GET 1ST BYTE
718B ED B1              3140        CPIR            SEARCH FOR IT
718D 28 07              3150        JR   Z,NEXT7    FOUND IT!
718F D9                 3160 BCZERO EXX             OUT OF RANGE; CHECK
7190 CB 43              3170        BIT  0,E          NOT-FOUND FLAG,
7192 20 59              3180        JR   NZ,NONE      TAKE APPROPRIATE
7194 18 5D              3190        JR   FIN          EXIT
7196                    3200 *
7196                    3210 * 1ST BYTE WAS FOUND; SEARCH
7196                    3220 * FOR SUCCESSIVE BYTES IN STRING
7196                    3230 *
7196 15                 3240 NEXT7  DEC  D          DECR BYTE COUNT
7197 28 25              3250        JR   Z,PRINT2   FOUND WHOLE STRING!
7199 78                 3260        LD   A,B        BC = 0?
```

```
719A B1          3270         OR   C
719B 28 F2       3280         JR   Z,BCZERO     YES, YOU'RE DONE
719D C5          3290         PUSH BC           IS NEXT BYTE "WILD"?
719E             3300 *                           LET'S FIND OUT
719E 78          3310         LD   A,E          NUMBER OF BYTES IN STRING
719F 92          3320         SUB  D              MINUS CURRENT BYTE COUNT
71A0 47          3330         LD   B,A          RESULT IS SHIFT COUNTER
71A1 3A FC 71    3340         LD   A,(MASK)   GET MASK TO BE SHIFTED
71A4 05          3350 ROTATE  DEC  B            ROTATE MASK LEFT 0 IF
71A5 28 03       3360         JR   Z,TEST       NEXT BYTE IS 2ND IN
71A7 07          3370         RLCA              STRING, 1 IF NEXT BYTE
71A8 18 FA       3380         JR   ROTATE       IS 3RD, ETC
71AA CB 7F       3390 TEST    BIT  7,A        IF BIT 7 IS SET, NEXT BYTE
71AC             3400 *                           WILL BE "WILD"
71AC C1          3410         POP  BC
71AD DD 2B       3420         DEC  IX         DECR POINTER
71AF DD 7E 00    3430         LD   A,(IX+0)   GET NEXT BYTE
71B2 28 04       3440         JR   Z,NOSKIP   IF BYTE WAS WILD, GO
71B4 23          3450         INC  HL           RIGHT ON TO NEXT
71B5 08          3460         DEC  BC
71B6 18 DE       3470         JR   NEXT7
71B8 ED A1       3480 NOSKIP  CPI              FIND NEXT BYTE OF
71BA 28 DA       3490         JR   Z,NEXT7       STRING?
71BC 18 C5       3500         JR   LOOK2      IF NOT, START OVER
71BE             3510 *
71BE             3520 * PRINT ADDRESS OF
71BE             3530 * STRING WHEN FOUND
71BE             3540 *
71BE E5          3550 PRINT2 PUSH HL
71BF D5          3560        PUSH DE
71C0 D9          3570        EXX
71C1 CB 83       3580 GO3    RES  0,E        RESET NOT-FOUND FLAG
71C3 10 0D       3590        DJNZ NEXT8      DECR LOCATIONS-PRINTED
71C5             3600 *                       COUNTER, SKIP ON 0
71C5 3E AF       3610        LD   A,'/'      PRINT  /
71C7 CD 79 70    3620        CALL TV
71CA CD A8 01    3630        CALL KEYBD      WAIT FOR KEYPRESS
71CD CD E6 00    3640        CALL ERASE      CLEAR SCREEN
71D0 06 7F       3650        LD   B,127D     RESET COUNTER
71D2 D9          3660 NEXT8  EXX
71D3 2B          3670 BAXUP  DEC  HL         GET ADDR OF 1ST BYTE
71D4 1D          3680        DEC  E            IN STRING FOUND
71D5 20 FC       3690        JR   NZ,BAXUP
71D7 5C          3700        LD   E,H        PRINT ADDRESS
71D8 CD 46 02    3710        CALL NUMOUT
71DB 5D          3720        LD   E,L
71DC CD 46 02    3730        CALL NUMOUT
71DF CD 6F 70    3740        CALL SPACE
71E2 CD 6F 70    3750        CALL SPACE
71E5 D1          3760        POP  DE
71E6 E1          3770        POP  HL
71E7 78          3780        LD   A,B        DOES THIS FIND
71E8 B1          3790        OR   C            COINCIDE WITH END
71E9 20 98       3800        JR   NZ,LOOK2     OF RANGE?  IF SO,
71EB 18 06       3810        JR   FIN          STOP, ELSE SEARCH AGAIN
71ED 21 68 73    3820 NONE   LD   HL,NON      PRINT "NONE"
71F0 CD 00 02    3830        CALL TVE
71F3 CD AE 70    3840 FIN    CALL KEYPR      WAIT FOR KEYPRESS
71F6 CA B4 70    3850        JP   Z,SEARCH   IF SPACE BAR IS
71F9             3860 *                       PRESSED, SEARCH AGAIN
71F9 C3 00 70    3870        JP   START      ELSE BACK TO THE MENU
71FC             3880 MASK   DS   1
71FD             3890 * * * * * * * * * * * * * * * * * *
71FD             3900 * ROUTINE TO MOVE
71FD             3910 * A BLOCK IN MEMORY
71FD             3920 *
71FD CD E6 00    3930 MOVE   CALL ERASE        CLEAR SCREEN AND
7200 21 45 73    3940        LD   HL,MV          PRINT BANNER
7203 CD 00 02    3950        CALL TVE
7206 21 53 73    3960        LD   HL,AB
7209 CD 00 02    3970        CALL TVE
720C 21 6D 73    3980        LD   HL,TO
720F CD 00 02    3990        CALL TVE
7212 06 2F       4000        LD   B,47D
7214 CD 81 70    4010        CALL LINE
7217             4020 *
7217             4030 * GET PARAMETERS
7217             4040 *
7217 CD 87 70    4050        CALL GETAB      GET START
721A 13          4060        INC  DE           AND END
721B D5          4070        PUSH DE           ADDRESSES
721C E5          4080        PUSH HL           FOR BLOCK
721D D1          4090        POP  DE         CALCULATE SIZE
721E E1          4100        POP  HL           OF BLOCK
721F ED 52       4110        SBC  HL,DE
7221 D5          4120        PUSH DE
7222 E5          4130        PUSH HL
7223 06 38       4140        LD   B,56D
7225 CD 81 70    4150        CALL LINE
7228 3E C3       4160        LD   A,'C'      PROMPT FOR
722A CD 74 70    4170        CALL QUEST        DESTINATION
722D CD A6 04    4180        CALL NUMIN        ADDRESS
7230 57          4190        LD   D,A
7231 CD A6 04    4200        CALL NUMIN
7234 5F          4210        LD   E,A
7235 C1          4220        POP  BC         POP BLOCK SIZE
7236 E1          4230        POP  HL           & START ADDR
7237             4240 *
7237             4250 * MOVE THE BLOCK
7237             4260 *
7237 ED B0       4270        LDIR
7239             4280 *
7239             4290 * PRINT END ADDRESS
7239             4300 * OF MOVED BLOCK
7239             4310 *
7239 06 38       4320        LD   B,56D
723B CD 81 70    4330        CALL LINE
723E 21 94 73    4340        LD   HL,ND
7241 CD 00 02    4350        CALL TVE
7244 1B          4360        DEC  DE
7245 D5          4370        PUSH DE
7246 5A          4380        LD   E,D
7247 CD 46 02    4390        CALL NUMOUT
724A D1          4400        POP  DE
724B CD 46 02    4410        CALL NUMOUT
724E CD AE 70    4420        CALL KEYPR      WAIT FOR KEYPRESS
7251 28 AA       4430        JR   Z,MOVE     IF SPACE BAR,
7253             4440 *                       MOVE AGAIN
7253 C3 00 70    4450        JP   START      ELSE BACK TO MENU
7256             4460 * * * * * * * * * * * * * * * * *
7256             4470 * ROUTINE TO COMPARE TWO
7256             4480 * BLOCKS OF MEMORY AND
7256             4490 * DISPLAY MISMATCHES
7256             4500 *
7256             4510 *
7256             4520 * INITIALIZATION
7256             4530 *
7256 01 00 00    4540 COMPAR LD   BC,0       SET NOT-FOUND FLAG
7259 C5          4550        PUSH BC           & PAGING COUNTER
725A CD E6 00    4560        CALL ERASE      CLEAR SCREEN AND
725D 21 3D 73    4570        LD   HL,CMPR      PRINT BANNER
7260 CD 00 02    4580        CALL TVE
7263 21 53 73    4590        LD   HL,AB
7266 CD 00 02    4600        CALL TVE
7269 21 6D 73    4610        LD   HL,TO
726C CD 00 02    4620        CALL TVE
726F 06 2B       4630        LD   B,43D
7271 CD 81 70    4640        CALL LINE
7274             4650 *
7274             4660 * GET PARAMETERS
7274             4670 *
7274 CD 87 70    4680        CALL GETAB      GET START
7277 06 38       4690        LD   B,56D        & END ADDR
7279 CD 81 70    4700        CALL LINE         OF BLOCK 1
727C 3E C3       4710        LD   A,'C'
727E CD 74 70    4720        CALL QUEST      GET START
7281 CD A6 04    4730        CALL NUMIN        ADDR OF
7284 F5          4740        PUSH AF           BLOCK 2
7285 CD A6 04    4750        CALL NUMIN        INTO IY
7288 C1          4760        POP  BC
7289 4F          4770        LD   C,A
728A C5          4780        PUSH BC
728B FD E1       4790        POP  IY
728D             4800 *
728D             4810 * COMPARE BYTE N FROM BLOCK 1
728D             4820 * WITH BYTE N FROM BLOCK 2; IF
728D             4830 * NOT IDENTICAL, PRINT ADDRESSES
728D             4840 * AND PRINT EACH BYTE AS HEX OR
728D             4850 * OCTAL AND AS ASCII
728D             4860 *
728D 7E          4870 LOOK   LD   A,(HL)
728E FD BE 00    4880        CP   (IY+0)     ARE BYTES EQUAL?
7291 20 0D       4890        JR   NZ,FOUND
7293 23          4900 NEX00  INC  HL         YES, BUMP POINTERS
7294 FD 23       4910        INC  IY
7296 D5          4920        PUSH DE         SUBTRACT BLOCK 1'S
7297 EB          4930        EX   DE,HL        POINTER FROM
7298 ED 52       4940        SBC  HL,DE        BLOCK 1 END ADDR
729A EB          4950        EX   DE,HL
729B D1          4960        POP  DE
729C 38 2E       4970        JR   C,ENDALL   IF PTR => END, DONE
729E 18 ED       4980        JR   LOOK       OTHERWISE CONTINUE
72A0             4990 *
72A0             5000 * SUBR TO PRINT
72A0             5010 * BYTES THAT DON'T MATCH
72A0             5020 *
72A0 C1          5030 FOUND  POP  BC         POP PAGING COUNTER
72A1 CB 41       5040        BIT  0,C        IF BIT 0,C IS ZERO,
72A3             5050 *                       THIS IS THE FIRST
72A3             5060 *                       MISMATCH FOUND
72A3 20 06       5070        JR   NZ,ONGO
72A5 CD E6 00    5080        CALL ERASE      SO WE CLEAR SCREEN,
72A8 01 01 0F    5090        LD   BC,0F01H     SET PAGING COUNTER
72AB             5100 *                       TO 15D & SET FLAG IN
72AB             5110 *                       C TO SHOW THAT THE
72AB             5120 *                       2 BLOCKS ARE NOT
72AB             5130 *                       IDENTICAL
72AB C5          5140 ONGO   PUSH BC
72AC E5          5150        PUSH HL
72AD D5          5160        PUSH DE
72AE CD E0 72    5170        CALL PRINT      PRINT BYTE & ADDR
72B1             5180 *                       FROM BLOCK 1
72B1 FD E5       5190        PUSH IY
72B3 E1          5200        POP  HL
72B4 CD E0 72    5210        CALL PRINT      THEN FROM BLOCK 2
72B7 06 20       5220        LD   B,32D
72B9 CD 81 70    5230        CALL LINE
72BC D1          5240        POP  DE
72BD E1          5250        POP  HL
72BE C1          5260        POP  BC
72BF 10 08       5270        DJNZ NEX3@3     IF PAGING COUNTER
72C1             5280 *                       IS 0, PAGE IS FULL
72C1 CD A8 01    5290        CALL KEYBD      WAIT FOR KEYPRESS
72C4 CD E6 00    5300        CALL ERASE
72C7 06 0F       5310        LD   B,15D      RELOAD PAGING COUNTER
72C9 C5          5320 NEX3@3 PUSH BC           & SAVE ON STACK
72CA 18 C7       5330        JR   NEX00
72CC             5340 *
72CC             5350 * WRAP-UP
72CC             5360 *
72CC C1          5370 ENDALL POP  BC         WERE ANY MISMATCHES
72CD CB 41       5380        BIT  0,C          FOUND?
72CF 20 06       5390        JR   NZ,FINI
72D1 21 97 73    5400        LD   HL,IDENT   YES, PRINT "SAME"
72D4 CD 00 02    5410        CALL TVE
72D7 CD AE 70    5420 FINI   CALL KEYPR      WAIT FOR KEYPRESS
72DA CA 56 72    5430        JP   Z,COMPAR   IF SPACE BAR,
72DD             5440 *                       RERUN COMPARE
72DD C3 00 70    5450        JP   START      ELSE BACK TO MENU
72E0             5460 *
72E0             5470 * ROUTINE TO PRINT ONE
72E0             5480 * BYTE, WITH ASCII EQUIVALENT
```

```
72E0                5490 * AND CURRENT POINTER ADDRESS
72E0                5500 *
72E0 5C             5510 PRINT  LD   E,H         PRINT ADDRESS
72E1 CD 46 02       5520        CALL NUMOUT
72E4 5D             5530        LD   E,L
72E5 CD 46 02       5540        CALL NUMOUT
72E8 3E BD          5550        LD   A,'='
72EA CD 79 70       5560        CALL TV
72ED 5E             5570        LD   E,(HL)     PRINT BYTE IN
72EE CD 46 02       5580        CALL NUMOUT       HEX OR OCTAL
72F1 3E BD          5590        LD   A,'='
72F3 CD 79 70       5600        CALL TV
72F6 7E             5610        LD   A,(HL)
72F7 FE 80          5620        CP   80H        IS IT ASCII WITH
72F9 30 04          5630        JR   NC,DGI       HIGH BIT ON?
72FB 06 A2          5640        LD   B,0A2H     NO, PRINT "
72FD 18 04          5650        JR   ASCII
72FF 06 A7          5660 DGI    LD   B,0A7H     YES, PRINT '
7301 18 00          5670        JR   NEX2T
7303 C6 80          5680 ASCII  ADD  80H        SET HIGH BIT
7305 FE 80          5690        CP   80H        NOW IS IT A
7307 30 02          5700        JR   NC,NEX2T     PRINTABLE CHAR?
7309 18 04          5710        JR   NXT220     NO, PRINT SPACE
730B FE FF          5720 NEX2T  CP   0FFH        YES; IS IT RUB?
730D                5730 *                        (RUBOUT WOULD
730D                5740 *                        CLEAR SCREEN)
730D 20 08          5750        JR   NZ,NEX45   NO, YOU CAN PRINT IT
730F CD 6F 70       5760 NXT220 CALL SPACE
7312 CD 6F 70       5770        CALL SPACE
7315 18 09          5780        JR   IND
7317 F5             5790 NEX45  PUSH AF
7318 78             5800        LD   A,B        PRINT " OR '
7319 CD 79 70       5810        CALL TV
731C F1             5820        POP  AF         PRINT BYTE IN ASCII
731D CD 79 70       5830        CALL TV
7320 06 04          5840 IND    LD   B,4
7322 CD 81 70       5850        CALL LINE
7325 C9             5860        RET
7326                5870 * * * * * * * * * * * * * * * * *
7326                5880 * ROUTINE TO SET
7326                5890 * HEX MODE
7326                5900 *
7326 3E C8          5910 HEXMOD LD   A,'H'      SEE NOTE #3
7328 32 A7 01       5920        LD   (0167H),A
732B C3 08 70       5930        JP   START2
732E                5940 *
732E                5950 * TEXT FOR
732E                5960 * BANNERS
732E                5970 *
732E                5980 MON    DC   'MONITOR'
  CD   CF   CE   C9   D4
  CF   D2
7335                5990        DB   0
  00
7336                6000 SRCH   DC   'SEARCH'
  D3   C5   C1   D2   C3
  C8
733C                6010        DB   0
  00
733D                6020 CMPR   DC   'COMPARE'
  C3   CF   CD   D0   C1
  D2   C5
7344                6030        DB   0
  00
7345                6040 MV     DC   'MOVE'
  CD   CF   D6   C5
7349                6050        DB   0
  00
734A                6060 HX     DC   'HEX MODE'
  C8   C5   D8   A0   CD
  CF   C4   C5
7352                6070        DB   0
  00
7353                6080 AB     DC   ' A - B '
  A0   C1   A0   AD   A0
  C2   A0   A0
7360                6090        DB   0
  00
```

```
735C                6100 FOR    DC   'FOR '
  C6   CF   D2   A0
7360                6110        DB   0
  00
7361                6120 STRNG  DC   'STRING'
  D3   D4   D2   C9   CE
  C7
7367                6130        DB   0
  00
7368                6140 NON    DC   'NONE'
  CE   CF   CE   C5
736C                6150        DB   0
  00
736D                6160 TO     DC   'TO C - D'
  D4   CF   A0   C3   A0
  AD   A0   C4
7375                6170        DB   0
  00
7376                6180 UTIL   DB   0FFH
  FF
7377                6190        DB   24D
  18
7378                6200        DC   'TVC-64 UTILITIES'
  D4   D6   C3   AD   B6
  B4   A0   D5   D4   C9
  CC   C9   D4   C9   C5
  D3
7388                6210        DB   10D
  65
7389                6220        DC   '@ MONITOR'
  B8   A0   CD   CF   CE
  C9   D4   CF   D2
7392                6230        DB   60D
  3C
7393                6240        DB   0
  00
7394                6250 ND     DC   '!D?'
  C4   BF
7396                6260        DB   0
  00
7397                6270 IDENT  DC   ' SAME'
  A0   D3   C1   CD   C5
739C                6280        DB   0
  00
739D                6290 *
739D                6300 * INPUT BUFFER
739D                6310 *
739D                6320        DS   8D
73A5                6330 INBUF  DS   1

NO ERRORS FOUND

FILE  3000  67E8
READY
LTABL
AB      7353   ASCII   7303   BAXUP   71D3   BCZERO 718F
BIT0    713E   CMPR    733D   COMPAR  7256   DGI    72FF
ENDALL  72CC   ENDPUT  7163   ERASE   00E6   FIN    71F3
FINI    72D7   FINISH  04A9   FOR     735C   FOUND  72A0
GETAB   7087   GO3     71C1   HEX     7153   HEXMOD 7326
HX      734A   IDENT   7397   INMED   7062   INBUF  73A5
IND     7320   INPUT   70FC   INPUT0  7049   KEYBD  01A8
KEYPR   70AE   LINE    7081   LOAD    7158   LOOK   729D
LOOK2   7183   MASK    71FC   MON     732E   MONTR  0500
MOVE    71FD   MV      7345   ND      7394   NEX00  7293
NEX2T   7308   NEX303  72C9   NEX45   7317   NEXT5  7158
NEXT6   7179   NEXT7   7196   NEXT8   7102   NON    7368
NONE    71ED   NOSKIP  7188   NOTLIT  7112   NTWILD 7155
NUMBER  7123   NUMIN   04A6   NUMOUT  0246   NUMS   70A7
NXT220  730F   ONGO    72AB   PRINT   72E0   PRINT2 71BE
QUEST   7074   QUEST1  7077   ROTATE  71A4   SEARCH 7084
SETLIT  711C   SHORT   716F   SPACE   706F   SPC    707A
SRCH    7336   START   7000   START2  7008   STRNG  7361
TABLE   7065   TEST    71AA   TO      736D   TV     7079
TVE     0200   UTIL    7376
FILE  3000  67E8
READY
```

PILOT ON NORTHSTAR DISK

Dear Dr. Dobb: *Received: 78 June 9*

Through the kind cooperation of the author, Dr. John Starkweather, the source code for 8080 PILOT which you published in your *Journal* last year is now available in machine readable form from the North Star Software Exchange.

Anyone interested may order diskette NSSE5 from:

North Star Software Exchange
2547 Ninth Street
Berkeley, CA 94710

The price is $9.50 including the cost of the diskette. The diskette also includes a remarks file and an executable file containing a preliminary version of PILOT modified to run with the North Star DOS. This file also includes the standard PILOT demonstration program.

Sincerely,
Thos Sumner

THE 6502 PROGRAM EXCHANGE

News Release *Received: 78 June 13*

Apple II users can now run an extended version of the high-level language FOCAL (DEC trademark). FCL65E, as it is called, is available on an Apple II format cassette ($25) and occupies 1000H-2665H in memory. A Mini-Manual ($6) provides I/O information and descriptions of FCL65E's commands. The FCL65E User's Manual ($12) contains 104 pages of programming examples to help you exploit the power of the language. The complete source listing in cross-assembly form ($35) will enable the enthusiast to customize FCL65E for his particular applications. KIM and TIM cross-assemblies, paper tapes, and KIM Hypertape cassettes are also available. For further details and information on other available 6502 software, send $1.00 to The 6502 Program Exchange, 2920 Moana, Reno, NV 89509.

A Shorter Printer Driver for PolyMorphic's BASIC

BY RICHARD B. EMERSON
158 E. Butler Ave.
Ambler, PA 19002
COPYRIGHT © 1978

I am the proud owner of a PolyMorphic Systems Poly 88 with an 8080A CPU, Video Terminal Interface (VTI), 20K of memory from various vendors (Godbout, IMS, and IMSAI in order of preference), A Poly Cassette Interface, and a Poly Printer Interface. It is this last item that led me to write this article.

Sooner or later every computer owner wants a permanent copy of his triumphs (the very non-permanent CRT is fine for the less memorable events!) and at the same time it becomes painfully clear that a hard copy device is not cheap. Most printers that warrant any serious consideration are priced at or above the cost of a CPU and cabinet. I am fortunate enough to have access to a TI-745 portable terminal. This fact removed the first obstacle to getting a printer running. The second problem is the matter of finding the electronics to drive the printer. Unfortunately, most printers lack the good grace to allow you to hook your printer up to your particular bus, be it S-100, SWTPC, Benton Harbor, or Greyhound. This is where the manufacturers of parallel to serial and serial to parallel interfaces come in. They make boards that put the right signals in the right places so that your printer can be "heard". My first thought was to buy a printer interface from PolyMorphic but a shortage caused by problems with the printed circuit manufacturer meant that the interfaces were as scarce as programs that run perfectly the first time. I decided to look elsewhere.

I didn't have too much success finding replacements because the Poly 88 CPU clock runs about 9% slower than most 8080 systems. This is so that the VTI can use the system clock to generate the needed sync signals for an essentially flicker free display. When I tried to use a Franklin Electric 3S+P I/O board the baud rates came out all wrong even at rates as low as 110 baud. The card is capable of running as fast as 9600 baud and every rate was off by a significant amount. If you do have a need for three serial ports and if you have a 2 MHz clock, the Franklin Electric board at $150 is a bargain that is hard to beat. Actually, even had the board been compatible with my Poly 88, I would still have been faced with the problem of software. Poly's printer interface uses a USART with connections to ports 0 and 1. Because the cassette interface also uses the same ports, just changing the IN and OUT instructions to new ports would not have the desired results even if one assumes that just changing ports would work at all. This is problem number three; without the software to drive the printer interface, all you will have is an impressive looking drain on your budget.

While standing around waiting for my printer interface kit to be delivered I reviewed what I needed to drive my printer. The assembler that I had just bought (PolyMorphic's Assembler version G02) has a driver already written into it. My BASIC doesn't have a driver in it but Poly does supply an auxiliary program that will drive a printer connected to their printer interface card. It seemed, then, that I was spared having to write anything to make my printer work.

My waiting was rewarded by a phone call from the good folks at Personal Computer Corporation in Fraser, PA. My interface had arrived! Before leaving Personal Computer I decided to check the parts list against the parts that were actually in the box as there weren't too many parts to check. It was a good thing that I checked because ALL of the parts for driving a current loop device, such as a Teletype, were missing. Personal Computer offered to replace the parts with some from another kit in the shipment, but the parts were missing from that kit, too! In my case the loss is annoying more than anything else, but for $65 I think that PolyMorphic could have done better.

The kit went together with only one or two bits of excitement. The first thrill was finding out that what I thought to be a 1N5252 diode wasn't. Thinking nasty thoughts about kit packers, I double-checked the manual to see if an alternate part number was listed somewhere. Sure enough, four pages later, buried in the instructions was the other number, 1N5254A. The second thrill was finding that the resistors R2 and R5 were not mentioned because they seem to have been dropped from the design (I like to see sequential numbers when it comes to parts call-outs.). A note saying that they had been dropped would have saved me much page flipping. By following the rather crude sketches on page nine of the manual, I set up the jumpers needed to test the card. The next step was a smoke test so I pulled all of the cards in the mainframe except the CPU which is where the interface is connected. This may seem a bit extreme, pulling all of the cards out, but it is cheaper than having a short or wiring error smoke more than just the new card.

Having passed the smoke test properly, I ran the test program which echos data at 9600 baud. This ran the first time off, a very pleasant surprise. All that remained was to hook up the TI-745 and try it out. The owner's manual from Texas Instruments told me what pins did what on the terminal's connector and so it was easy to connect up a cable. Well, it was almost easy, the control leads took a little tinkering. The TI-745 requires that the Data Carrier Detect (DCD) line be pulled high before the terminal will run. By connecting the DCD line to the terminal's own Data Terminal Ready (DTR) line, the terminal tricks itself into running. I did not want to put the jumper between the DCD line and the DTR line on the TI-745's connector so I borrowed pins 5 and 6 of J1 on the interface card. Ordinarily, pin 8, the DCD line, does not appear on the jumper block, J1. However, by tying the pin 8 side of the CF jumper near J2 to the pin 5 side of the CC jumper, it is possible to make the DCD line appear on J1. All that remains is to tie pins 5 and 6 together at J1 and to remove the jumper that PolyMorphic recommends go between pins 9 and 10. While the steps discussed above are fine for a TI-745, other printers may need different connections. The printer interface manual suggests connections for Teletypes, Decwriters (available from Heathkit at a fair price), and Hytypers.

My first attempt to use the printer with the assembler was a great success. I was also able to use BPRINT, the BASIC printer driver, to list off programs written in BASIC and also to copy the program's output from the screen. I soon found that BPRINT has one major flaw. It takes up over 500 bytes, and, since I'm cheap, I'd rather put program code in those bytes than a space-eating buffer. The majority of the space is consumed by a 256 byte circular buffer while the code resides in the remaining space. The obvious question was, could the buffer be cut down without hurting performance and the answer is yes (Why else would I write this article?).

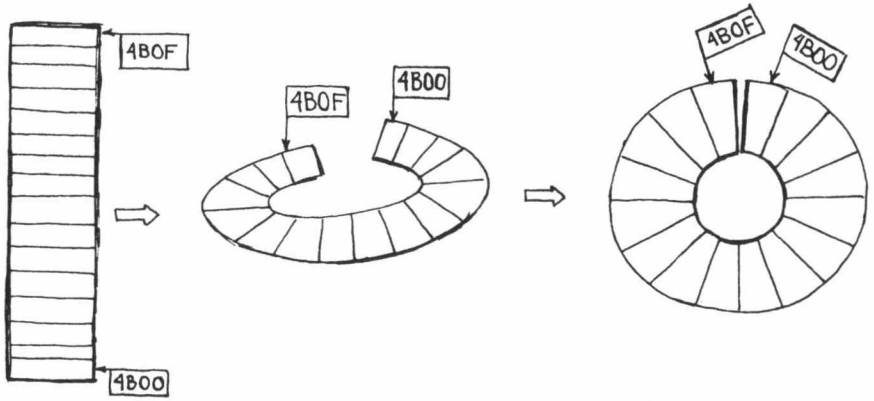
The accompanying listing of BPRINT retains the essential features of PolyMorphic's BPRINT. It switches on the printer with CTRL/Q and the keyboard is switched on with CTRL/R. CTRL/S and CTRL/T turn off the printer and keyboard respectively. The restart address is not the same as the one that Poly specifies. Use 4A8D instead. The baud rate table on page five of Poly's writeup for BPRINT still applies for the most part, the only change is that the constant for the baud rate is now at 4A42. If you want to change the number of pad characters from the current value of 10, put the new value into 4A7C but be sure that it is not less than 1 or evil things will happen. Sharp-eyed readers will note that the buffer extends from 4B0F down to 4B00 while ERRMSG extends from 4ACD up to 4B03. This is not a real problem because either BPRINT loads, in which case ERRMSG is excess baggage, or BPRINT doesn't load, in which case the buffer isn't likely to be needed. If you are still bothered by this, all that is needed is to shorten ERRMSG and be sure to end it with a zero byte. For those who understand circular buffers, that's it. There is only one final caution. Sometimes the printer will start printing one character over and over when restarting with the CTRL/Q. Don't panic, just type another CTRL/Q and hit return. The USART is being asked to drive two devices and it appears that noise on the CPU card can sometimes cause the USART to start sending nonsense. In the writeup that comes with PolyMorphic's BPRINT, they insist that you must restart BASIC through the point labeled START (4A8D on this version). In fact it appears that 99% of the time you can restart through 2003 as always. This has been tried in both versions of BPRINT with good success. The only "gotcha" seems to be that WH1 (0C24) and SRA4 (0C16) must not be modified. It is, in fact, possible to type CTRL/Q, CTRL/R, return, and then record type-ins made after typing CTRL/Z. The screen's display of the registers and so on do not come through WH1 so they are not printed, but any type-ins made will appear on the screen as always and on the printer, too.

One thing needs to be made quite clear about circular or ring buffers. The name describes only the concept; they aren't really circular. But it makes it easier to visualize how the buffer functions. What happens is that a range of addresses is selected such that the highest and lowest addresses seem to be adjacent. The accompanying sketches show how this feat of memory topology is possible. In this program, the lowest two digits of the pointers' addresses meet this requirement. If you look at the lines after the labels WRITE and SEND you will see that the L register is decremented and tested for zero. In this case L will be restored to 0F making 4B00 seem adjacent to 4B0F. A buffer that is 256 bytes long would be restored to FF automatically by the DCR L instruction. The HL register pair may contain either BPP (Buffer Put Pointer) or BGP (Buffer Get Pointer) but for the purposes of this discussion all that matters is understanding why address 4B00 can be thought of as being adjacent to 4B0F. Study the sketches of how the pointers move. At the start of the program both BPP and BGP point to the same place. As data comes in BPP moves ahead of BGP, and BPP will continue to advance until it is again equal to BGP and the buffer is full. Any time a character is read from the buffer, BGP advances and it may continue to advance until it is equal to BPP and the buffer is empty. In the second example the buffer has started to fill with the string "THE QUICK BROWN FOX", however, the character "3" is left over from something else that was in the buffer. BPP is loading an "N" and BGP is getting a "T". The one thing that must not happen is to have the "get" pointer move ahead of the "put" pointer. If this should happen you might wind up getting back characters that haven't been put in the buffer yet! Actually old characters would come out but in any case the results would not be valid.

The example of the buffer is one character away from being full. Since BPP may not pass BGP or "wrap-around", what can be done? The easiest way out is to stop BASIC from producing too many characters for the buffer. WAIT does this by making sure that the interrupts have been enabled and then halting the CPU. When the USART asks for the next character it generates an interrupt, forcing the CPU to start running at routine TISR (Terminal Interrupt Service Routine). There is a chance that something else generated the interrupt (The console keyboard and real-time clock both use interrupts and you may have added devices that also generate interrupts).) so the first thing that is done is to verify that the USART did, in fact, cause the interrupt. If it did, READ or WRITE is executed, depending on the USART's status buffer, otherwise the CPU is sent off through IORET, the monitor's return point for all interrupt service routines, to see what happened.

There you have it, circular buffers are in fact handy bits of software that allow you to tie together a fast CPU and a slow printer without resorting to magic incantations or lots of expensive chips. As you can see, circular buffers are quite easy to understand once you, ah, get a-round to it . . .

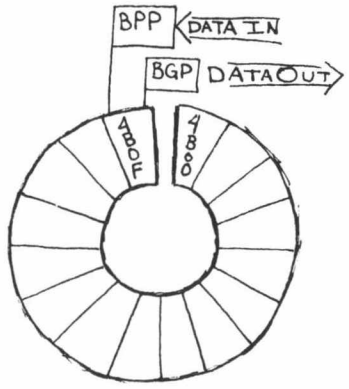
```
                ;BASIC PRINTER DRIVER VER 3.1 - APRIL, 1978
                ;COPYRIGHT (C) 1978, RICHARD B. EMERSON * ALL RIGHTS RESERVED
                ;
                ;BASIC REFERENCES AND ENTRY POINTS
        2000    BGO     EQU     2000H   ;START OF BASIC
        200E    BTXT    EQU     200EH   ;BASIC BANNER
        205D    BMLL    EQU     205DH   ;BASIC MEM. LOWER LIMIT
        206B    BKD     EQU     206BH   ;BASIC KYBD DRIVER
                ;MONITOR ENTRY POINTS
        02AD    SETUP   EQU     2ADH    ;USART SETUP CODE IN MONITOR
        0064    IORET   EQU     64H     ;STD. SERVICE ROUTINE RETURN CODE
        0C16    SRA4    EQU     0C16H   ;USART INTERRUPT SERVICE ROUTINE ADD.
        0C24    WH1     EQU     0C24H   ;WORMHOLE FOR SCREEN O/P
                ;MISC. CONSTANTS
        007F    PAD     EQU     7FH     ;PAD CHAR.
        000D    CR      EQU     0DH     ;CARRIAGE RETURN
        000A    LF      EQU     0AH     ;LINE FEED
        0011    XON     EQU     11H     ;PRTR ON
        0013    XOFF    EQU     13H     ;PRTR OFF
        0012    DC2     EQU     12H     ;KYBD ON
        0014    DC4     EQU     14H     ;KYBD OFF
                ;THE CIRCULAR BUFFER AND NEW LIMIT FOR BASIC
        4B00    BUFR    EQU     4B00H   ;BUFFER MUST END ON LO BYTE = 00
        4B0F    MEND    EQU     BUFR+0FH
49B1                    ORG     49B1H   ;START AT BASIC'S LAST WORD ADD. + 1
                ;TERMINAL INTERRUPT SERVICE ROUTINES
                ;ENTRY TO THESE ROUTINES IS VIA SRA4 AND AN INTERRUPT FROM THE
                ;USART ON THE CPU CARD. TISR'S ADDRESS IS STORED IN SRA4 WHEN
                ;XON IS EXECUTED.  CHANGING SRA4 AND WH1 WILL REQUIRE RELOAD-
                ;ING AT START.
49B1 DB01       TISR    IN      1       ;GET USART STATUS
49B3 1F                 RAR             ;DID USART REQUEST DATA?
49B4 DACC49             JC      WRITE   ;YES, GO SEND IT
49B7 1F                 RAR             ;DOES USART HAVE DATA?
49B8 D26400             JNC     IORET   ;NO, FALSE ALARM
                ;USART READ ROUTINE
                ;READ DATA FROM FOREIGN KEYBOARD AND GIVE IT TO BASIC
49BB 3A894A             LDA     IFLG    ;YES, USART HAS DATA FROM TTY
49BE B7                 ORA     A       ;IS I/P FROM TTY ALLOWED?
49BF CAC749             JZ      FLUSH   ;NO, FLUSH DATA
49C2 DB00               IN      0       ;YES, READ DATA
49C4 C36B20             JMP     BKD     ;GIVE IT TO BASIC
49C7 DB00       FLUSH   IN      0       ;READ DATA JUST TO FLUSH USART
49C9 C36400             JMP     IORET   ;ALL DONE
                ;USART WRITE ROUTINE
                ;WRITE DATA FROM BASIC ONTO FOREIGN PRINTER
49CC 2A874A     WRITE   LHLD    BGP     ;SET HL TO NEXT CHAR TO BE WRITTEN
49CF 3A854A             LDA     BPP     ;GET LAST CHAR PUT INTO BUFFER INTO ACC
49D2 BD                 CMP     L       ;CIRCULAR BUFFER EMPTY?
49D3 CAE549             JZ      VAMP    ;YES,NO DATA SO SEND DUMMY CHAR
49D6 7E                 MOV     A,M     ;NO, GET OLDEST CHAR FROM BUFFER
49D7 2D                 DCR     L       ;BGP=BGP-1
49D8 C2DD49             JNZ     WRITE1  ;DID BUFFER WRAP AROUND? NO, SEND CHAR
49DB 2E0F               MVI     L,0FH   ;SET BGP TO 0FH, TOP OF BUFFER
49DD 22874A     WRITE1  SHLD    BGP     ;SAVE NEW BGP
49E0 D300               OUT     0       ;SEND CHAR TO USART
49E2 C36400             JMP     IORET   ;ALL DONE
49E5 3E0D       VAMP    MVI     A,PAD   ;SEND DUMMY CHAR TO PRTR
49E7 C3DD49             JMP     WRITE1
                ;BASIC TO TERMINAL ROUTINES
                ;ENTRY FROM BASIC TO BPRINT
49EA E5         BENT    PUSH    H       ;SAVE REG.S FOR RETURN TO BASIC
49EB F5                 PUSH    PSW
49EC FE11               CPI     XON     ;DOES ACC. HAVE XON?
```



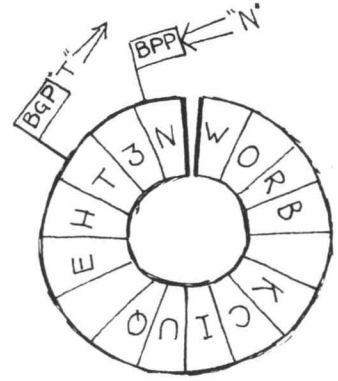
IF YOU TAKE THIS PIECE OF MEMORY AND TWIST IT SO...

...IT BECOMES A SPIRAL LIKE THIS...

... AND LOOK AT IT FROM "ABOVE", YOU WILL SEE A CIRCULAR BUFFER



THE BUFFER AND POINTERS AT START-UP



THE BUFFER WHILE HANDLING THE STRING, "THE QUICK BROWN"

```
49EE CA334A             JZ      XONR    ;YES, ENABLE PRTR
49F1 FE13               CPI     XOFF    ;IS IT AN XOFF?
49F3 CA594A             JZ      XOFFR   ;YES, DISABLE PRTR
49F6 FE12               CPI     DC2     ;IS IT A DC2?
49F8 CA614A             JZ      DC2R    ;YES, ENABLE KYBD
49FB FE14               CPI     DC4     ;HOW ABOUT DC4?
49FD CA604A             JZ      DC4R    ;YES, DISABLE KYBD
4A00 3A8A4A             LDA     OFLG    ;GET OUTPUT FLAG
4A03 B7                 ORA     A       ;IS TTY O/P ALLOWED?
4A04 CA724A             JZ      BEXIT   ;OFLG=0, TTY O/P NOT ALLOWED
4A07 F1                 POP     PSW     ;ACC. HAS BEEN MODIFIED BY <ORA A>
4A08 F5                 PUSH    PSW     ;RESTORE IT AND RESAVE IT ON STACK
4A09 FE0D               CPI     CR      ;IS THIS A <CR>?
4A0B CA674A             JZ      CROUT   ;YES, SEND IT
4A0E CD174A             CALL    SEND    ;ITS JUST PLAIN OLD TEXT
4A11 C3724A             JMP     BEXIT   ;ALL DONE
                ;SEND DATA FROM BASIC TO TTY
4A14 FB         WAIT    EI              ;ALLOW USART TO SEND INT.
4A15 76                 HLT             ;HALT CPU UNTIL USART INT.S FOR CHAR
4A16 F1                 POP     PSW     ;RESTORE ACC. WITH NEW CHAR IN IT
4A17 F5         SEND    PUSH    PSW     ;SAVE CHAR, PUT IT ON STACK
4A18 F3                 DI              ;KEEP USART FROM INTERRUPTING
4A19 2A854A             LHLD    BPP     ;FIND OUT WHERE TO PUT CHAR IN
4A1C 77                 MOV     M,A     ;TO BUFFER AND DEPOSIT IT
4A1D 3A874A             LDA     BGP     ;GET NEXT CHAR TO COME FROM BUFFER
4A20 2D                 DCR     L       ;BPP=BPP-1
4A21 C2264A             JNZ     SEND1   ;IF BUFFER NOT FULL GET OLDEST CHAR
4A24 2E0F               MVI     L,0FH   ;ELSE RESET BPP TO 0FH
4A26 3A874A     SEND1   LDA     BGP     ;GET OLDEST CHAR
4A29 BD                 CMP     L       ;WAS BUFFER FULL?
4A2A CA144A             JZ      WAIT    ;YES, WAIT UNTIL USART EMPTIES IT
4A2D 22854A             SHLD    BPP     ;STORE WHERE TO PUT NEXT CHAR
4A30 F1                 POP     PSW     ;RESTORE ACC. WITH CHAR IN IT
4A31 FB                 EI              ;LET USART INT.
4A32 C9                 RET             ;ALL DONE
                ;ENABLE PRTR
4A33 F3         XONR    DI              ;KEEP INT.S OUT
4A34 D5                 PUSH    D       ;SAVE REG.S FOR RETURN TO BASIC
4A35 C5                 PUSH    B
4A36 21B149             LXI     H,TISR  ;GET USART INT. SVC. RTN. ADD.
4A39 22160C             SHLD    SRA4    ;STORE IT MONITOR'S SRA TABLE
4A3C 328A4A             STA     OFLG    ;NON-ZERO OFLG SAYS TTY O/P IS ALLOWED
4A3F CDAD02             CALL    SETUP   ;SET BAUD RATE, ETC.
4A42 16AA40DA           DB      16H,0AAH,40H,0DAH,0     ;DATA FOR SETUP RTN
4A46 00
4A47 C1                 POP     B       ;RESTORE REG.S
4A48 D1                 POP     D
4A49 210F4B             LXI     H,BUFR+0FH      ;INITIALIZE BUFFER POINTERS
4A4C 22854A             SHLD    BPP
4A4F 22874A             SHLD    BGP
4A52 3E27               MVI     A,27H   ;ENABLE USART THRU STATUS PORT
4A54 D301               OUT     1
4A56 C3724A             JMP     BEXIT   ;ALL DONE
                ;DISABLE PRTR
4A59 AF         XOFFR   XRA     A       ;RESET OFLG TO ZERO,TTY O/P NOT ALLOWED
4A5A 328A4A             STA     OFLG
4A5D C3724A             JMP     BEXIT   ;ALL DONE
                ;DISABLE KYBD
4A60 AF         DC4R    XRA     A       ;RESET IFLG TO ZERO,TTY I/P NOT ALLOWED
4A61 32894A     DC2R    STA     IFLG    ;IF IFLG NOT ZERO,TTY I/P IS ALLOWED
4A64 C3724A             JMP     BEXIT   ;ALL DONE
                ;SEND <CR> AND FILLER WHILE PRINTER HEAD RETURNS TO LEFT MARGIN
4A67 CD174A     CROUT   CALL    SEND    ;SEND <CR> TO TTY
4A6A CD794A             CALL    FILL    ;FIDDLE WHILE HEAD RETURNS
4A6D 3E0A               MVI     A,LF    ;SEND <LF> TO TTY
4A6F CD174A             CALL    SEND
               ;COMMON EXIT BACK TO BASIC - BY RESTORING WH1'S NORMAL CONTENTS
                ; WH1 ALSO DISPLAYS DATA ON THE SCREEN WHILE PRINTING
4A72 F1         BEXIT   POP     PSW     ;RESTORE ACC.
4A73 2A8B4A             LHLD    WRMSTR  ;GET OLD WORMHOLE CONTENTS
4A76 E3                 XTHL            ;REPLACE THEM ON STACK
4A77 FB                 EI              ;ENABLE INT.S
4A78 C9                 RET             ;ALL DONE

                ;FILL SENDS DUMMY DATA TO KEEP PRTR BUSY
4A79 3E00       FILL    MVI     A,PAD   ;GET DUMMY DATA
4A7B 060A               MVI     B,10    ;COUNT BACK FROM 10
4A7D CD174A     FILL1   CALL    SEND    ;SEND PADDING TO TTY
4A80 05                 DCR     B       ;IS B EMPTY YET?
4A81 C27D4A             JNZ     FILL1   ;NO, KEEP FILLING
4A84 C9                 RET             ;ALL DONE
                ;FLAGS AND POINTERS
4A85            BPP     DS      2       ;BUFFER PUT POINTER
4A87            BGP     DS      2       ;BUFFER GET POINTER
4A89            IFLG    DS      1       ;TTY INPUT FLAG
4A8A            OFLG    DS      1       ;TTY OUTPUT FLAG
4A8B            WRMSTR  DS      2       ;WORMHOLE STORAGE FOR OLD WH1 ADDRESS
                ;HOOKUP CODE ATTACHES BPRINT TO BASIC
                ; N O T E :  START IS THE BEST RE-ENTRY POINT.  IF SRA4 AND WH1
                ; HAVE NOT BEEN MODIFIED, USE 2003.  RE-ENTRY THROUGH START IS
                ; GOING TO SCRATCH ANY EXISTING BASIC PROGRAMS.  IF A PROGRAM
                ; GLITCH REQUIRES A RESTART, TRY 2003 FIRST.
4A8D F3         START   DI              ;KEEP OUT INTERRUPTS
4A8E 210E20             LXI     H,BTXT  ;POINT TO BASIC'S BANNER
4A91 11CE4A             LXI     D,TEXT  ;POINT TO REFERENCE BANNER
4A94 0E17               MVI     C,23    ;CHECK 23 CHAR.S
4A96 1A         LOOP1   LDAX    D       ;GET REF. CHAR
4A97 BE                 CMP     M       ;DOES BASIC MATCH THIS?
4A98 C2BE4A             JNZ     ERROR   ;NO, GO BLOW UP!
4A9B 23                 INX     H       ;GET NEXT CHAR
4A9C 13                 INX     D
4A9D 0D                 DCR     C       ;ARE WE DONE?
4A9E C2964A             JNZ     LOOP1   ;NO, CHECK NEXT CHAR
4AA1 210F4B             LXI     H,MEND  ;STORE NEW END OF BASIC
4AA4 225D20             SHLD    BMLL    ;IN BASIC
4AA7 AF                 XRA     A       ;ACC. = 0
4AA8 32894A             STA     IFLG    ;RESET FLAGS
4AAB 328A4A             STA     OFLG
4AAE 2A250C             LHLD    WH1+1   ;TAKE OVER WORMHOLE1
4AB1 228B4A             SHLD    WRMSTR
4AB4 21EA49             LXI     H,BENT  ;PLACE ENTRY ROUTINE ADDRESS
4AB7 22250C             SHLD    WH1+1   ;IN WORMHOLE AND
4ABA FB                 EI              ;ALLOW INTERRUPTS
4ABB C30020             JMP     BGO     ;START UP BASIC
                ;ANNOUNCE THAT BASIC WON'T ACCEPT BPRINT
4ABE FB         ERROR   EI              ;ALLOW INT. (I.E. CTRL/Z)
4ABF 21E54A             LXI     H,ERRMSG        ;POINT TO MESSAGE
4AC2 7E         ERRORL  MOV     A,M     ;GET CHAR
4AC3 B7                 ORA     A       ;CHAR=0?
4AC4 CAC24A             JZ      ERRORL  ;YES, HANG SYSTEM
4AC7 CD240C             CALL    WH1     ;NO, DISPLAY CHAR
4ACA 23                 INX     H       ;POINT TO NEXT CHAR
4ACB C3C24A             JMP     ERRORL  ;NEXT CHAR, PLEASE
                ;THIS BANNER MUST MATCH BASIC'S BANNER
4ACE 506F6C79   TEXT    DB      'Poly 88 BASIC version A'
4AD2 20383820
4AD6 42415349
4ADA 43207665
4ADE 7273696F
4AE2 6E2041

                ;PLACE YOUR FAVORITE EXPLETIVE HERE!
4AE5 49206361   ERRMSG  DB      'I can"t hook up to this BASIC!',0
4AE9 6E227420
4AED 686F6F6B
4AF1 20757020
4AF5 746F2074
4AF9 68697320
4AFD 42415349
4B01 432100
```

Fast Cassette Interface for the 6502

BY CARL MOSER
3239 Linda Drive
Winston-Salem, NC 27106

The February 1977 issue of *Byte* (page 82) contained an interesting article on a minimum hardware cassette interface. I have used this technique to develop a cassette I/O to record and load on tape at over 1600 baud. Because I don't unpack bytes for recording, the effective data rate is over 160 bytes/second. The accompanying software listing for 6502 systems provides a record start sequence which requires at least ten 16 bytes followed with an OE byte to be inputted in sucession before loading can commence. At the end of loading, two byte checksum is used to detect errors. The hardware consists of a direct connection from a one-bit output port to the microphone input and a non-inverting hysterisis circuit incorporating an LM339 comparator as the playback electronics. Actually, I've used a direct connection for the playback with success but some cassette decks won't work unless the comparator is used. My General Electric and two Sanyo tape decks work very well without the comparator but the Realistic deck won't operate at all without the comparator.

An interesting note is that some tape decks put the signal on the barrel of the record and play jacks instead of on the inner tip. I've found that in my Realistic deck, the earphone output signal is on the barrel and the record input in on the inner tip. With both the record and play leads inserted, this creates an effective short.

To use this software, enter data in memory locations 0180-0184 as follows:

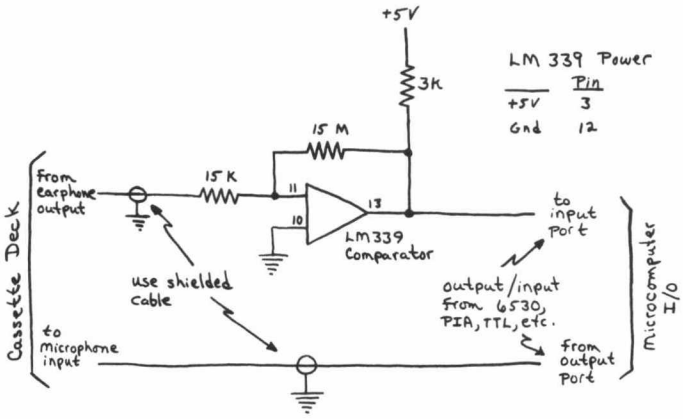
0180 = LOAD/NO
0181-0182 = Start Address
0183-0184 = End Address

The record routine will record from Start Address to End Address. LOAD/NO has no meaning to the record software.

The load routine will load from Start Address to End Address but store data in memory only if LOAD/NO does not equal zero. When equal 00, LOAD/NO can be used for verifying and conditionally selecting modules on tape.

The load and record routines have callable entry points at C/WRITE (3E00) and C/READ (3EA5), and non-callable entry points at LOAD.ENTRY (3F41) and RECORD.ENT (3F52). If C/READ is called, the Z-bit of the PSR will be true on return if no error was detected and false if errors occurred. If execution is at the non-callable entry LOAD.ENTRY, a break (via a BRK instruction) will be executed at end of loading and register A will indicate if the data was loaded correctly: R(A)=00 for good load, and EE for error.

To sum up, this has been a very reliable scheme and works error-free with the cheapest tapes (even Concert tapes which can be bought at many department stores at 3 for $1.00).



Cassette I/O Hardware

LISTING

```
ASSEMBLE 10
W:G

0010          .RA $3EAA
0020          .OS
0030 ;
0040 ;***** FAST CASSETTE INTERFACE *****
0050 ;
0060 ;
0070 ;== INPUT/OUTPUT ==
0080 C/PORT     .DE $6E00  CASSETTE I/O
0090 C/PORTD    .DE $6E01  PORT DIRECTION REG.
0100 ;BIT 2 = WRITE TO CASSETTE; BIT 3 = READ FROM CASSETTE
0110 ;CONNECT DIRECTLY INTO CASSETTE DECK VIA 6530 I/O
0120 ;
0130 ;
0140 ;== VARIABLES ==
0150 CHECKSUM   .DE $C2  TWO BYTE CHECKSUM
0160 COUNT      .DE ADDRS
0170 FORMBYTE   .DE $C4
0180 SYNCCOUNT  .DE FORMBYTE
0190 BITCOUNT   .DE $C5
0200 ADDRS      .DE $C6
0210 ;
0220 ;INPUT PARMS
0230 LOAD/NO    .DE $0180  00=NO LOAD:  01=LOAD
0240 START      .DE $0181
0250 END        .DE $0183
```

```
                        0260 :
                        0270 :
                        0280 :
                        0290 :C/WRITE:  WRITE TO TAPE FROM (START) TO (END)
                        0300 :
3E00  AD016E            0310 C/WRITE    LDA C/PORTD
3E03  0904              0320            ORA #04
3E05  8D016E            0330            STA C/PORTD
                        0340 :THE ABOVE INITIALIZES BIT 2 FOR OUTPUT ON 6530
                        0350 :
3E08  A920              0360            LDA #20  32 TIMES
3E0A  85C6              0370            STA *COUNT
3E0C  A916              0380 LOOP/RECST LDA #16  SYNC CHAR.
3E0E  20413E            0390            JSR WRITE/BYTE
                        0400 :
3E11  A910              0410            LDA #10
3E13  85C4              0420            STA *SYNC+COUNT
3E15  205D3E            0430 LOOP/DELSY JSR OUT:ZERO
3E18  C6C4              0440            DEC *SYNC+COUNT
3E1A  D0F9              0450            BNE LOOP/DELSY
                        0460 :DELAY TIME FOR SYNC
                        0470 :
3E1C  C6C6              0480            DEC *COUNT
3E1E  D0EC              0490            BNE LOOP/RECST
                        0500 :
3E20  20843E            0510            JSR MOVE+ST/AD  START > ADDRS (2)
                        0520 :
3E23  A90F              0530            LDA #0F  RECORD START CHAR.
3E25  20413E            0540            JSR WRITE/BYTE
                        0550 :
3E28  A200              0560            LDX #00
3E2A  86C2              0570            STX *CHECKSUM   CLEAR CHECKSUM
3E2C  86C3              0571            STX *CHECKSUM+$01
                        0580 :
3E2E  A1C6              0590 LOOP/DATA  LDA (ADDRS,X)  LOAD DATA
3E30  20413E            0600            JSR WRITE/BYTE
3E33  20BF3E            0610            JSR INC/COMP
3E36  90F6              0615            BCC LOOP/DATA
                        0619 :
                        0620 :
3E38  A5C3              0621            LDA *CHECKSUM+$01
3E3A  48                0622            PHA     SAVE HI CKSUM
3E3B  A5C2              0630            LDA *CHECKSUM
3E3D  20413E            0631            JSR WRITE/BYTE   WRITE LO CKSUM FIRST
3E40  68                0632            PLA     HI CKSUM NEXT
                        0640 :THE ABOVE WRITES BOTH CHECKSUM BYTES
                        0650 :
                        0660 :
                        0670 :ROUTINE TO WRITE A BYTE TO TAPE
                        0680 :
3E41  85C4              0690 WRITE/BYTE STA *FORM+BYTE
3E43  20343F            0691            JSR CKSUM+ADD  UPDATE CHECKSUM COUNTER
3E46  207C3E            0700            JSR OUT:ONE  START BIT
3E49  A908              0710            LDA #08 8 BITS
3E4B  85C5              0720            STA *BIT.COUNT
3E4D  06C4              0730 DATA/LOOP  ASL *FORM+BYTE  SHIFT LEFT INTO CARRY
3E4F  9005              0740            BCC ZERO.BIT
3E51  207C3E            0760 ONE.BIT    JSR OUT:ONE
3E54  F003              0770            BEQ CK+END+BY
3E56  205D3E            0790 ZERO.BIT   JSR OUT:ZERO
3E59  C6C5              0800 CK+END+BY  DEC *BIT.COUNT
3E5B  D0F0              0810            BNE DATA/LOOP
                        0820 :NOW OUTPUT 1 STOP BIT
                        0830 :
                        0840 :ROUTINE OUTPUT A ZERO TO TAPE
                        0850 :
3E5D  A920              0860 OUT:ZERO   LDA #20  '0' DELAY CONSTANT
                        0870 :
```

```
                        0880 :
                        0890 :ROUTINE WRITE TO TAPE
                        0900 :
3E5F  48                0910 WRITE      PHA   SAVE DELAY CONSTANT
3E60  AD006E            0920            LDA C/PORT
3E63  0904              0930            ORA #04   OUT A '1'
3E65  8D006E            0940            STA C/PORT
3E68  68                0950            PLA
3E69  48                0960            PHA
3E6A  AA                0970            TAX   DELAY CONSTANT
3E6B  207B3E            0980            JSR LOOPD
3E6E  AD006E            0990            LDA C/PORT
3E71  29FB              1000            AND #FB   OUT A '0'
3E73  8D006E            1010            STA C/PORT
3E76  68                1020 X          PLA
3E77  AA                1030            TAX   DELAY CONSTANT
3E78  CA                1040 LOOPD      DEX
3E79  D0FD              1050            BNE LOOPD
3E7B  60                1060            RTS
                        1070 :
                        1080 :
                        1090 :
                        1100 :ROUTINE OUTPUT A ONE TO TAPE
                        1110 :
3E7C  A950              1120 OUT:ONE    LDA #50  '1' DELAY CONSTANT
3E7E  D0DF              1130            BNE WRITE
                        1140 :
                        1150 :
                        1160 :DELAY FOR '0' TIME FOR READ
                        1170 :
3E80  A230              1180 READ.DELAY LDX #30
3E82  D0F4              1190            BNE LOOPD
                        1200 :
                        1210 :
                        1220 :
                        1230 :ROUTINE MOVE FROM START TO ADDRS
                        1240 :
3E84  AD8101            1250 MOVE+ST/AD LDA START
3E87  85C6              1260            STA *ADDRS
3E89  AD8201            1270            LDA START+$01
3E8C  85C7              1280            STA *ADDRS+$01
3E8E  60                1290            RTS
                        1300 :
                        1310 :
                        1320 :
                        1330 :ROUTINE INCREMENT AND COMPARE
                        1340 :
3E8F  E6C6              1350 INC/COMP   INC *ADDRS
3E91  D002              1360            BNE SKIP/INC
3E93  E6C7              1370            INC *ADDRS+$01
3E95  A5C7              1380 SKIP/INC   LDA *ADDRS+$01
3E97  CD8401            1390            CMP END+$01
3E9A  9008              1400            BCC NOT/END
3E9C  A5C6              1410            LDA *ADDRS
3E9E  CD8301            1420            CMP END
3EA1  9001              1430            BCC NOT/END
3EA3  38                1440            SEC
3EA4  60                1450 NOT/END    RTS
                        1460 :ON RETURN, C=CLEAR: NOT END: C=SET: END REACHED
                        1470 :
                        1480 :
                        1490 :
                        1500 :
                        1510 :C/READ:  READ FROM TAPE TO (START) TO (END)
                        1520 :
3EA5  A200              1530 C/READ     LDX #00
3EA7  86C6              1540            STX *COUNT
3EA9  20EF3E            1550 LOOP/LOAD  JSR READ/BYTE
```

```
3EAC  C916    1560            CMP #16  SYNC
3EAE  D004    1570            BNE SKIP/1
3EB0  E6C6    1580            INC *COUNT
3EB2  D0F5    1590            BNE LOOP/LOAD
              1600 :
3EB4  A4C6    1610 SKIP/1     LDY *COUNT
3EB6  C00A    1620            CPY #0A   MUST BE > = 10 SYNC'S
3EB8  90EB    1630            BCC C/READ
3EBA  C90F    1640            CMP #0F  RECORD START
3EBC  D0E7    1650            BNE C/READ
              1660 :
3EBE  A000    1670            LDY #00
3EC0  84C2    1680            STY *CHECKSUM
3EC2  84C3    1681            STY *CHECKSUM+$01   CLEAR CHECKSUM LOCATIONS
3EC4  20943E  1690            JSR MOVE*ST/AD   START > ADDRS (2)
              1700 :
              1710 ;NOW LOAD DATA
3EC7  20EF3E  1720 LOOP/69    JSR READ/BYTE
3ECA  ACA041  1730            LDY LOAD/NO  CKG. IF TO STORE
3ECD  F002    1740            BEQ SKIP/STORE
3ECF  81C6    1750            STA (ADDRS,X)
3ED1  208F3E  1760 SKIP/STORE JSR INC/COMP
3ED4  90F1    1770            BCC LOOP/69
3ED6  A5C3    1771            LDA *CHECKSUM+$01
3ED8  48      1772            PHA   SAVE CHSUM HI
3ED9  A5C2    1780            LDA *CHECKSUM
3EDB  48      1790            PHA   SAVE CHECKSUM LO
3EDC  20EF3E  1800            JSR READ/BYTE
3EDF  68      1810            PLA
3EE0  C5C4    1820            CMP *FORM*BYTE   CHECK CHECKSUM LO
3EE2  D007    1821            BNE RETURN
3EE4  20EF3E  1822            JSR READ/BYTE
3EE7  68      1823            PLA
3EE8  C5C4    1824            CMP *FORM*BYTE   CHECK CHECKSUM HI
3EEA  60      1830            RTS
3EEB  68      1831 RETURN     PLA
3EEC  A9FF    1832            LDA #FF  CLEAR Z-BIT
3EEE  60      1833            RTS
              1840 ;ON RETURN Z-BIT=TRUE:GOOD LOAD;  Z-BIT=FALSE:ERROR
              1850 ;
              1860 ;
              1870 ;ROUTINE READ A BYTE FROM TAPE
              1880 ;
3EEF  202E3F  1890 READ/BYTE  JSR IN/PORT
3EF2  D0FB    1900            BNE READ/BYTE  LOOP UNTIL 0
              1910 ;
3EF4  202E3F  1920 WAIT*FOR*1 JSR IN/PORT
3EF7  F0FB    1930            BEQ WAIT*FOR*1  LOOP UNTIL 1
              1940 ;
3EF9  20803E  1950            JSR READ.DELAY
3EFC  202E3F  1960            JSR IN/PORT
3EFF  F0F3    1970            BEQ WAIT*FOR*1  IF ZERO
              1980 ;
3F01  202E3F  1990 WAIT*FOR*0 JSR IN/PORT
3F04  D0FB    2000            BNE WAIT*FOR*0  WAIT TIL END OF START BIT
              2010 ;
3F06  A908    2020            LDA #08
3F08  85C5    2030            STA *BIT.COUNT
              2040 ;
3F0A  202E3F  2050 WAIT*TO*CN JSR IN/PORT
3F0D  F0FB    2060            BEQ WAIT*TO*CN  LOOP UNTIL '1'
3F0F  20803E  2070            JSR READ.DELAY
3F12  202E3F  2080            JSR IN/PORT
3F15  F008    2090            BEQ PROCESS*0  IF '0' THEN ZERO. ELSE ONE
3F17  202E3F  2110 PROCESS*1  JSR IN/PORT
3F1A  D0FB    2120            BNE PROCESS*1  LOOP UNTIL '0'
3F1C  38      2130            SEC
3F1D  B001    2140            BCS ROTATE*IN
3F1F  18      2160 PROCESS*0  CLC
3F20  26C4    2170 ROTATE*IN  ROL *FORM*BYTE   ROTATE CARRY
3F22  C6C5    2180            DEC *BIT.COUNT
3F24  D0E4    2190            BNE WAIT*TO*CN
3F26  A5C4    2200            LDA *FORM*BYTE
3F28  20843F  2201            JSR CKSUM*ADD   UPDATE CHECKSUM
3F2B  A5C4    2202            LDA *FORM*BYTE
3F2D  60      2210            RTS
              2220 ;
              2230 ;
              2240 ;INPUT FROM TAPE
              2250 ;
3F2E  AD006E  2260 IN/PORT    LDA C/PORT
3F31  2908    2270            AND #08  MASK OUT ALL BUT BIT 3
3F33  60      2280            RTS
              2281 ;
              2282 ;
              2283 ;
              2284 ;UPDATE CHECKSUM COUNTERS
              2285 ;
3F34  18      2286 CKSUM*ADD  CLC
3F35  D8      2287            CLD
3F36  65C2    2288            ADC *CHECKSUM+$00   ADD R(A) TO CKSUM LO
3F38  85C2    2289            STA *CHECKSUM+$00
3F3A  A900    2290            LDA #00
3F3C  65C3    2291            ADC *CHECKSUM+$01   ADD 00 TO CKSUM HI
3F3E  85C3    2292            STA *CHECKSUM+$01
3F40  60      2293            RTS
              2294 ;
              2300 ;
              2310 ;
3F41  20A53E  2320 LOAD*ENTRY JSR C/READ
3F44  D008    2330            BNE BAD
3F46  A900    2340            LDA #00  INDICATE GOOD LOAD BY R(A)=00
3F48  00      2350 B          BRK
3F49  EA      2360            NOP
3F4A  EA      2370            NOP
3F4B  4C413F  2380            JMP LOAD*ENTRY
3F4E  A9EE    2390 BAD        LDA #EE  INDICATE BAD LOAD BY R(A)=EE
3F50  D0F6    2400            BNE B
              2410 ;
3F52  20303E  2420 RECORD*ENT JSR C/WRITE
3F55  00      2430            BRK
3F56  EA      2440            NOP
3F57  EA      2450            NOP
3F58  4C523F  2460            JMP RECORD*ENT
              2470 ;
              2480 END*OF*PGM .EN

LABEL FILE:  [ / = EXTERNAL ]

/C/PORT=6E00            /C/PORTD=6E01           /CHECKSUM=00C2
/COUNT=00C6             /FORM*BYTE=00C4         /SYNC*COUN T=00C4
/BIT.COUNT=00C5         /ADDRS=00C6             /LOAD/NO=0180
/START=0181             /END=0183               C/WRITE=3E30
LOOP/RECST=3E0C         LOOP/DELSY=3E15         LOOP/DATA=3E2E
WRITE/BYTE=3E41         DATA/LOOP=3E4D          ONE.BIT=3E51
ZERO.BIT=3E56           CK*END*BY=3E59          OUT:ZERO=3E5D
WRITE=3E5F              X=3E76                  LOOPD=3E78
OUT:ONE=3E7C            READ.DELAY=3E80         MOVE*ST/AD=3E94
INC/COMP=3E8F           SKIP/INC=3E95           NOT/END=3EA4
C/READ=3EA5             LOOP/LOAD=3EA9          SKIP/1=3EB4
LOOP/69=3EC7            SKIP/STORE=3ED1         RETURN=3EEB
READ/BYTE=3EEF          WAIT*FOR*1=3EF4         WAIT*FOR*0=3F01
WAIT*TO*CN=3F0A         PROCESS*1=3F17          PROCESS*0=3F1F
ROTATE*IN=3F20          IN/PORT=3F2E            CKSUM*ADD=3F34
LOAD.ENTRY=3F41         B=3F48                  BAD=3F4E
RECORD.ENT=3F52         END*OF*PGM=3F58
```

A BASIC Renumbering Program

BY JOHN B. TRENHOLME
120 Montair Ct.
Danville, CA 94526
(415) 820-0909

This program renumbers Microsoft BASIC programs. It has been designed to have all its routines self-contained so that it doesn't depend on the BASIC interpreter for subroutines. This makes it larger than it might be, but also means it will work with a variety of interpreters rather than just one. The specific version presented is for 3.1 8K cassette BASIC, but simple changes (below) will adapt it to other versions.

The program allows you to renumber starting from a line number in the middle of the program, rather than just from the beginning. This is so you can identify logical blocks of program statements by giving them line numbers starting with (for example) 1000, 2000, 3000, etc. Debugging is much easier when this is done, and other people can read the listing without getting hopelessly lost.

Input to the program consists of three numbers in RAM:

1. The increment to be used between lines (stored at 0004H & 0005H).
2. The new line number of the first line to be renumbered (stored at 0006H & 0007H).
3. The old line number of the first line to be renumbered (stored at 0039H & 003AH = 57_{10} & 58_{10}).

These numbers are stored in low RAM postitions so that BASIC can POKE them to the desired values. The first two are in places that BASIC uses to store the addresses of internal routines for the user. These addresses are never used by BASIC. The third argument is stored in two NOP locations following a RETurn in RST7. If interrupt is used via RST7, another location will have to be used.

The renumbering program takes 512 bytes. The version presented here is for use with a SOL, so it is assembled in the SOL scratch RAM starting at C900H. To put it elsewhere by hand, just change all the C9's that are the third byte of 3-byte instructions to 3E if you want the program to start at 3E00 (or whatever), and all the CA's to 3F (or whatever). The first C9, for example, is at C90E. The program uses two bytes of RAM which are presently at C8FE and C8FF; so C8FE will have to be changed to somewhere else. Except for these two bytes, the whole thing can be in ROM.

The following short program will renumber BASIC from inside itself. It is especially handy while writing a program, since you can quickly make space for more lines if you inadvertently leave something out. When done, you can move the first line of the program up to 10000 or so and renumber the successive blocks to higher multiples of 1000 (then hitting RETURN during input and typing more numbers probably won't clobber the program). Then just delete the lines containing the renumbering stuff, and you're set to go.

```
1 INPUT "Increment between lines"; I
2 J = INT(I/256): POKE 4,I-256*J: POKE 5,J
3 INPUT "New number of first renumbered line"; I
4 J = INT(I/256): POKE 6,I-256*J: POKE 7,J
5 INPUT "Old number of first line to renumber"; I
6 J = INT(I/256): POKE 57,I-256*J: POKE 58,J
7 POKE 73,0: POKE 74,201: J = USR(J)
```

In the last line, 73 and 74 are the locations of the user-defined function, which depends on the version of BASIC being used. The 201 is the high byte of C900H in "split decimal" notation, which is what BASIC uses when referring to memory locations. Change 201 if you move the program. To reuse the same increment you used last time, type GOTO 3 instead of RUN. If you've added new lines between old ones and want to clean things up, just use GOTO 7.

Be careful when using this program. No error checking is done to insure that you don't renumber the program in non-ascending order, allow the line numbers to get larger than 65535 so they will wrap around to 0, or otherwise screw things up. The author, from painful experience, always keeps a backup copy of the program before making changes.

The one check that is made is for non-existent line numbers. If the program finds GOTO 300 and there is no line 300, it makes up a line number which is one more than the line number that would follow the last line and puts that in instead. For example if the increment between lines is 10 and the last line number becomes 540 after renumbering, the GOTO 300 becomes GOTO 551. This is easily picked out when reading a listing, and avoids incorrect jumps if (as is often the case) you start typing new lines in at 550.

In Microsoft BASIC, the program is stored in RAM as a series of lines. Each line has a two-byte pointer at the beginning, which points to the next line. Then there are two bytes containing the line number in lo-hi binary form. Then the text of the line is written in ASCII, except that reserved words such as FOR, GOTO, SIN, etc. are replaced by one-byte "tokens" which have their high bit set to 1 (i.e., they are "negative"). In some versions, tokens may have more than one byte. The text of each line is ended by a 00 byte. The program as a whole is ended by a 0000 pointer, so the end has three 00 bytes in a row (one for the line, two for the pointer). Because the pointers (which are used for fast searches in GOTO's and GOSUB's) are in absolute address form, the text can't be moved unless the pointers are reset; it would be better to use one byte to specify the length of the line so the program would be relocatable and shorter. This would limit line length to 256 bytes, which should not be a problem.

The program works by moving the program text as a whole up near the stack, which is presumably well above the end of the program (if not, memory is almost full and disaster may occur—you did keep a backup copy, didn't you?). It then starts reading the text of the program, looking for GOTO, GOSUB, and THEN tokens (ELSE also in extended). When a token is found, it then looks for numbers after the token (leading blanks before the numbers are copied unchanged). When a number is found, it is converted to binary and a new number is created. Instead of making a table of old and new numbers, the program makes up the new number by using the

old line numbers that are stored in the program text. It starts at the first line to be renumbered, and counts from the first new line number by the increment amount, stepping once for each line encountered. When the line with the old line number is found, it has the new line number. This new number is then converted to ASCII and inserted into the program text. The program looks for a comma after the line number; if one is found, it assumes there is a series of line numbers (as in ON K GOTO 10, 20, 30) and looks for another.

As the search and conversion occur, the new text is moved back down in RAM, and the pointers are reset appropriately. The moving up and down is necessary because the length of the text may change as (for example) line 214 becomes line 10160 (life would be much easier if the line numbers were pre-converted to binary before text storage). After all the numbers in the text are reset, the numbers at the beginnings of the lines are set to their new values. A jump is then made to a section of BASIC which resets the stack and zeroes out the variables (since the renumbering has very probably clobbered them). The stack is preset before the jump so that the RETurn from this BASIC routine is to location 0000, which causes BASIC to give its OK and wait for more input.

The one word of RAM used is the storage for a pointer to the beginning of the first line of be renumbered during the text change phase, and for the new line number storage when the numbers at the front of the lines are being changed. The only tricky part is the updating of this pointer when the line it points to is moved back down in RAM.

The method of using the lines themselves to make up new line numbers is slower than making up a table (see Steve Wozniak's recent article in *DDJ* no. 23 on renumbering in an Apple), and in fact takes time proportional to the square of the number of lines to be renumbered. However, the program renumbers a 400 line program in about 7 seconds on a 2 MHz 8080 system, so speed should not be a problem except in exceptional cases. The space saved means larger programs can be renumbered, so the tradeoff between time and space goes to space in this case.

Other versions of BASIC can be renumbered also, but a new version of the program will be needed for each one (sorry 'bout that). The things the program has to know are:

1. The location of the stack base pointer STACK. This is set by BASIC's initialization routine, after it knows how much memory it can use. Look in this section of the code to find where it is (or see the listing of NEW below).
2. The location of the pointer TEXT to the beginning of the BASIC text. This usually follows STACK in RAM. The text usually begins at or near the beginning of the initialization code, so you can look for locations containing this address (plus or minus).
3. The location of the pointer SCALR to the end of the BASIC text and the beginning of scalar variable storage. This usually follows TEXT in RAM.
4. The location NEW2 of the entry to the routine for NEW which does everything except text removal. NEW will set SCALR to equal TEXT to delete text, then erase array variables and strings similarly, reset the stack, and RETurn. This program has to jump in after the text removal phase. See the appendix below for more about NEW.
5. The tokens for GOTO, GOSUB, THEN, and (if extended) ELSE. These are most easily found by writing a short program containing them and looking at the resulting text of the program (starting where TEXT points). In this version, ELSE is not used, but room has been left for testing it by testing twice for THEN (see after label NONUM). This makes hand patching easy.

There are a number of useful subroutines buried in this program. Comparison of DE and HL is done by DEMHL. TSDIG checks to see if a byte represents a numeric character (0-9) in ASCII. ASCDE is a standard routine to convert an unsigned ASCII number between 0 and 65535 to binary (without error checking). And DEASC will convert an unsigned binary number to ASCII in RAM in about 2000 machine cycles, with leading zeroes suppressed (the heart of this routine is a modified version of one found in an article by Tabb and Roginsky in *EDN* for January 5, 1977, page 46).

I hope you have a lot of fun using this routine, modifying it, and (hopefully) improving it. If you come up with something nice (the needed addresses for other BASICs, faster or shorter code, etc.) drop a note to good ol' *Dr. Dobb's* so the rest of us can know about it.

Appendix

The NEW routine can be found by looking in the command address table for any specific version of Microsoft BASIC. In 3.1 8K cassette, this starts at 0170H. The entries are in the same order as the literal table of commands, which contains the ASCII text of the commands, functions, etc. (the first or last character of each reserved word will have its high bit on); this starts at 0088H (3.1 8K cassette only!). Just count off two bytes per routine until you come to the entry for NEW, which will be the address in lo-hi order. Then look in that location for the routine. In 3.1 8K cassette, NEW starts at 039DH and looks like this:

```
NEW     RNZ             QUIT IF SOMETHING FOLLOWS
NEW1    LHLD TEXT       GET POINTER TO TEXT AREA
        XRA  A          ZERO
        MOV  M,A        LO POINTER =0
        INX  H
        MOV  M,A        HI POINTER =0
        INX  H
        SHLD SCALR      SET TEXT TOP

--------THIS IS WHERE WE COME IN--------

NEW2    LHLD TEXT       POINT TO TEXT
        DCX  H
NEW3    SHLD NTEXT      POINTER TO NEXT TEXT
        LHLD MEMTP      TOP OF MEMORY
        SHLD STRNG      ZERO OUT STRINGS
        CALL RESTR      DO "RESTORE"
        LHLD SCALR
        SHLD ARRAY      ZERO OUT SCALARS
        SHLD ENDVR      ZERO OUT ARRAYS
NEW4    POP  B          RETURN ADDRESS
        LHLD STACK
        SPHL            RESET STACK
        LXI  H,SLOTS    RESET TEMPORARIES
        SHLD SLOTP
        LXI  H,0        RETURN ADDRESS =0
        PUSH H
        SHLD STEXT      TEXT IF STOPPED
        LHLD NTEXT      NEXT TEXT
        XRA  A
        STA  VARFL      FLAG FOR FOR, DIM, FN
        PUSH B          RESTORE RETURN ADDRESS
        RET             ALL DONE
```

LISTING

```
0000                    0001 ************************************
0000                    0002 * RENUMBER MICROSOFT BASIC PROGRAMS*
0000                    0003 ************************************
0000                    0004 * @ COPYRICHT 1978 JOHN TRENHOLME  *
0000                    0005 * PERMISSION GRANTED FOR ALL NON-  *
0000                    0006 * COMMERCIAL USE BY INDIVIDUALS.   *
0000                    0007 ************************************
0000                    0008 *
0000                    0009 * THESE CONSTANTS FOR 3.1 8K CASETTE
0000                    0010 *
0000                    0011 STACK   EQU    0241H
0000                    0012 TEXT    EQU    STACK+2
0000                    0013 SCALR   EQU    TEXT+2
0000                    0014 NEW2    EQU    03A9H
0000                    0015 * TOKENS
0000                    0016 GOTO    EQU    88H
0000                    0017 GOSUB   EQU    8CH
0000                    0018 THEN    EQU    0A1H
0000                    0019 ELSE    EQU    THEN    FOR 3.1
0000                    0020 *
0000                    0021 * ARGUMENTS IN LOW RAM FOR POKE
0000                    0022 *
0000                    0023 * LINE NUMBER INCREMENT LOC.
0000                    0024 DELTA   EQU    0004H
0000                    0025 * NEW START LINE NUMBER LOC.
0000                    0026 NEW1    EQU    0006H
0000                    0027 * OLD START LINE NUMBER LOC.
0000                    0028 OLD1    EQU    0039H   (NO RST 7)
0000                    0029 *
0000                    0030         ORG    0C8FEH (FOR SOLOS)
C8FE 00 00              0031 POINT   DW     0       TEMPORARY
C900                    0032 * START OF CODE
C900 2A 41 02           0033 RENUM   LHLD   STACK
C903 F9                 0034         SPHL
C904 21 00 00           0035         LXI    H,0
C907 E5                 0036         PUSH   H       ULTIMATE RETURN
C908 2A 43 02           0037         LHLD   TEXT
C90B E5                 0038         PUSH   H
C90C CD 40 C9           0039         CALL   FIND    NC IF NOT FOUND
C90F D1                 0040         POP    D
C910 D0                 0041         RNC    .       IF NO OLD MAX
C911                    0042 * MOVE TEXT TO TOP
C911 21 E0 FF           0043         LXI    H,-32   ELBOW ROOM
C914 39                 0044         DAD    SP
C915 4D                 0045         MOV    C,L
C916 44                 0046         MOV    B,H
C917 2A 45 02           0047         LHLD   SCALR
C91A 0B                 0048 MVTXT   DCX    B
C91B 2B                 0049         DCX    H
C91C 7E                 0050         MOV    A,M
C91D 02                 0051         STAX   B
C91E CD 5F C9           0052         CALL   DEMHL
C921 C2 1A C9           0053         JNZ    MVTXT
C924 C5                 0054         PUSH   B       SAVE LOW END
C925 69                 0055         MOV    L,C
C926 60                 0056         MOV    H,B
C927                    0057 * NEW POINTER CHAIN
C927 5D                 0058 CHAIN   MOV    E,L
C928 54                 0059         MOV    D,H
C929 7E                 0060         MOV    A,M
C92A 23                 0061         INX    H
C92B B6                 0062         ORA    M
C92C                    0063 * 00 00 IS END OF TEXT
C92C CA 65 C9           0064         JZ     WHERE
C92F 23                 0065         INX    H       LINE # LO
C930 23                 0066         INX    H       LINE # HI
C931 23                 0067         INX    H
C932 AF                 0068         XRA    A
C933                    0069 * 00 MEANS END OF THIS LINE
C933 BE                 0070 FIND0   CMP    M
C934 23                 0071         INX    H
C935 C2 33 C9           0072         JNZ    FIND0
C938 EB                 0073         XCHG
C939 73                 0074         MOV    M,E
C93A 23                 0075         INX    H
C93B 72                 0076         MOV    M,D
C93C EB                 0077         XCHG
C93D C3 27 C9           0078         JMP    CHAIN
C940                    0079 *
C940                    0080 * FIND START LINE NUMBER OR NEXT
C940                    0081 *
C940 E5                 0082 FIND    PUSH   H       RESULT
C941 4E                 0083         MOV    C,M     POINTER LO
C942 23                 0084         INX    H
C943 46                 0085         MOV    B,M     POINTER HI
C944 79                 0086         MOV    A,C
C945 B0                 0087         ORA    B
C946 CA 5D C9           0088         JZ     FAIL    IF END
C949 C5                 0089         PUSH   B
C94A 23                 0090         INX    H
C94B 5E                 0091         MOV    E,M
C94C 23                 0092         INX    H
C94D 56                 0093         MOV    D,M
C94E 2A 39 00           0094         LHLD   OLD1
C951 CD 5F C9           0095         CALL   DEMHL
C954 E1                 0096         POP    H
C955 D2 5C C9           0097         JNC    GOTIT
C958 C1                 0098         POP    B       CLEAR STACK
C959 C3 40 C9           0099         JMP    FIND
C95C 37                 0100 GOTIT   STC
C95D E1                 0101 FAIL    POP    H
C95E C9                 0102         RET
C95F                    0103 *
C95F                    0104 * SET FLAGS FOR DE MINUS HL
C95F                    0105 *
C95F 7A                 0106 DEMHL   MOV    A,D
C960 94                 0107         SUB    H
C961 C0                 0108         RNZ
C962 7B                 0109         MOV    A,E
C963 95                 0110         SUB    L
C964 C9                 0111         RET
C965                    0112 *
C965                    0113 * FIRST LINE TO RENUMBER
C965                    0114 *
C965 E1                 0115 WHERE   POP    H       -> MOVED
C966 E5                 0116         PUSH   H
C967 CD 40 C9           0117         CALL   FIND
```

```
C96A 22 FE C8            0118          SHLD    POINT
C96D 2A 43 02            0119          LHLD    TEXT
C970 4D                  0120          MOV     C,L
C971 44                  0121          MOV     B,H
C972 21 43 02            0122          LXI     H,TEXT
C975 E3                  0123          XTHL
C976                     0124 *
C976                     0125 * MAIN LOOP OVER LINES
C976                     0126 *
C976 E3                  0127 LOOP     XTHL
C977 71                  0128          MOV     M,C     RESET LAST
C978 23                  0129          INX     H       ...POINTER
C979 70                  0130          MOV     M,B
C97A D1                  0131          POP     D       -> TEXT
C97B C5                  0132          PUSH    B       FOR NEXT PASS
C97C 2A FE C8            0133          LHLD    POINT   IS THIS FIRST?
C97F CD 5F C9            0134          CALL    DEMHL
C982 C2 8B C9            0135          JNZ     CHEK0
C985 21 FE C8            0136          LXI     H,POINT  IF SO, RESET
C988 71                  0137          MOV     M,C
C989 23                  0138          INX     H
C98A 70                  0139          MOV     M,B
C98B EB                  0140 CHEK0    XCHG    .       IS IT THE END?
C98C 7E                  0141          MOV     A,M
C98D 02                  0142          STAX    B
C98E 03                  0143          INX     B
C98F 23                  0144          INX     H
C990 B6                  0145          ORA     M
C991 CA 7A CA            0146          JZ      FINAL   YES, TEXT END
C994 7E                  0147          MOV     A,M
C995 02                  0148          STAX    B
C996 03                  0149          INX     B       LINE # LO
C997 23                  0150          INX     H
C998 7E                  0151          MOV     A,M
C999 02                  0152          STAX    B
C99A 03                  0153          INX     B       LINE # HI
C99B 23                  0154          INX     H
C99C 7E                  0155          MOV     A,M
C99D 02                  0156          STAX    B
C99E                     0157 * SCAN TEXT OF LINE
C99E 03                  0158 LINE     INX     B
C99F 23                  0159          INX     H
C9A0 7E                  0160          MOV     A,M
C9A1 FE 88               0161 NONUM    CPI     GOTO
C9A3 CA B5 C9            0162          JZ      LNUMB
C9A6 FE 8C               0163          CPI     GOSUB
C9A8 CA B5 C9            0164          JZ      LNUMB
C9AB FE A1               0165          CPI     THEN
C9AD CA B5 C9            0166          JZ      LNUMB
C9B0                     0167 * DUMMY "ELSE" FOR 3.1
C9B0 FE A1               0168          CPI     ELSE
C9B2 C2 D9 C9            0169          JNZ     TEST0
C9B5                     0170 * CONVERT LINE NUMBER(S) AFTER
C9B5                     0171 * "GOTO", "GOSUB", "THEN", OR "ELSE"
C9B5 02                  0172 LNUMB    STAX    B
C9B6 03                  0173          INX     B
C9B7 23                  0174          INX     H
C9B8 7E                  0175          MOV     A,M     MOVE OVER...
C9B9 FE 20               0176          CPI     ' '     LEADING BLANKS
C9BB CA B5 C9            0177          JZ      LNUMB
C9BE CD E3 C9            0178          CALL    TSDIG
C9C1 DA A1 C9            0179          JC      NONUM   IF NO NUMBER
C9C4                     0180 * CONVERT TO BINARY
C9C4 CD EA C9            0181          CALL    ASCDE
C9C7 E5                  0182          PUSH    H
C9C8 C5                  0183          PUSH    B
C9C9                     0184 * GET NEW NUMBER (IF ANY)
C9C9 CD 0A CA            0185          CALL    NEWLN
C9CC E1                  0186          POP     H
C9CD                     0187 * NOW WRITE AS ASCII
C9CD CD 3D CA            0188          CALL    DEASC
C9D0 4D                  0189          MOV     C,L
C9D1 44                  0190          MOV     B,H
C9D2 E1                  0191          POP     H
C9D3 7E                  0192          MOV     A,M
C9D4 FE 2C               0193          CPI     ','     MORE TO COME?
C9D6 CA B5 C9            0194          JZ      LNUMB
C9D9 02                  0195 TEST0    STAX    B
C9DA B7                  0196          ORA     A
C9DB C2 9E C9            0197          JNZ     LINE
C9DE 03                  0198          INX     B
C9DF 23                  0199          INX     H
C9E0 C3 76 C9            0200          JMP     LOOP    00 =END OF LINE
C9E3                     0201 *
C9E3                     0202 * SET CARRY IF 'A' NOT DIGIT
C9E3                     0203 *
C9E3 FE 30               0204 TSDIG    CPI     '0'
C9E5 D8                  0205          RC
C9E6 FE 3A               0206          CPI     '9'+1
C9E8 3F                  0207          CMC
C9E9 C9                  0208          RET
C9EA                     0209 *
C9EA                     0210 * ASCII AT HL-> TO BINARY IN DE
C9EA                     0211 *
C9EA 11 00 00            0212 ASCDE    LXI     D,0
C9ED CD E3 C9            0213 DIGIT    CALL    TSDIG   DIGIT?
C9F0 D8                  0214          RC      .       NOPE; QUIT
C9F1 E5                  0215          PUSH    H
C9F2 6B                  0216          MOV     L,E
C9F3 62                  0217          MOV     H,D
C9F4 29                  0218          DAD     H       X2
C9F5 29                  0219          DAD     H       X4
C9F6 19                  0220          DAD     D       X5
C9F7 29                  0221          DAD     H       X10
C9F8 D6 30               0222          SUI     '0'
C9FA 5F                  0223          MOV     E,A
C9FB 16 00               0224          MVI     D,0
C9FD 19                  0225          DAD     D
C9FE EB                  0226          XCHG
C9FF                     0227 * NOW DE = 10*DE + DIGIT
C9FF E1                  0228          POP     H
CA00 23                  0229 IGBLK    INX     H
CA01 7E                  0230          MOV     A,M
CA02 FE 20               0231          CPI     ' '
CA04 CA 00 CA            0232          JZ      IGBLK   IGNORE BLANKS
CA07 C3 ED C9            0233          JMP     DIGIT   INSIDE OR END
CA0A                     0234 *
CA0A                     0235 * OLD LINE IN DE -> NEW LINE IN DE
CA0A                     0236 *
CA0A 2A 39 00            0237 NEWLN    LHLD    OLD1
CA0D CD 5F C9            0238          CALL    DEMHL
CA10 D8                  0239          RC      .       IF BELOW LIMIT
CA11 2A 06 00            0240 NEW      LHLD    NEW1
CA14 E5                  0241          PUSH    H
CA15 2A FE C8            0242          LHLD    POINT
CA18 4E                  0243 NEXTL    MOV     C,M
CA19 79                  0244          MOV     A,C
CA1A 23                  0245          INX     H
```

```
CA1B B6             0246          ORA     M       AT END?
CA1C CA 3A CA       0247          JZ      NOWAY
CA1F 46             0248          MOV     B,M
CA20 23             0249          INX     H
CA21 7E             0250          MOV     A,M
CA22 93             0251          SUB     E       LO BYTES
CA23 CA 32 CA       0252          JZ      HIBYT
CA26                0253 * INCREMENT NEW LINE NUMBER
CA26 E1             0254 BUMP     POP     H
CA27 C5             0255          PUSH    B
CA28 4D             0256          MOV     C,L
CA29 44             0257          MOV     B,H
CA2A 2A 04 00       0258          LHLD    DELTA
CA2D 09             0259          DAD     B
CA2E E3             0260          XTHL
CA2F C3 18 CA       0261          JMP     NEXTL
CA32 23             0262 HIBYT    INX     H
CA33 7E             0263          MOV     A,M
CA34 92             0264          SUB     D
CA35 C2 26 CA       0265          JNZ     BUMP
CA38 D1             0266          POP     D       OK, GOT IT
CA39 C9             0267          RET
CA3A                0268 * MAKE FUNNY LINE NUMBER
CA3A D1             0269 NOWAY    POP     D
CA3B 13             0270          INX     D
CA3C C9             0271          RET
CA3D                0272 *
CA3D                0273 * 16-BIT UNSIGNED BINARY INTEGER IN
CA3D                0274 * DE TO ASCII STRING WHERE HL->
CA3D                0275 * WITH LEADING ZEROES SUPPRESSED
CA3D                0276 *
CA3D 7A             0277 DEASC    MOV     A,D
CA3E B3             0278          ORA     E
CA3F CA 65 CA       0279          JZ      ITIS0
CA42 E5             0280          PUSH    H
CA43 EB             0281          XCHG
CA44 CD 6B CA       0282          CALL    BCD     10'S & 1'S
CA47 4F             0283          MOV     C,A
CA48 CD 6B CA       0284          CALL    BCD     1000'S & 100'S
CA4B 55             0285          MOV     D,L     10000'S
CA4C 47             0286          MOV     B,A
CA4D 1E 00          0287          MVI     E,0     SUPPRESS ZEROES
CA4F E1             0288          POP     H
CA50 CD 58 CA       0289          CALL    DIG2    HIGH TWO
CA53 50             0290          MOV     D,B
CA54 CD 58 CA       0291          CALL    DIG2    MID TWO
CA57 51             0292          MOV     D,C
CA58                0293 * PUT OUT TWO DIGITS IN D
CA58 7A             0294 DIG2     MOV     A,D
CA59 0F             0295          RRC
CA5A 0F             0296          RRC
CA5B 0F             0297          RRC
CA5C 0F             0298          RRC
CA5D CD 61 CA       0299          CALL    PUT     FIRST LEFT ONE
CA60 7A             0300          MOV     A,D
CA61                0301 * PUT OUT RIGHT DIGIT IN 'A'
CA61 E6 0F          0302 PUT      ANI     15
CA63 BB             0303          CMP     E       LEAD 0?
CA64 C8             0304          RZ      .       IF SO, DON'T
CA65 1E 30          0305 ITIS0    MVI     E,48    PRINT ZEROES
CA67 83             0306          ADD     E       ASCII OFFSET
CA68 77             0307          MOV     M,A
CA69 23             0308          INX     H
CA6A C9             0309          RET
CA6B                0310 * CONVERT TWO DECADES
CA6B 16 10          0311 BCD      MVI     D,16    BIT COUNT
CA6D AF             0312          XRA     A       START AT 0
CA6E 29             0313 CVT      DAD     H       SHIFT LEFT
CA6F 8F             0314          ADC     A       A = 2*A + CARRY
CA70 27             0315          DAA     .       DO IN DECIMAL
CA71 D2 75 CA       0316          JNC     TST
CA74 23             0317          INX     H       BACK INTO HL
CA75 15             0318 TST      DCR     D
CA76 C2 6E CA       0319          JNZ     CVT
CA79 C9             0320          RET
CA7A                0321 *
CA7A                0322 * PUT IN NEW NUMBERS ON LINES
CA7A                0323 *
CA7A 02             0324 FINAL    STAX    B
CA7B 03             0325          INX     B
CA7C E1             0326          POP     H       CLEAR STACK
CA7D 69             0327          MOV     L,C
CA7E 60             0328          MOV     H,B
CA7F 22 45 02       0329          SHLD    SCALR   NEW BASIC TOP
CA82 2A FE C8       0330          LHLD    POINT
CA85 E5             0331          PUSH    H
CA86 2A 06 00       0332          LHLD    NEW1
CA89 22 FE C8       0333          SHLD    POINT   STORE LINE #
CA8C E1             0334          POP     H
CA8D E5             0335 NEWNM    PUSH    H
CA8E 7E             0336          MOV     A,M
CA8F 23             0337          INX     H
CA90 B6             0338          ORA     M
CA91 CA AC CA       0339          JZ      QUIT    IF DONE
CA94 EB             0340          XCHG
CA95 2A FE C8       0341          LHLD    POINT
CA98 EB             0342          XCHG
CA99 23             0343          INX     H
CA9A 73             0344          MOV     M,E
CA9B 23             0345          INX     H
CA9C 72             0346          MOV     M,D
CA9D 2A 04 00       0347          LHLD    DELTA
CAA0 19             0348          DAD     D       INCREMENT #
CAA1 22 FE C8       0349          SHLD    POINT
CAA4 E1             0350          POP     H       FOLLOW POINTERS
CAA5 7E             0351          MOV     A,M
CAA6 23             0352          INX     H
CAA7 66             0353          MOV     H,M
CAA8 6F             0354          MOV     L,A
CAA9 C3 8D CA       0355          JMP     NEWNM
CAAC                0356 * 0 RETURN IS ON STACK...
CAAC                0357 * NOTE BASIC WILL RESET STACK
CAAC                0358 * BUT SAVE THE RETURN
CAAC E1             0359 QUIT     POP     H
CAAD C3 A9 03       0360          JMP     NEW2    BACK TO BASIC
```

ASCDE	C9EA	BCD	CA6B	BUMP	CA26	CHAIN	C927
CHEK0	C98B	CVT	CA6E	DEASC	CA3D	DELTA	0004
DEMHL	C95F	DIG2	CA58	DIGIT	C9ED	ELSE	00A1
FAIL	C95D	FINAL	CA7A	FIND	C940	FIND0	C933
GOSUB	008C	GOTIT	C95C	GOTO	0088	HIBYT	CA32
IGBLK	CA00	ITIS0	CA65	LINE	C99E	LNUMB	C9B5
LOOP	C976	MVTXT	C91A	NEW	CA11	NEW1	0006
NEW2	03A9	NEWLN	CA0A	NEWNM	CA8D	NEXTL	CA18
NONUM	C9A1	NOWAY	CA3A	OLD1	0039	POINT	C8FE
PUT	CA61	QUIT	CAAC	RENUM	C900	SCALR	0245
STACK	0241	TEST0	C9D9	TEXT	0243	THEN	00A1
TSDIG	C9E3	TST	CA75	WHERE	C965		

YET ANOTHER 6502 DISASSEMBLER FIX

Dear Dr. Dobbs: *Received: 78 May 22*

The Basic language disassembler for the MOS Technology 6502, published in Dr. Dobbs volume 3, number 1 (page 6), has proved to be an interesting and useful program, but appears to contain a number of errors. Some of these were corrected in volume 3, number 3 (page 3), but further bugs have since been found. I would recommend that the following changes be made, in addition to incorporation of those published previously:

```
DELETE STATEMENTS 611 AND 908
MAKE THE FOLLOWING CHANGES (ADDING NEW STATEMENTS 601 AND 645):
240 DATA 14100,35304,56100,56100,56100,35303, 3303,56100
300 DATA 36100,24210,33130,56100,28300,24300,33300,56100
310 DATA 12200,24222,56100,56100,56100,24203,33203,56100
320 DATA 16100,24304,56100,56100,56100,24303,33303,56100
330 DATA 42100, 1221,56100,56100,56100, 1200,56200,56100
380 DATA 23100,56100,53100,56100,49300,47300,48300,56100
390 DATA  4200,47222,56100,56100,49203,47203,48204,56100
520 DATA 45100,43304,56100,56100,56100,43303,25303,56100
601 D1=-123456
645 IF C=56100 THEN 1000
902 IF D1=-123456 THEN 909
1010 GOTO 910
```

These changes correct a number of errors in the five digit codes used for deciphering the instructions. A bug is also corrected in which a spurious operand with a value of zero would be printed if the first location disassembled contained a single byte instruction. Another change is made so that disassembly will not terminate if a non-valid opcode is sensed. This change was made both for reasons of convenience and to allow verification to be done, as described below. As with the original version of this program, no attempt is made to convert addresses or opcodes to hexadecimal or ASCII representation, nor is any attempt made to improve the nicety of the output formats.

When a computer program is written that includes a complex numeric table, it is desirable to devise a method of verifying the correctness of each entry, without the necessity of waiting for all possibilities to be encountered during execution. For this program, an exerciser was devised to generate each opcode possible, so that the program can be seen to properly disassemble every opcode and format combination. This method of verification is desirable, since the disassembled output generated by the program can then be considered to be reliable. The verification technique was performed on a PET computer, and is detailed here so that it can be repeated in the event that further program changes are made, to re-confirm that new bugs are not introduced.

To perform the verification, first load the disassembler into the computer. Then add the following additional program statements:

```
2000  FOR J=0 TO 255
2010  FOR K=0 TO 5
2020  POKE 6656+6*J+K,J
2030  NEXT K
2040  NEXT J
```

This program will generate every possible opcode and will repeat it six times, to allow the disassembler to treat it as having 1, 2, or 3 bytes. The block of opcodes occupies 1536 bytes, and is placed into the high end of the PET memory (for an 8K machine). A different area of memory may have to be used for other computers. The opcode generator is executed with the following command: RUN 2000.

When the program returns to the READY point, the disassembler is executed with the normal RUN command. Enter starting and ending addresses of 6656 and 8191, respectively. The disassembly that follows will include all 6502 opcodes and formats. Verification can be made by comparing the computer output with the MOS Technology manual (MCS 6500 Microcomputer Family Programming Manual, Second Edition, January, 1976, Appendix D). Note that the summary cards distributed for the 6502 may be lacking the ROR instruction from either the alphabetic table or the numeric opcode table. The PET computer output will scroll across the screen at a high rate. To slow down the display speed to a more convenient rate, hold down the RVS key during execution.

For the convenience of users of the PET computer desiring a cassette copy of the corrected program, I will be happy to send a copy to any reader who mails me a blank cassette and 50 cents for return postage.

Very truly yours,
Bennett Meyer

P.O. Box 575
White Plains, NY 10602

M6809 MEMORY TEST

Dear Dr. Dobbs: *Received: 78 June 15*

Congratulations on "Resident Memory Test Systems: With an Example for the 6800" in Number 25. Not because it was a good 6800 program—which it was not—but because of the reason it was not.

My 6800 based S-100 CPU would no more know what to do with a LDU, TFR or PSHS instruction than it could deal with a MOV or CALL instruction. Apparently, Mr. Ritter has given *Dr. Dobbs* a memory test program for the M6809, Motorola's new enhanced 6800.

Some of the features of the 6809 were described in the February 2, 1978, *Electronics*, p. 95; others are evident in Mr. Ritter's article. The program in the article shows that the 6809's instruction set is much more flexible to use than that of the 6800. It includes long branches, two, rather than one, stack registers and two index registers with a full set of load and store instructions for each, the ability to selectively push, pull and exchange all registers and compute and load effective addresses. The instructions vary from one to four bytes in length and appear to use mnemonics which are consistent with those for the 6800.

I hope that Motorola will continue to make software (and other data) for their 6809 available and that *Dr. Dobbs* will publish it.

Sincerely,
Kenneth Drexler

Three Embarcadero Center
Suite 2300
San Francisco, CA 94111

How embarrassing. Not being a 6800-er I failed to look over the code closely, assuming that if it came directly from Motorola, they must have it working, which indeed they do . . . on the 6809. I have learned that the 6809 should be available in the fourth quarter of 1978 at a very *reasonable price. It will be compatible with the entire 6800 chip family, have an on-chip oscillator/clock driver, 2 accumulators, 2 index registers and 2 stack pointers. The 6809 instruction set will be a superset of the 6800 instruction set. It will have 11 addressing modes, will be able to carry out 16 bit operations and . . . when you get one you'll be able to run a memory test from* Dr. Dobb's. *—TRW*

APPLE USERS CAN ACCESS DOW JONES INFORMATION SERVICE

News Release *Received: 78 June 14*

Apple Computer, Inc. has announced a new service which will provide owners of its computers with stock portfolio information and other financial services. Using a telephone link up, users of Apple II Computers will be able to dial Dow Jones & Co., Inc.'s Stock Quote Reporter Service for fifteen-minute-delayed stock and bond quotations. This information along with software provided by Apple will enable the user to determine current portfolio value, short and long term gains, and rate of return, among other things. At a later date, Apple II users will also be able to call up current news stories on companies in their user's portfolio.

The cost of the stock quote service will include a one-time fee of $25 plus a usage charge of $3 for the first three minutes plus 50 cents a minute thereafter for each usage session. It will be available from any authorized Apple dealer starting June 26th.

The Dow Jones & Co., Inc. Stock Quote Reporter Service consists of a large computer facility in New Jersey linked to a nationwide communications network. The service dispenses current stock quote information (delayed 15 minutes) and current and recent past (up to 90 days) financial news stories from the Dow Jones news media: *The Wall Street Journal, Barrons* and *The Dow Jones News Service* (Broad Tape).

Presently, users wanting access to network information have to purchase a terminal, such as a teletypewriter, and modem. For most users, each request to the network requires the entry of a series of statements into the terminal to log onto the system and thereafter to enter another sequence of statements to acquire each piece of information desired.

Like the terminal, Apple connects to the network via a modem and accesses the network by way of a password acquired when the user purchases the service. The password is presented to the Dow Jones network via the Apple II keyboard. However, the Apple brings to the network computer power which simplifies and performs the task of logging onto and processing information from the network much more efficiently. Moreover, because Apple is a computer, the amount of information processing that can be carried out is greatly increased. For example, with Apple Financial Evaluation Service II–a portfolio analysis program–the Apple automatically calculates long and short term profits and losses, rate of return, etc.

To log onto the Dow Jones network with Apple, the user loads Apple Financial I–a stock quotation program–then follows the instructions displayed by the program. These tell the user to dial the local telephone number of the Dow Jones Network and then insert the telephone receiver into a modem. Once a telephone link is established between the Apple II and the network via the modem, the user is only requested to enter his password to log on and gain information from the network.

Besides the logon feature, the Apple II is also very cost effective in its use of network time. With Apple Financial I, all symbols for which a quotation is desired can be entered into the Apple prior to logging on. Thereafter, Apple requests the desired quotes from the stored information without further keyboard input from the user. Thus the time on the network is always spent accessing and transferring data and never involves costly manual input typing time.

Apple Financial I permits the user to access quotes for up to 21 stocks per portfolio. This number is increased by adding additional cassettes, each of which can contain 21 stocks. Thus it is possible to have one large portfolio or a number of small portfolios. In this way an investment counselor can quickly access quotes for each of several clients simply by loading the appropriate cassette. Moreover, stocks in a given portfolio can easily be added or deleted.

With Apple Financial II, the amount of processing performed is much greater. The program permits the user to calculate his/her portfolio's current value (up to 16 stocks) and calculate short and long term gains and losses for each stock and the cumulative profit and/or loss as well. Thus, an investment counselor with a number of clients has instant access to the performance of each stock in any of the many portfolios he/she may be managing. Like Apple Financial I, this second program will be easy to use for the investor.

Other programs planned include a news retrieval routine which will permit the user to enter a company's stock symbol and receive a series of headlines associated with that symbol. The user then has the option of selecting a headline and receiving the text of that story on the Apple display screen or optional printer. Other such programs as this are planned for the future. See local dealers or contact Apple Computer, 10260 Brandley Dr., Cupertino, CA 95014; (408) 996-1010.

MICRO PROGRAMMING WORKSHOP

News Release *Received: 78 June 19*

WINTEK has added a new 5-day hands-on advanced programming workshop to their microprocessor training schedule for Fall '78. The course objective is to develop those skills required to plan, prepare, test, and document 6800/6801 microprocessor applications software. The course is offered November 27–December 1, 1978 in Lafayette, IN. For information call Jerilyn Williams, (317) 742-6802; or write WINTEK Corp., 902 N. 9th Street, Lafayette, IN 47904.

PERSONAL COMPUTER NEWS

News Release *Received 78 June 26*

The Personal Computer News is a monthly newsletter edited by Donald L. Wallace, and is published from Dayton, Ohio. PCN is dedicated entirely to a variety of reader services, and contains no paid advertisements. PCN features a regular news column detailing developments in the microcomputer industry and related technologies; objective product and software evaluations geared to the small businessman and the hobbyist alike; and two features due to commence this summer include a PCN-operational Software Exchange and a Trading Post classified advertisement section. In addition, a Software Sources listing culls the latest entrepreneurial offerings from the microcomputer media, and an Index to Computer-Related Articles cross-references features in the popular computerist magazines. Subscription Rates: $9/year in the US; $15/year in Canada and Mexico; and $24/year overseas. Personal Computer News, PO Box 425, Dayton, Ohio 45419.

FIX AND MANUAL CORRECTIONS FOR PROCESSOR TECH'S GPM

Received: 78 May 1

Users of Processor Technology's General Purpose Memory Module (GPM) board, part of the widely advertised Subsystem B, will find that they can not load or use the cassette tape version of the ALS-8 Assembler without first modifying the GPM board.

The problem is caused by the fact that the GPM board responds to memory addresses in the range of EOOOH to FFFFH, whether or not any ROM chips are installed on the board.

The following modification to the board disables this response and allows locating a standard 8K RAM board in this address space, so that the ALS-8 tape can be loaded. The modification does not affect operation of the CUTER monitor or of the PHANTOM startup feature to the CUTER monitor.

In the following instructions, all location descriptions assume the board to be held with the component side facing you and the edge connector down as shown in the card outline drawing (Fig. 5-1) in the GPM manual.

1. On the front (component side of the board), cut the thin trace connecting the pad located near the top end of U30 and the pad located under the body of R8.

Note 1: This is the first thin trace just above the top end of U31. This trace carries the card select signal from pin 12 of U30 to pin 15 of U32. By cutting the trace between these pads, the board can easily be returned to its original state should you ever install ROM chips for the ALS-8.

2. On the rear (copper side of the board), connect the following seven wire jumpers:

FROM	TO	
pin 10 of U30	pin 1 of U23	*Note 2*
pin 2 of U23	pin 12 of U19	
pin 10 of U19	pin 11 of U19	
top end of R3	pin 13 of U19	*Note 3*
pin 12 of U30	pin 11 of U23	*Note 4*
pin 10 of U23	pin 9 of U19	
pin 8 of U19	pin 15 of U32	*Note 5*

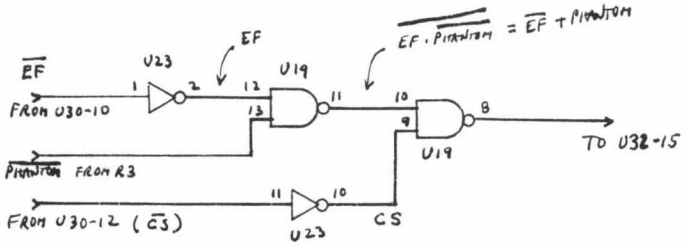
Note 2: Pin 10 of U30 is available at a pad below U30. This pad may be located by following the thin trace, on the rear of the board, from this pin down to the pad adjacent to the "1" digit of the number "100" located near the bottom edge connector.

Note 3: The PHANTOM signal is available at several locations on the board. The most convenient one from a wiring standpoint seems to be at the top end of R3.

Note 4: Pin 12 of U30 is available at the pad above U30 referred to in step 1.

Note 5: Pin 15 of U32 is available at the pad under the body of R8, referred to in step 1.

The overall circuit will look as in the attached logic diagram.



GPM FIX FOR ALS-8 1/9/78

The following GPM manual errors, discovered during the course of the above design, may also be of interest:

Table 3-1: The S1-8 description should read "allows phantom signals from auto start (on GPM) to S-100 bus". Switch settings should show S1-8 "ON" for an IMSAI, if auto start from the GPM is desired.

Figure 5-1: (card outline drawing) and Figure 5-2 (schematic) both identify U9 as a Prom B, a 2708. U9 is really ROM C, the CUTER ROM, a 9216B.

Figure 5-2: (schematic) has several other errors:

The signal at U30 pin 12 (CARD SELECT) should be shown as active low.

The signal at U30 pin 9 (ROM C) should be shown as active high.

The unlabeled dual input gate shown connected to bus pin 76, is really an inverter (U23) with pin 9 as input and pin 8 as output.

Ed Elizondo

6 Cypress Court
East Windsor, N.J. 08520

2 NEW F-8 PRODUCTS

News Release *Received: 78 March 24*

Comptronics has introduced two new F-8 products for the serious hobbyist and design engineer.

The first product is an F-8 CPU board compatible with the S-100 bus. Complete with 3850 CPU and 3853 SMI, the unit provides sockets for 2K of EPROM monitor, two PIO sockets, and connections for six I/O ports. The board has 64 bytes of scratch pad RAM, and a fully buffered data bus. The Model F-8S100 sells for $239 as a kit, or $275 assembled.

The second product is an F-8 microcomputer with keyboard and six digit display. The unit provides audio interface and speaker compatible with the on-board KD-BUG (3856) music routine, 2K of RAM expandable through an S-100 connector, and 1K of EPROM with 4 additional 2708 sockets. Model KD80 sells for $375 as a kit, and $425 assembled. Contact: Don R. Swanke, Comptronics, 19824 Ventura Blvd., Woodland Hills, CA 91364; (213) 340-8843.

Custom File Routines for ALS-8

BY LARRY A MAREK
6008 East Lake Dr., Apt. 3B
Lisle, IL 60532

Here are a set of routines I have worked out to take advantage of the tape routines in CUTER/SOLOS and the ALS-8 assembler. The ALS-8 assembler has the ability to assemble source code given to it by an outside driver in addition to the normal assembling from a file in memory. CUTER/SOLOS has the ability to appear like it is reading a byte from a tape whenever asked. "Appears" is important because the assembler could not perform its operations fast enough between bytes if the tape was actually in constant movement. By combining the two features, assemblies can be made of HUGE source files. The only restraint is that enough memory be available to hold the produced object code.

To use these routines change line 0006 of the listing to agree with your CUTER/SOLOS starting address, then assemble the program anywhere in your system.

Before a tape file can be used, CUTER/SOLOS must "OPEN" the tape file, so make "OPEN" a custom command using the address associated with the label "OPEN" in your assembly.

Example: CUSTE /OPEN/ 0000

The command has three forms: "OPENW m n", "OPENR m n", and "OPEN". "OPENW m n" prepares a file that will be written to. Optional parameter m can specify the tape unit 1, 2, or if left blank, will default to unit #1. (Note—two tape files may be open at the same time, but one must be a read and the other a write). Optional parameter n specifies the baud rate of the tape—1 or unspecified is normal 1200 baud, 2 is 300 baud (Note—open files can not have different baud rates, also the last OPEN command used will control the rate). "OPENR m n" prepares a file that will be read from, options m and n are the same as in "OPENW". Plain "OPEN" will display which tape files are open and if they are a read or a write file.

When a tape file is emptied or filled CUTER/SOLOS must be allowed to close the tape file. Thus, another custom command in ALS-8 should be set up with the address of the label "CLOSE" in your assembly.

Example: CUSTE /CLOS/ 0106

The two forms of this command are: "CLOSW" to close a write tape file, and "CLOS" to close a read tape file.

Before the assembler can assemble from the tape routines, a file must be recorded out on tape in byte format. To do this another custom command in ALS-8 with the address corresponding to label "FSAV" in your assembly is used.

Example: CUSTE /FSAV/ 015F

The command has only one form "FSAV". This will start the tape and record the source file in the byte format. A write tape file must be already open and when "FSAV" is done the file will still be open. The tape file remains open so that large programs need not be in memory all at one time. Some words of caution here—when you are "FSAVing" out a large file—DO NOT—close the tape file until after the very last section has been "FSAVed".

The "CLOS" command will cause any left over bytes (byte format is actually 256 byte blocks) to be saved out on tape along with an end of tape file marker. Also, the very last line of the source file must be an "END" psuedo-op. This is so the assembler can tell when to begin its second pass. Because VERY much of the tape is blank between 'byte blocks' "FSAVed" tape files will use about 3-4 times as much tape as the same file saved by a normal CUTER/SOLOS "SAVE". This makes these tape files V E R Y slow.

The last custom command should be associated with your address for the label "BUILD".

Example: CUSTE /BUIL/ 0180

"BUIL" is the opposite of "FSAV". It will read byte format tape files into the current file. By specifying an address after the "BUIL" command the tape file will be read into the current memory file until the hex address is reached, then just the remaining bytes needed to finish a line are read. (If no end is specified the whole file will be read in.) Thus "BUIL" and "FSAV" can be used to pass very large source files amongst hobbyists without regard to memory limitation. Each user "BUILds" a memory file according to his/her available memory.

The last thing to define in ALS-8 is the tape file input driver with your address for "ENTRY".

Example: IODR /CASSI/ 0091 0000

ALS-8 is now ready to assemble any size program for you. ALS-8 does a carriage return and a line feed at the end of its first pass. This is your cue to rewind the tape. The motor will turn on automatically. When a tape is fully rewound strike any key on the keyboard to continue the assembly.

Remember, all routines work with the file name as found in the "CURRENT" file in ALS-8. As tape files are written they will get the current file name. When reading tape files the current file name must match the name used to write the tapes.

Here are some "quirks" I found in CUTER/SOLOS as I was writing these routines. Tape unit two is 42H not 40H as described in the CUTER/SOLOS manual. A big mistake exists in CUTER/SOLOS—if a call is made to "FOPEN" when the designated file is already open the stack is "POPed" once too many times! Instead of returning with the carry bit set, the processor takes off to whatever garbage is on the stack. (I spent several hours one night tracking this problem only to read about it the next day in an old "SOLUS" newsletter). This explains the point and push in lines 0072 and 0073 of this source file. By pushing the error handler address on the stack the extra pop will put it in the P.C. register and the error will be handled by "TAPER".

For those of you using ALS-8 that is now available on a CUTS cassette I have found a "bug" in it. The problem is in the "ASSM AAAA BBBB" form of the assemble command. The assembler is supposed to assemble the code for start address "AAAA" but put the object code at address "BBBB". I have only thought of one real use for this—when you are just trying an assembly (to get the syntax errors out) before you break a large source up into byte format. The cassette version is supposed to handle this and it does okay until it hits and "ORG" pseudo-op. Then BOTH the "AAAA" and the "BBBB" address become that of the "ORG" value. In both the older PT #1 and the old ROM version of ALS-8 an "ORG" statement just ADDED to the "BBBB" address. The fix is relatively simple—read in the cassette then make these changes:

```
ADDR      CASSETTE VALUE      CHANGE TO

ECBF            22                7D          MOV A,L
ECC0            F1                C3          JMP PATCH
ECC1            D0                F0
ECC2            C9                F9

F9F0            00                93          SUB L
F9F1            00                5F          MOV E,A
F9F2            00                7C          MOV A,H
F9F3            00                9A          SBB D
F9F4            00                57          MOV D,A
F9F5            00                2A          LHLD BBUF+2
F9F6            00                F1
F9F7            00                D0
F9F8            00                19          DAD D
F9F9            00                22          SHLD BBUF+2
F9FA            00                F1
F9FB            00                D0
F9FC            00                C9          RET
```

CORRECTION ON MODEM ARTICLE

Dear Tom: *Received: 78 June 12*

Received your letter and complimentary copies of the Jun/July issue of *Dr. Dobb's Journal* today. It is very satisfying to see my article in print. Unfortunately, there seems to be an error in one section of the article.

The last paragraph in col. 1 on page 6 should read: ". . . the system must, at a minimum, provide for ring detection (*the Hayes unit provides this while IDS does not*)."

The significance of the correction is that it establishes the foundation for the claim that the Hayes unit can be directly connected to the phone lines without an intervening DAA. This claim is basically correct (except for the required certification and certain other technical details that I allude to in the article) and I certainly don't deny it.

I noticed on page 44 of Issue 6 the article "Computer Network Poll" reports reasonable success with the Hayes unit. The manual I used as reference on the Hayes unit related to the original unit. I note the article refers to a revision 3 modem board with anticipation of revision 4 shortly. On the one hand, this seems to bear out my reservations about the board. On the other hand, it seems Hayes, with help from users of his board, is making an effort to correct its deficiencies.

Thanks for accepting my article for publication and keep up the good work. Your previous issue (Vol 3, Issue 5) was really outstanding.

Sincerely,
Tim J. Pugh, Jr.

9655-M Homestead Court
Laurel, MD 20810

```
0000                    ;  CUSTOM FILE ROUTINES FOR ALS-8
0001                    ;  USING CUTER TAPE I/O.
0002                    ;
0003                    ;  BY: LARRY A. MAREK    APRIL 4,1978
0004                    ;
0005
0006 CUTER  EQU  0C000H  CUTER START
0007                    ;
0008 FOPEN  EQU  CUTER+07H   FILE OPEN CALL
0009 FCLOS  EQU  CUTER+0AH   FILE CLOSE CALL
0010 RDBYT  EQU  CUTER+0DH   READ BYTE CALL
0011 WRBYT  EQU  CUTER+10H   WRITE BYTE CALL
0012 SINP   EQU  CUTER+01FH  SYSTEM INPUT
0013 DHEAD  EQU  CUTER+82CH  DUMMY HEADER LOC
0014 HTYPE  EQU  CUTER+822H  TYPE BYTE IN HEADER
0015 TSPD   EQU  CUTER+80DH  SPEED CONTROL BYTE
0016 RUNIT  EQU  CUTER+814H  READ UNIT HOLDER
0017 WUNIT  EQU  CUTER+815H  WRITE UNIT HOLDER
0018                    ;
0019 S1200  EQU  0       1200 BAUD
0020 S300   EQU  20H     300 BAUD
0021 UNIT1  EQU  80H     TAPE #1 (CUTER OKAY)
0022 UNIT2  EQU  42H     TAPE #2 (ERROR IN CUTER)
0023 TAPPT  EQU  0FAH    TAPE CONTROL PORT
0024                    ;
0025 FILE0  EQU  0D000H  CURRENT FILE NAME
0026 BOFP   EQU  0D005H  BEGIN OF FILE
0027 EOFP   EQU  0D007H  END OF FILE
0028 BBUF   EQU  0D0EFH  BINARY BUFFER
0029 IBUF   EQU  0D1E4H  INPUT BUFFER
0030 SWCH1  EQU  0D0FDH  CONTROL SWITCH
0031 ALS8   EQU  0E060H  ALS-8 RETURN
0032 ACHK   EQU  0E369H  ADDR CHECK & INCR
0033 SCRN   EQU  0E380H  ASCII LINE OUTPUT
0034 WHAT   EQU  0E7DDH  OUTPUT ERROR
0035 FCHK   EQU  0E6D3H  FILE CHECK & UPDATE
0036 CRLF   EQU  0E216H  CR/LF OUTPUT
0037                    ;
0038 EOF    EQU  1       END-OF-FILE MARKER
0039 CR     EQU  0DH     ASCII C/R
0040 NULL   EQU  0       ASCII NULL
0041                    ;  THE OPEN FILE COMMAND, "OPEN" GIVES THE
0042                    ;  FILE STATUS, "OPENW" OPENS A WRITE FILE,
0043                    ;  "OPENR" OPENS A READ FILE.
0044                    ;  SECOND AND THIRD PARM'S IF GIVEN
0045                    ;  SPECIFY UNIT THEN SPEED (DEFAULT IS
0046                    ;  UNIT #1, 1200 BAUD.
0047                    ;
0048
0049 OPEN   LHLD BBUF    GET PARAM.
0050        CALL NUMBR   CONVERT TO UNIT
0051        LDA  IBUF+4  LOOKING FOR...
0052        CPI  'W'     A WRITE FILE?
0053        JZ   WSTOR   STORE WRITE FILE
0054        CPI  'R'     A READ FILE?
0055        JNZ  TELLEM  NEITHER TELL WHAT IS
0056        MOV  A,B     GET UNIT
0057        STA  PUNIT   PUT IN READ UNIT
0058        PUSH PSW     SAVE ON STACK TOO
0059 SPED   LHLD BBUF+2  2ND PARAM
0060        CALL NUMBR   CONVERT TO UNIT
0061        MOV  A,B     OUT OF B
0062        ANI  7DH     STRIP BIT 8 & 2
0063        RPC          BIT 7 TO BIT 6
0064        STA  TSPD    TELL CUTER
0065        LXI  H,FILE0 SHOW FILE NAME
0066        LXI  D,DHEAD  TO HEADER
0067        MVI  B,5     CHARS IN NAME
0068        CALL MVNAM   MOV FILE NAME
0069        XRA  A       CLEAR A
0070        STAX D       EMPTY BYTE
0071        MVI  A,'S'+80H  NON-EXEC SOURCE
0072        INX  D       POINTS TO TYPE-BYTE

0000 2A EF D0
0003 CD 81 00
0006 3A E8 D1
0009 FE 57
000B CA 79 00
000E FE 52
0010 C2 41 00
0013 78
0014 32 14 C8
0017 F5
0018 2A F1 D0
001B CD 81 00
001E 78
001F E6 7D
0021 0F
0022 32 0D C8
0025 21 00 D0
0028 11 2C C8
002B 06 05
002D CD 4F 01
0030 AF
0031 12
0032 3E D3
0034 13
```

```
0035 12
0036 F1
0037 11 43 01
003A D5
003B 21 2C C8
003E CD 07 C0
0041
0041 CD 16 E2
0044 3A 14 C8
0047 21 EC 01
004A FE 42
004C CA 57 00
004F 21 D7 01
0052 FE 80
0054 C2 5D 00
0057 CD 16 E2
005A CD 80 E3
005D 3A 15 C8
0060 21 17 02
0063 FE 42
0065 CA 70 00
0068 21 01 02
006B FE 80
006D C2 60 E0
0070 CD 16 E2
0073 CD 80 E3
0076 C3 60 E0
0079 78
007A 32 15 C8
007D F5
007E C3 18 00
0081 7D
0082 B4
0083 06 80
0085 C8
0086 FE 01
0088 C8
0089 06 42
008B FE 02
008D C8
008E C3 DD E7
0091
0091
0091
0091
0091 E5
0092 D5
0093 C5
0094
0094 3A FD D0
0097 B7
0098 C2 A5 00
009B 3C
009C 32 FD D0
009F
009F 3A 14 C8
00A2 CD 0D C0
00A5
00A5
00A5
00A5
00A5
00A5 21 DF D1
00A8 3A 14 C8
00AB E5
00AC CD 0D C0
00AF E1
00B0 FA CF 00
00B3 DA 43 01
00B6 77
00B7 FE 0D
00B9 CA C5 00
00BC FE 20
00BE DA A8 00

0073          STAX   D       PUT IT IN TOO
0074          POP    PSW     RE-GET UNIT
0075          LXI    D,TAPER  DUE TO ERROR IN...
0076          PUSH   D       OUTER TO TAPER THIS WAY
0077          LXI    H,DHEAD  GIVE NAME...
0078          CALL   FOPEN   TELL OUTER
0079 TELLEM   EQU    $
0080          CALL   CRLF    NEATER WITH A BLANK
0081          LDA    RUNIT   GET READ FILE
0082          LXI    H,R2OPN  THIS READ
0083          CPI    UNIT2   FILE TWO IN READ?
0084          JZ     ROUT    TELL READ #2
0085          LXI    H,R1OPN  THIS READ
0086          CPI    UNIT1   FILE ONE IN READ?
0087          JNZ    TRYW    NEITHER TRY WRITE
0088 ROUT     CALL   CRLF    WE WANT A NEW LINE
0089          CALL   SCRN    OUTPUT IT
0090 TRYW     LDA    WUNIT   ANY WRITES?
0091          LXI    H,W2OPN  THIS WRITE
0092          CPI    UNIT2   IS IT TWO?
0093          JZ     WOUT    YES TELL 'EM
0094          LXI    H,W1OPN  THIS WRITE
0095          CPI    UNIT1   IS IT ONE?
0096          JNZ    ALS8    NOBODY - GET OUT
0097 WOUT     CALL   CRLF    NEW LINE
0098          CALL   SCRN    GIVE IT OUT
0099          JMP    ALS8    WERE DONE!
0100 WSTOR    MOV    A,B     MOVE UNIT
0101          STA    WUNIT   PUT AWAY
0102          PUSH   PSW     ON STACK TOO
0103          JMP    SPED    FIND OUT SPEED
0104 NUMBR    MOV    A,L     ONE SIDE...
0105          ORA    H       AGAINST OTHER
0106          MVI    B,UNIT1  ASSUME ONE
0107          RZ     .       NO NUMBER DEFAULT
0108          CPI    1       UNIT ONE?
0109          RZ     .       YES - USE IT
0110          MVI    B,UNIT2  SWITCH TO TWO
0111          CPI    2       UNIT TWO?
0112          RZ     .       WAS UNIT TWO
0113          JMP    WHAT    ONLY ONE OR TWO
0114 *
0115 * THIS IS THE ENTRY POINT FOR THE "ASSI"
0116 * INPUT DRIVER
0117 *
0118 ENTRY    EQU    $
0119          PUSH   H       SAVE REG'S FOR...
0120          PUSH   D       ASSEMBLER
0121          PUSH   B
0122 ENT2     EQU    $
0123          LDA    SWCH1   ENTRY INDICATOR
0124          ORA    A       TEST IT
0125          JNZ    LINER   IF NOT FIRST ACCESS
0126          INR    A       ALSO USE AS PASS IND.
0127          STA    SWCH1   FIX ENTRY PATH
0128 SET2     EQU    $
0129          LDA    RUNIT   READ FILE...
0130          CALL   RDBYT   TO THROW AWAY FIRST BYTE
0131 *
0132 * LINER READS A LINE OF SOURCE AND PLACES
0133 * IT IN IBUF FOR ASSMBLER
0134 *
0135 LINER    EQU    $
0136          LXI    H,IBUF-5  POINT H&L
0137 CHRS     LDA    RUNIT   GET FILE NUMBER
0138          PUSH   H       RDBYT WILL LOSE H
0139          CALL   RDBYT   GET A CHAR
0140          POP    H       RESTORE IBUF POINTER
0141          JM     REWIND  END OF FILE
0142          JC     TAPER   RATS! TAPE ERROR
0143          MOV    M,A     PLACE CHAR
0144          CPI    CR      END OF LINE?
0145          JZ     LINDN   LINE IS DONE
0146          CPI    ' '     CONTROL CHAR?
0147          JC     CHRS    DON'T PUT IT IN

00C1 23
00C2 C3 A8 00
00C5
00C5
00C5
00C5
00C5
00C5 3A 14 C8
00C8 CD 0D C0
00CB
00CB C1
00CC D1
00CD E1
00CE C9
00CF
00CF
00CF
00CF
00CF
00CF
00CF
00CF
00CF 3A FD D0
00D2 FE 01
00D4 C2 CB 00
00D7 3C
00D8 32 FD D0
00DB CD 25 01
00DE 3A 14 C8
00E1 D3 FA
00E3 CD 1F C0
00E6 CA E3 00
00E9 AF
00EA D3 FA
00EC
00EC
00EC
00EC 21 00 D0
00EF 11 2C C8
00F2 06 05
00F4 CD 4F 01
00F7 21 2C C8
00FA 3A 14 C8
00FD CD 07 C0
0100 DA 43 01
0103 C3 9F 00
0106
0106
0106
0106
0106
0106
0106 2A EF D0
0109 CD 81 00
010C 3A E8 D1
010F FE 57
0111 CA 2F 01
0114 3A 14 C8
0117 B8
0118 C2 DD E7
011B CD 25 01
011E AF
011F 32 14 C8
0122 C3 60 E0
0125 3A 14 C8
0128 CD 0A C0
012B DA 43 01
012E C9
012F 3A 15 C8
0132 B9
0133 C2 DD E7
0136 CD 0A C0
0139 DA 43 01

0148          INX    H       OKAY BUMP POINTER
0149          JMP    CHRS    THEN MORE CHARS
0150 *
0151 * A LINE IS DONE, MUST THROW AWAY THE CHAR
0152 * COUNT THAT FOLLOWS EACH CARRIAGE RETURN
0153 *
0154 LINDN    EQU    $
0155          LDA    RUNIT   FILE #
0156          CALL   RDBYT   READ ANOTHER BYTE
0157 OUT      EQU    $
0158          POP    B       RESTORE ASSM'S...
0159          POP    D       REGISTERS
0160          POP    H
0161          RET    .       ASSMBLER TAKES OVER
0162 *
0163 * REWIND CHECKS PASS COUNT AND TAKES
0164 * APPROPRIATE ACTION
0165 *
0166 * - NOTE - ASSEMBLER DOES A CR/LF TO INDICATE
0167 *   WHEN TO REWIND THE TAPE UNIT, WHEN THE
0168 *   TAPE IS REWOUND HIT ANY KEY TO CONT.
0169 *
0170 REWIND   EQU    $
0171          LDA    SWCH1
0172          CPI    1       WAS THIS PASS ONE?
0173          JNZ    OUT     NOPE - ALL DONE
0174          INR    A       INCREASE PASS COUNT
0175          STA    SWCH1   AND HIDE AWAY
0176          CALL   RCLOS   TEMPORARY CLOSE
0177          LDA    RUNIT   TAPE #
0178          OUT    TAPPT   TURN ON THAT TAPE
0179 WAIT     CALL   SINP    THEN WAIT FOR...
0180          JZ     WAIT    KEY-PUSH TO CONTINUE
0181          XRA    A       CLEAR A
0182          OUT    TAPPT   TO TURN OFF TAPES
0183 *
0184 * RAN OUT OF FILE, NOW THE FILE MUST
0185 * BE RE-OPENED.
0186 *
0187          LXI    H,FILE0  FILE NAME
0188          LXI    D,DHEAD  DUMMY HEADER LOC
0189          MVI    B,5     CHAR'S IN NAME
0190          CALL   MVNAM   MOVE FILE NAME
0191          LXI    H,DHEAD  POINT H&L TO IT
0192          LDA    RUNIT   READ FILE #
0193          CALL   FOPEN   OPEN THE FILE
0194          JC     TAPER   ERROR SOMEPLACE
0195          JMP    SET2    TAKE IT FROM THE TOP
0196 *
0197 * CLOSE FILE COMMAND, "CLOSW" CLOSES
0198 * A WRITE FILE, "CLOS" CLOSES A READ.
0199 * 2ND PARAM TELLS WHICH UNIT TO CLOSE
0200 *
0201 CLOSE    EQU    $
0202          LHLD   BBUF    GET PARAM
0203          CALL   NUMBR   CONVERT IT
0204          LDA    IBUF+4   GET MODIFIER
0205          CPI    'W'     WRITE FILE?
0206          JZ     WCLOS   ELSE DEFALT TO READ
0207          LDA    RUNIT   IS READ UNIT...
0208          CMP    B       SAME AS COMMAND?
0209          JNZ    WHAT    NOPE!, ELSE...
0210          CALL   RCLOS   START WITH THIS
0211          XRA    A       THEN CLEAR A
0212          STA    RUNIT   AND UNIT
0213          JMP    ALS8    THAT'S IT
0214 RCLOS    LDA    RUNIT   FILE TO CLOSE
0215          CALL   FCLOS   LET OUTER DO IT
0216          JC     TAPER   ALREADY CLOSED?
0217          RET    .       IS OKAY
0218 WCLOS    LDA    WUNIT   WRITE FILE #
0219          CMP    B       SAME AS COMMAND?
0220          JNZ    WHAT    WRONG UNIT?
0221          CALL   FCLOS   GIVE TO OUTER
0222          JC     TAPER   WE WENT WRONG?
```

```
013C AF          0223          XRA   A       CLEAR A
013D 32 15 C8    0224          STA   WUNIT   AND WRITE UNIT
0140 C3 60 E0    0225          JMP   ALS8    THAT'S THAT
0143             0226 *
0143             0227 * TAPE ERROR HANDLER
0143             0228 *
0143             0229 TAPER    EQU   $
0143 CD 16 E2    0230          CALL  CRLF    START NEW LINE
0146 21 C6 01    0231          LXI   H,ERMES  POINT
0149 CD 80 E3    0232          CALL  SCRN    OUTPUT
014C C3 60 E0    0233          JMP   ALS8    AND RETURN
014F             0234 *
014F             0235 * MOVE THE CURRENT FILE NAME INTO CUTER'S
014F             0236 * AREA AND REPLACE SPACE'S WITH NULLS
014F             0237 *
014F 7E          0238 MVNAM    MOV   A,M     PICK UP CHAR
0150 FE 20       0239          CPI   ' '     ALS-8 SPACE FILLS
0152 C2 57 01    0240          JNZ   CHARS   NO GOES AROUND
0155 3E 00       0241          MVI   A,NULL  CUTER NULL FILLS
0157 12          0242 CHARS    STAX  D       PUT IN DHEAD
0158 23          0243          INX   H       BUMP H
0159 13          0244          INX   D       AND D
015A 05          0245          DCR   B       DO ALL...
015B C2 4F 01    0246          JNZ   MVNAM   FIVE CHAR'S
015E C9          0247          RET   .       THEN GO BACK
015F             0248 *
015F             0249 * FSAVE - COMMAND TO PUT CURRENT FILE OUT
015F             0250 * TO TAPE IN "BYTE" FORMAT
015F             0251 *
015F             0252 *  - NOTE - FILE MUST BE OPEN FOR A WRITE
015F             0253 *     AND WILL REMAIN OPEN (IN CASE OF MULTI-
015F             0254 *     SECTION FILE)
015F             0255 *
015F             0256 FSAV     EQU   $
015F 2A 07 D0    0257          LHLD  EOFP    GET END OF FILE
0162 22 F1 D0    0258          SHLD  BBUF+2  IN "TO" COUNTER
0165 2A 05 D0    0259          LHLD  BOFP    BEG'N OF FILE
0168 22 EF D0    0260          SHLD  BBUF    IN "FROM" COUNTER
016B 46          0261 FLOUT    MOV   B,M     GET CHAR
016C 3A 15 C8    0262          LDA   WUNIT   GET UNIT
016F E5          0263          PUSH  H       SAVE H&L
0170 CD 10 C0    0264          CALL  WRBYT   LET CUTER WRITE IT
0173 E1          0265          POP   H       RESTORE H&L
0174 DA 43 01    0266          JC    TAPER   ERROR SOMEPLACE
0177 CD 69 E3    0267          CALL  ACHK    BUMP & CHECK POINTERS
017A D2 6B 01    0268          JNC   FLOUT   NOT DONE
017D C3 60 E0    0269          JMP   ALS8    ALL DONE
0180             0270 *
0180             0271 * "BUILD" COMMAND TO READ "BYTE" FORMAT
0180             0272 *  SOURCE FILES INTO THE CURRENT FILE UNTIL
0180             0273 *  THE GIVEN MEMORY LIMIT IS REACHED, IF NO
0180             0274 *  LIMIT IS GIVEN WHOLE FILE IS READ IN.
0180             0275 *
0180 2A EF D0    0276 BUILD    LHLD  BBUF    GET PARAM
0183 7D          0277          MOV   A,L     GET ONE SIDE
0184 B4          0278          ORA   H       TEST AGAINST OTHER
0185 C2 89 01    0279          JNZ   LIMIT   NUMBER WAS GIVEN
0188 2B          0280          DCX   H       MAKES LIMIT FFFF
0189 22 F1 D0    0281 LIMIT    SHLD  BBUF+2  IN "TO" COUNTER
018C 2A 05 D0    0282          LHLD  BOFP    BEG'N OF FILE
018F 22 EF D0    0283          SHLD  BBUF    IN "FROM" COUNTER
0192 3A 14 C8    0284 BLOOP    LDA   RUNIT   FILE TO READ
0195 E5          0285          PUSH  H       SAVE H&L
0196 CD 0D C0    0286          CALL  RDBYT   GET A FILE CHAR
0199 E1          0287          POP   H       RESTORE POINTER
019A FA C0 01    0288          JM    ENDER   END OF FILE?
019D DA 43 01    0289          JC    TAPER   CRC ERROR?
01A0 77          0290          MOV   M,A     CHAR OK, STORE IT
01A1 CD 69 E3    0291          CALL  ACHK    BUMP POINTER, CHECK END
01A4 D2 92 01    0292          JNC   BLOOP   NOT END YET
01A7 2B          0293          DCX   H       BACK UP A BYTE
01A8             0294 *
01A8             0295 * MEMORY LIMIT REACHED, FOR A CORRECT FILE
01A8             0296 * THE REMAINDER OF THE LINE MUST BE READ IN
01A8             0297 *
01A8 23          0298 FINLN    INX   H       NEXT CHAR
01A9 3A 14 C8    0299          LDA   RUNIT   FILE TO READ
01AC E5          0300          PUSH  H       SAVE H&L
01AD CD 0D C0    0301          CALL  RDBYT   GET A CHAR
01B0 E1          0302          POP   H       RESTORE H
01B1 FA C0 01    0303          JM    ENDER   END SLIPPED IN
01B4 DA 43 01    0304          JC    TAPER   CRC ERROR
01B7 77          0305          MOV   M,A     STORE CHAR
01B8 FE 0D       0306          CPI   CR      END OF LINE MARKER
01BA C2 A8 01    0307          JNZ   FINLN   MORE TO LINE
01BD 23          0308 ENDFL    INX   H       LAST CHAR...
01BE 36 01       0309          MVI   M,EOF   IS MARKER
01C0 2A 05 D0    0310 ENDER    LHLD  BOFP    BEG'N OF FILE
01C3 C3 D3 E6    0311          JMP   FCHK    NOW UPDATE FILE
01C6             0312 *
01C6             0313 *
01C6 21 21 20 54 0314 ERMES    ASC   #!! TAPE ERROR !!#
     41 50 45 20
     45 52 52 4F
     52 20 21 21
01D6 0D          0315          DB    CR
01D7             0316 *
01D7             0317 *
01D7 55 4E 49 54 0318 R1OPN    ASC   #UNIT 1 OPEN FOR READ#
     20 31 20 4F
     50 45 4E 20
     46 4F 52 20
     52 45 41 44
01EB 0D          0319          DB    CR
01EC             0320 *
01EC             0321 *
01EC 55 4E 49 54 0322 R2OPN    ASC   #UNIT 2 OPEN FOR READ#
     20 32 20 4F
     50 45 4E 20
     46 4F 52 20
     52 45 41 44
0200 0D          0323          DB    CR
0201             0324 *
0201             0325 *
0201 55 4E 49 54 0326 W1OPN    ASC   #UNIT 1 OPEN FOR WRITE#
     20 31 20 4F
     50 45 4E 20
     46 4F 52 20
     57 52 49 54
     45
0216 0D          0327          DB    CR
0217             0328 *
0217             0329 *
0217 55 4E 49 54 0330 W2OPN    ASC   #UNIT 2 OPEN FOR WRITE#
     20 32 20 4F
     50 45 4E 20
     46 4F 52 20
     57 52 49 54
     45
022C 0D          0331          DB    CR
022D             0332 *
022D             0333 *

ACHK    E369     ALS8    E060     BBUF    D0EF     BLOOP   0192
BOFP    D005     BUILD   0180     CHARS   0157     CHRS    00A8
CLOSE   0106     CR      000D     CRLF    E216     CUTER   C000
DHEAD   C82C     ENDER   01C0     ENDFL   01BD     ENT2    0094
ENTRY   0091     EOF     0001     EOFP    D007     ERMES   01C6
FCHK    E6D3     FCLOS   C00A     FILE0   D000     FINLN   01A8
FLOUT   016B     FOPEN   C007     FSAV    015F     HTYPE   C822
IBUF    D1E4     LIMIT   0189     LINDN   00C5     LINER   00A5
MVNAM   014F     NULL    0000     NUMBR   0091     OPEN    0000
OUT     00CB     R1OPN   01D7     R2OPN   01EC     RCLOS   0125
RDBYT   C00D     REWIN   00CF     ROUT    0057     RUNIT   C814
S1200   0000     S300    0020     SCRN    E380     SET2    009F
SINP    C01F     SPED    0018     SWCH1   D0FD     TAPER   0143
TAPPT   00FA     TELLE   0041     TRYW    005D     TSPD    C80D
UNIT1   0080     UNIT2   0042     W1OPN   0201     W2OPN   0217
WAIT    00E3     WCLOS   012F     WHAT    E7DD     WOUT    0070
WRBYT   C010     WSTOR   0079     WUNIT   C815
```

COMMON PILOT: Progress Report

BY LARRY KHERIATY and GEORGE GERHOLD
Micropi
2445 N. Nugent
Lummis Island, WA 98262

Increasing interest in using microcomputers as teaching tools in combination with increasing realization that BASIC is not a suitable language for most teaching applications has created a demand for microcomputer implementations of PILOT. This article is a status report on a number of microcomputer implementations of PILOT and a description of the extended version of PILOT which has been proposed as a national standard for the language. This extended version is called COMMON PILOT.

The original version of PILOT (often called CORE PILOT or PILOT 73) was created at the San Francisco Medical Center. The language was designed to work on Datapoint terminals in a stand-alone configuration. The simplicity of CORE PILOT appealed to users who wanted a language for introducing children to programming. The philosophy behind CORE PILOT appealed to users who wanted a language free of assumptions about proper teaching strategies. The latter group added extensions to CORE PILOT to convert it into a powerful tool for Computer-Assisted Instruction and for other types of interactive computing with an important dialogue component. Fortunately most of the extensions were influenced by an extended version written at Western Washington University; therefore the resulting dialects of PILOT turned out to be similar in form, but dissimilar in details of syntax. Under these circumstances standardization seems to be both an achievable and a worthwhile goal. The National Library of Medicine has assumed sponsorship of the task. Initially it appeared that there was a sharp division between the supporters of simplicity (which seemed to be essential for small machine implementations) and the supporters of power (which is essential for acceptance as a general CAI language). One of our reasons for attempting a microcomputer implementation of the WWU extended version of PILOT was to demonstrate that the division was based on false premises. The microcomputer implementation, which was done under the sponsorship of Western Washington University, was so successful that the resulting version was used as the starting point for a detailed proposal for the national standard. COMMON PILOT, as described herein, is a refinement of that original proposal; the refinements are based on extensive discussions with other PILOT users and on our experience in preparing a version for commercial distribution. We anticipate that this version will be backed by a number of institutions at the standardization meetings.

At this writing (May 1978) we have completed the system for a 6800 processor, and we are testing a version which takes full advantage of the calculator board and the multi-user capability of the Southwest Technical Products 6800/2 computer. This version will deliver COMMON PILOT on a 4 terminal system. Hardware requirements include a 32K 6800/2 computer, 4 serial interfaces, a dual floppy disk, a calculator board, the multi-user board, and 4 CRT's. As the total cost of this hardware is about $6000, the system opens the way for schools to begin using CAI at relatively low cost. Southwest Technical Products will be distributing systems software for this version of COMMON PILOT starting in the summer of 1978. The software will function on a single or on multiple-terminal systems.

In addition we are translating the system software into 8080/Z80 code, and we are testing COMMON PILOT written in PASCAL. PASCAL source copies will be available by Fall 1978 for $300.

Description of COMMON PILOT

At least two microcomputer versions of PILOT have been described in the microcomputer journals. These versions are actually subsets of CORE PILOT. COMMON PILOT includes numerous features which put it into a different class from CORE PILOT. Perhaps the most significant feature of COMMON PILOT is the fact that the PILOT program is interpreted from disk. The system automatically pages the program into memory. While only a few lines are in a buffer at one time, there are no restrictions on branching ranges, that is, branches are not restricted to that portion resident in memory. As a result there is virtually no limit on the length of the CAI program. A typical PILOT program to provide one hour of quality instruction may be 1500-2500 lines long and may occupy 60K-90K on a disk. Workspace in the memory is reserved for storage of variables, arrays, and strings. COMMON PILOT uses floating point variables, one- and two-dimensional numeric arrays, and varying length character strings (maximum length 999).

Language Summary

Labels; 1 to 6 characters (labeled line begins with *)

Operation Codes	Remarks
PR: option list	U – force to uppercase S – eliminate spaces E – escape allowed L – force to lowercase G – goto allowed W – clear label table blank option list leaves prior options unchanged
R: remark	ignored at execution time
T: text	type text after substituting current values for string and numeric variables
: text	continuation of previous type
A:	accept answer into answer buffer (optionally into numeric and/or string variable)
M: pattern	match answer buffer with pattern

J:	destination	jump to label, last A: , next M: , or next PR:
U:	label	use subroutine
E:		end subroutine or course
XI:	string variable	execute string variable as PILOT instruction
FI:	record, string variable	load variable from disk
FO:	record, string variable	write variable onto disk
D:	variable (N)	create 1 dimensional array
	variable (N,M)	create 2 dimensional array
	variable $ (N)	create string variable
C:	variable = expression	assignment
C:	/variable edit option	text editing of string variable

Edit Options:
- C – capitalize first letter
- U – translate string to uppercase
- /xy/ – replace all occurrences of character x by character y
- /x/ – remove all occurrences of character x

Match Pattern Symbols:
- * – match any single character
- % – match space, start, or end
- & – match any number of arbitrary characters
- ! – match one answer or another

Modifiers:
- H append to T for no CR/LF (TH:)
- J append to M for auto jump to next M if no match
- S append to M to allow for some spelling errors
- X append to A to override input text editing

Conditioners (append to any op code):
- Y execute only if last match was successful
- N execute only if last match was unsuccessful
- E execute only if error condition raised
- C execute only if last conditional was true
- n(0-9) execute only if equal to answer counter

Rational Expressions (append to any op code, enclose in parentheses):

execute only if logical value of expression is true

Execution Time Commands (entered by student in response to A:):

goto destination – jump to destination

@ text – causes U: SYSX to be executed

Computational Operators and Features:

+ – * / ** !! (string concatenation) & (and) !(or) ← diadics
=< > <> <= >= ← relational
~ (not) – ← monadics

%A – system variable, answer counter (number of sequential uses of same A: instruction)

%B – system variable, answer buffer

ABS, FIX, INT, SGN, RND, SIN, COS, ATN, SQR, EXP
LOG, LN, STR (float to string conversion), FLO (string to float conversion)
ASC, CHR, LEN, INS (instring)

Subscripts:

X(N), Y(N,N)	N = variable or constant, 0 to 999
A$(P)	character at position P
A$(P,L)	substring, starting at P for length L

It is evident from this summary that COMMON PILOT includes numeric and string operators which make it comparable to a good extended BASIC: random access disk files are supported (when handled in the DOS), and the conversational and pattern-matching capabilities are very powerful. We maintain that COMMON PILOT contains all the features necessary for versatile CAI, and that it does not contain unnecessary features. We do anticipate extensions to facilitate graphics since at present graphics can be handled only by loading control characters into string variables and displaying them via Type instructions. A set of graphics commands will emerge from the standardization meetings. They will take the form of a set of operators which can be included in the text field of a G: op code.

PILOT has the unusual history of being a CAI language which was designed and implemented by authors who were experienced CAI authors and who used the language to program CAI materials.* The result in the case of COMMON PILOT is a language which is powerful enough for programming sophisticated CAI materials, which is efficient enough for use on microcomputers, on minicomputers, and on large mainframe computers, and which is simple enough for teachers to master in a short workshop. With our implementations it will be easy for all CAI users to get access to 100% compatible systems that are extended enough to deliver high quality CAI. The SWTP 6800 version provides an inexpensive entry into CAI for new users. The PASCAL version provides an easy way to implement COMMON PILOT on larger machines either by running directly or by coding from PASCAL into machine language. While the exact configuration needed for the 8080 version is not yet defined–and may depend on vendor support –it will provide an alternate low-cost entry into CAI. Potential users desiring a more complete description of COMMON PILOT may obtain an author's guide from Micropi for $6.

Micropi also plans to offer courseware written in COMMON PILOT. We will have a number of courses available by Fall of 1978. It is hoped that the library of courses available will continually grow as the availability of low-cost hardware encourages more CAI authors. We also have the facility to do machine translation from COURSEWRITER II and III and much of BASIC into COMMON PILOT.

So, if you're interested in CAI that's like the big guy's CAI but at hobby prices, look into what COMMON PILOT offers. If you are tempted to invent another version of PILOT, please let us know what you propose. We'll transmit them, but not necessarily advocate them, to the standardization meetings. It just might not be too late to avoid doing to PILOT what has been done to BASIC.

* We have observed that programmers who are experienced with computers but new to COMMON PILOT and CAI often propose extensions to the language which are of very limited use in CAI programming.

Editors Note: People's Computer Company publications have long been interested in implementations and uses of PILOT. Articles, programs and listings can be found in the following: People's Computer Company Newspaper *Vol 5, Nos. 3-5* People's Computers *magazine Vol 5, No. 6, Vol 6, Nos. 1 & 2.*

A complete documentation and source listing of Dr. John Starkweather's 8080 PILOT can be found in Dr. Dobb's Journal *Vol 2, Nos. 4 & 5 (April, May, 1977).* –*TRW.*

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 28 | September 1978 | Volume 3, Issue 8

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

EM8080: An 8080 Emulator for 6800 Systems *Gary Gaugler* 4
Now you can run 8080 code on a 6800

Lisp for the 6800 *Fritz van der Wateren* 24
Description of an implementation of this list processing language

FORTH Dump Programs *John James* 26
Some examples of programming in FORTH

Dumping Northstar Disk Files *Titus Purdin* 38

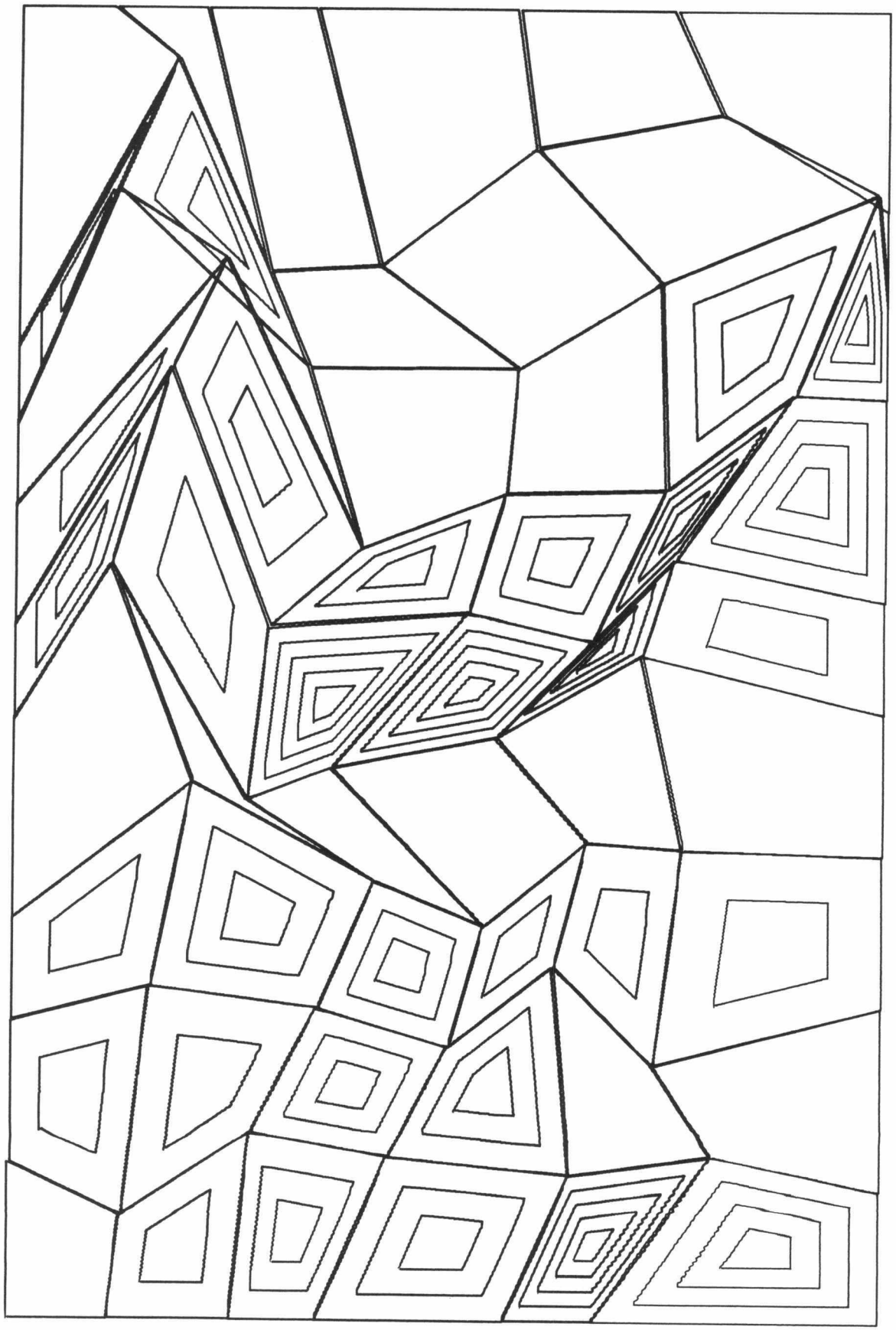
Vocal Memory Dump *Phil Mork* 42
Demonstrating a simple voice output technique

HARDWARE

The Last of the First *Aubrey Dahl* 20
The last Univac II is laid to rest

Making a New Case for the H-9 Terminal *Howard L. Nurse* 30
Converting Heath's terminal to display lower case

A Programmable IC Tester for KIM *Joel Swank* 33



© David Dameron 1979

INFLATION HITS DR. DOBB'S

A magazine depends on the good will of its readers for its continued existence. *Dr. Dobb's Journal* has been especially fortunate in having readers who have put up with irregular deliveries, occasional foot-in-mouth disease and have yet submitted valuable material out of sheer interest in the subject and a desire to share their enthusiasm with others. We count ourselves lucky for knowing such folks.

However, Time and Dow-Jones wait for no man, and the rising cost of everything is forcing us to increase the price of *Dr. Dobb's*. We have gone on for 2½ years without an increase in the cover price and now find it inevitable if we are to survive as an independent journal which accepts no paid advertising. Starting with issue number 29 (October, 1978) the cover price will be $2 per issue and the subscription price will be $15 per year. Until October first, we are including the opportunity to subscribe or renew your present subscription at the old price of $12.

While we regret the necessity of a price increase, we hope that it will enable us to continue to present the kind of articles and comment which will make personal computing an interesting and rewarding activity. For this we can only partially rely on money. The other part of the equation is the continued confidence and support of you, our readers and contributors.

– Tom Williams

SOME PRAISE FOR A SUPPLIER

Dear 'Doc': *Received: 78 June 8*

Just wanted to drop you a line to let you know how much I enjoy your mag. I think it's first class. I also think we ought to thank all of the authors who have so kindly allowed you (and all of us) to share the fruits of their labors.

Sometimes, we hobbyists are quick to condemn a supplier for poor service, but slow to praise good service. I want to do my share to correct this by mentioning a company which has gone out of its way to help me with an equipment problem. The company is George Risk Industries, G.R. I. Plaza, Kimball, Nebraska 69145. They are an OEM vendor of keyboards and related items, and have several low priced models for the hobbyist market. I had a problem with my keyboard, and I wrote them a letter describing the problem. To my surprise, I received a personal reply within 1 week, including mailing time. The info was helpful, but did not solve the problem, so I wrote again. Within about 10 days, I had not only received a reply, but also a bag of parts which fixed the problem, all at NO CHARGE. Considering the tales of woe we all read about, I was most impressed with this service. Along with the 'fix', the company included some other useful info. They are developing applications sheets which show how to interface GRI keyboards to popular microcomputers. Keep up the good work.

Regards.
Lewis Moseley

2514 Glendale Ct. NE
Conyers, GA 30207

... AND MORE PRAISE

Dear Tom: *Received: 78 July 24*

I have read for many months now your comments about the widely varying quality and support offered to the Microcomputer fraternity. Once in a while, something unusually good happens and this is the reason that I want to call it to your attention. The good guys in this case are KEA-Microdesign of Canada, the manufacturers of the Graphic-Add Board which when added to a P.T. Co VDM turns it into a device capable of excellent graphics for the extremely modest sum of $50.00.

I originally purchased Graphic-Add when it first came on the market and because of my limited software experience was at first chagrined to find that the software basically was written for SOL and the SOLOS operating system. I had neither. What I did have was a Micropolis disk system that used packed BCD for storage while Solos wanted pure Binary.

I wrote to Ken Anderson and got an immediate response and a number of helpful suggestions. By the time I was done, Ken Anderson had written me a conversion program using the peculiar Assembler Language calls of Micropolis so that I was able to link Micropolis Basic to Graphic-Add.

I was still glowing with pleasure from the help that I had received, when to my surprise, Ken Anderson sent me a complete updated set of software, *at no extra charge*, with a wonderful cassette which included a super version of LIFE with full graphics, a graphics version of Li-Chen Wang's Robot Language, and a patch called GAX, which when combined with P.T. Co's Basic 5 gives a powerful Basic with full Graphics Control Commands.

I have really never written letters of praise before. But when one gets this kind of support and interest from someone really interested in helping regardless of the miniscule cost of the purchased item, I figured I had to let you know.

By the way, while I am handing out rare praise, may I also compliment Micropolis for their support and rapid response to my many and varied questions about their system.

Sincerely yours,
Raymond M. Glueck

1 Rutlege Road
Pine Brook, N.J. 07058

FLASH WARNING – SAM 76 CP/M RELEASE

News Release *Received: 78 July 31*

Forty undescribed but working functions. Particularly dangerous uncautious execution %xu, etc. /=. Watch *Dr. Dobb's* for descriptions. The source program for the disk functions as implemented in the CP/M version will be published in *Dr. Dobb's Journal*, however copies may be purchased for the implementation of the moment – no guarantee that it will match the object code furnished – but to serve as a model for implementation in other disk systems; price is $10.00 postpaid; available as a xerox printed copy or in machine readable form on standard diskette or paper tape [TDL macro format]. SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534.

EM8080: An 8080 Emulator for 6800 Systems

BY GARY GAUGLER
2276 Beaver Valley Road
Fairborn, OH 45234

If you enjoy your 6800 as much as I do, you rarely spend much time with thoughts of the 8080 or Z80. Nevertheless, there always seems to be some program which appears in a magazine that I would like to try out but can't since it was written in 8080 code and won't work on my 6800. Well now we can run 8080 code right on our 6800 machine via "emulation."

Emulation is a technique whereby the code of the "target" (the 8080) is executed on a "host" machine (our 6800). The emulation process executes the target machine code via software written in the host's language. So what we have is a way to run 8080 machine language programs via software in a 6800 system; we do not have to rush out and buy an 8080. All of this emulation capability is not free; the cost is loss of speed in emulated code. Code which is run on an emulator (at least for macroprocessors like we use) will be slower, much slower, than it would be if run on a machine for which it was intended. Since fast execution was not a prime consideration for emulating, just the program's intended function, the slow execution time can be tolerated.

The emulator maintains a set of pseudo registers for the 8080. These are actually locations in the 6800 machine's RAM such that when a reference is made to a specific 8080 register via an 8080 machine instruction, the RAM location corresponding to that register is operated on to result in a RAM contents equal to that in the real 8080 register. One of the most important registers to emulate is the condition code register; great care must be taken to ensure that this emulated register contains valid results for *all* operations which might be performed.

The emulator presented here provides a bonus feature: it can execute 8080 code and in addition, it can do so in a single step mode. This is nice if you want to try out a routine you have seen or one which you have written in 8080 code and would like to observe the program flow step-by-step. This feature is possible because EM8080 contains two levels of emulation: a monitor level in which the user interacts with the emulator and provides instructions to the emulator, and another level which is the actual emulation function.

EM8080 displays the emulated registers when requested through the emulator monitor (the RG command), whenever the emulator stops emulating because it encounters an opcode it does not emulate, or when it encounters an address which was specified as being a break address. There are certain 8080 instructions which are not emulated—those which deal with the parity bit. When an instruction is encountered but is not emulated by EM8080, emulation ceases and a message "NOT EMULATED CODE" is displayed along with a register dump.

The instructions which EM8080 recognizes as monitor commands are: HP, AB, CB, LO, CR, RU, RG, SO, SF, PC, EO, EF, EX, control C, and space bar. When the emulator is first executed, a colon (:) is displayed which indicates that no emulation is in progress; the emulator is in a monitor mode. The HP command will print out a summary of the monitor commands and explanations in case you should need a little refreshing while emulating. The other commands are:

AB – This command will cause EM8080 to prompt for an address which will be the breakpoint address. When a program is being emulated and the address set up by the AB command is encountered, emulation ceases and a register dump is performed followed by a return to the emulator monitor.

CB – This command removes the break address as set by the AB command.

LO – This command permits the entry of octal data into memory.

CR – This command clears the emulated registers.

RG – This causes the emulated registers to be displayed.

SO – This command enables single stepping through the 8080 program.

SF – Turns off the single step mode.

PC – This command sets the 8080 program counter (PC) to a value entered from the keyboard.

EO – This command enables the echoing of characters when received.

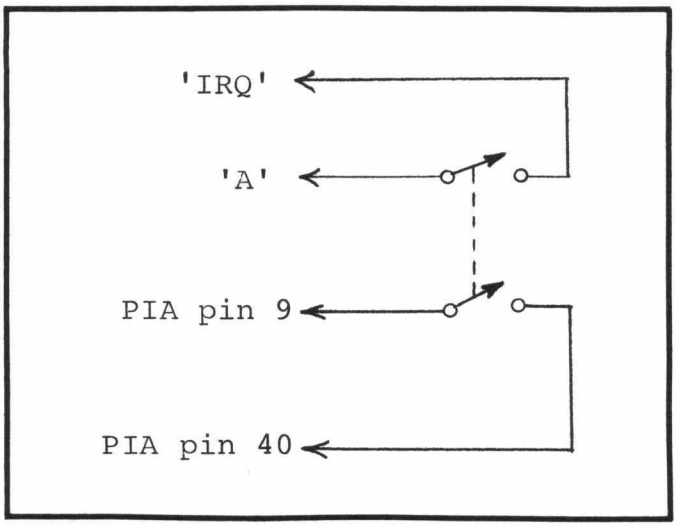
EF – Turns off the echoing of characters (normal mode).

EX – Also control C. Returns to MIKBUG monitor.

RU – Also space bar. Causes the 8080 program to be RUn. The space bar accomplishes the same function but is ideally suited to use in the single step mode since execution of each instruction is initiated via a single space bar depression.

I/O

The only specific care which must be taken before using the emulator is in emulating the input character function from the keyboard. EM8080 may be used with MIKBUG at 300 baud maximum or with an ACIA type monitor at up



Install a DPDT or DPST switch as shown. When emulating, throw the switch closed. When operating the computer normally, be sure that the switch is open.

to 9600 baud. The input character function is handled via an IRQ interrupt initiated by the start bit when using MIKGUB or via the receive data register full flag (RDRF) in an ACIA. The program listing is written for the MIKBUG monitor with the necessary modifications noted for use with an ACIA. Both versions have been tested with the benchmark program and perform the same.

Fig. 1 shows the hardware changes which must be made to the MIKBUG interface (the MP-C card with the SWTPC 68 computer). When using an ACIA, connect the IRQ (not the NMI) pad to the "I" pad on the MP-S serial card.

The emulator was benchmarked on the PALO ALTO Tiny Basic by Li-Chen Wang which appeared in *Dr. Dobb's Journal*, Vol. 1, No. 4, May, 1976 on page 12. This BASIC runs about twenty times slower than the same BASIC on an 8080 system. The emulator was run on an SWTPC using a /1 CPU board.

Due to the length of the emulator (about 3K), arrangements have been made with Computerware in Encinitas, CA to supply KC cassettes of the emulator and the benchmark BASIC for $9.95 post paid. Computerware's address is: 830 First St., Encinitas, CA 92024.

```
                    NAM   EM8080
                *  VERSION 2.5
                *
                *  6800 BASED 8080 EMULATOR
                *
                *  SLOWER THAN THE AVERAGE 8080.
                *
                *  (C) COPYRIGHT 1978 BY G. GAUGLER. ALL RIGHTS RESERVED.
                *  BASED ON ORIGINAL PGM BY V. BARCOCK
                *
                *  PERMISSION TO REPRODUCE FOR NON-COMMERCIAL PURPOSES
                *  IS GRANTED.
                *
                      OPT   NOG,NOP
E0D0            MIKBUG EQU  $E0D0     MONITOR ENTRY POINT
E07E            PDATA1 EQU  $E07E     PRINT STRING
E1D1            OUTCH  EQU  $E1D1     OUTPUT CHAR IN A-REG
E1AC            INCH   EQU  $E1AC     INPUT CHAR IN A-REG
E047            BADDR  EQU  $E047     BUILD 16 BIT ARG IN X-REG
E0CA            OUT2HS EQU  $E0CA     OUTPUT 2 HEX CHARS & SPACE
8004            PIA    EQU  $8004     MIKBUG PIA DATA PORT
E0C8            OUT4HS EQU  $E0C8     OUTPUT 4 HEX CHARS & SPACE
8005            PIAS   EQU  $8005     MIKBUG CONTROL PORT
A000            IO     EQU  $A000     IRQ VECTOR LOCATION
                      ORG   $5000
5000 20 0F      START BRA   BEGIN
5002 CE 5011    NMIV  LDX   #BEGIN
5005 FF A048          STX   $A048
5008 7E E0D0          JMP   MIKBUG    GO TO MIKBUG
500B CE 5A9A    CRLF  LDX   #CRLFS    DO A CR - LF
500E 7E E07E          JMP   PDATA1
5011            BEGIN EQU   *
5011 A6 05            LDA A #$05      CAUSE SERIAL PIA TO
5013 B7 8005          STA A PIAS      INTERRUPT ON NEGATIVE SHIFT
                * FOR ACIA USE: LDAA #$91
                *               STAA  $8008
                *
5016 01               NOP
5017 01               NOP
5018 01               NOP
5019 01               NOP
501A 01               NOP
501B CE 57AE          LDX   #GETTTY
501E FF A000    MONTOR STX  IO          TURN INTERRUPT OFF
5021 0F               SEI
5022 8D E7            BSR   CRLF
5024 86 3A            LDA A #':       PRINT COLON FOR EMULATOR LEVEL
5026 8D E1D1          JSR   OUTCH
5029 8D E1AC          JSR   INCH        GET 1ST CHAR OF MON INSTRUCTION
502C C6 20            LDA B #$20        SPACE KEY? YES. GO TO RUN
502E 11               CBA
502F 27 21            BEQ   MON5
5031 B7 5AD7          STA A MONCD
5034 C6 03            LDA B #$03      CONTROL C?
5036 11               CBA
5037 27 1E            BEQ   MON6      YES. BRANCH TO MIKBUG
5039 8D E1AC          JSR   INCH        GET 2ND CHAR OF MON INSTRUCTION
503C C6 0D            LDA B #$0D        WAS IT A CARRIAGE RETURN?
503E 11               CBA
503F 26 0A            BNE   MON4
5041 86 20            LDA A #'        YES. PUT A SPACE IN
5043 CE 5A9B          LDX   #LF
5046 8D E07E          JSR   PDATA1
5049 B7 5AD8    MON4  STA A MONCD+1
504C 8D 0C            BSR   MONTST
504E 8D BB            BSR   CRLF
```

```
 73 5050 20 CF          BRA     MONTOR      GO BACK TO MONITOR
 74 5052 BD 50D8 MON5   JSR     RUN
 75 5055 20 CA          BRA     MONTOR
 76 5057 7E 5002 MON6   JMP     NMIV        RETURN TO MIKBUG MONITOR

 78 505A CE 5080 MONTST LDX     #MONTBL     POINT TO MONITOR INSTRUCTION TABLE
 79 505D B6 5AD7 MN3    LDA A   MONCD      COMPARE FIRST POSITION
 80 5060 A1 00          CMP A   0,X         FIRST POSITION EQUAL?
 81 5062 26 0B          BNE     MN1         NO,
 82 5064 B6 5AD8        LDA A   MONCD+1     YES, 2ND POSITION EQUAL?
 83 5067 A1 01          CMP A   1,X
 84 5069 26 04          BNE     MN1         NO
 85 506B EE 02          LDX     2,X         YES, GET GO TO ADDRESS
 86 506D 6E 00          JMP     0,X         GO TO ROUTINE

 88 506F 86 5A  MN1     LDA A   #'Z         TEST FOR END OF TABLE
 89 5071 A1 00          CMP A   0,X
 90 5073 27 06          BEQ     MN2         YES, INVALID MONITOR INSTRUCTION
 91 5075 08             INX                 GO TO NEXT TABLE ENTRY
 92 5076 08             INX
 93 5077 08             INX
 94 5078 08             INX
 95 5079 20 E2          BRA     MN3         GO LOOK AT ANOTHER TABLE ENTRY
 96 507B 86 3F  MN2     LDA A   #'?         PRINT QUESTION MARK FOR ERROR
 97 507D 7E F1D1        JMP     OUTCH       ..& RETURN TO MONITOR

 99 50A0        MONTBL  EQU     *
100 50A0 48             FCC     /HP/        HELP
101 50A2 5A 9D          FDB     HELPRT
102 50A4 41             FCC     /AB/        SET ADDRESS BREAK STOP ADDRESS
103 50A6 50 C2          FDB     ADRSET
104 50A8 43             FCC     /CB/        CLEAR ADDRESS BREAK STOP CONDITION
105 50AA 50 BE          FDB     CLRAB
106 50AC 4C             FCC     /LO/        LOAD OCTAL
107 50AE 58 A2          FDB     LDOCT
108 50B0 43             FCC     /CR/        CLEAR REGS
109 50B2 51 1C          FDB     CLREGS
110 50B4 52             FCC     /RU/        RUN
111 50B6 50 D8          FDB     RUN
112 50B8 52             FCC     /RG/        DISPLAY REGISTERS
113 50BA 50 B8          FDB     REG
114 50BC 53             FCC     /SO/        TURN SINGLE CYCLE ON
115 50BE 51 0C          FDB     SCYC
116 50C0 50             FCC     /PC/        SET PROGRAM COUNTER
117 50C2 51 73          FDB     SETPC
118 50C4 53             FCC     /SF/        TURN SINGLE CYCLE OFF
119 50C6 51 10          FDB     SCYF
120 50C8 45             FCC     /EO/        TURN ECHO ON
121 50CA 51 14          FDB     ECHN
122 50CC 45             FCC     /EF/        TURN ECHO OFF
123 50CE 51 18          FDB     ECHF
124 50D0 45             FCC     /EX/        EXIT TO MONITOR
125 50D2 E0 D0          FDB     MIKBUG
126 50D4 5A             FCC     /ZZ/
127 50D6 A0 4A          FDB     EXTRNL
128 50D8        REG     EQU     *
129 50D8 BD 500B        JSR     CRLF
130 50DB 7E 5140        JMP     REGD        DISPLAY REGISTERS

132 50DE 7F 5AD9 CLRAB  CLR     ABFLG       CLEAR ADDRESS BREAK CONDITION
133 50E1 39             RTS

135 50E2        ADRSET  EQU     *
136 50E2 BD 500B        JSR     CRLF
137 50E5 CE 5A73        LDX     #ABMSG
138 50E8 BD E07E        JSR     PDATA1
139 50EB BD E047        JSR     BADDR
140 50EE FF 5ADA        STX     ABADDR
141 50F1 BD 500B        JSR     CRLF
142 50F4 7C 5AD9        INC     ABFLG
143 50F7 39             RTS

145 50F8        RUN     EQU     *           RUN 8080 PROGRAM
146 50F8        RUNC    EQU     *
147 50F8 0E             CLI                 ENABLE INTERRUPTS
148 50F9 7F 5ADC        CLR     HTFLG       CLEAR HALT FLAG IF ON
149 50FC 7D 5AD9        TST     ABFLG       TEST FOR ADDRESS BREAK
150 50FF 27 0A          BEQ     SKP1
151 5101 FE 5ABF        LDX     EPC         TEST FOR ADDRESS BREAK
152 5104 BC 5ADA        CPX     ABADDR
153 5107 26 02          BNE     SKP1
154 5109 20 0F          BRA     RUNA        GO CAUSE A HALT

156 510B        SKP1    EQU     *
157 510B BD 5182        JSR     IFETCH      EXECUTE ONE 8080 INSTRUCTION
158 510E 7D 5ADC        TST     HTFLG       TEST FOR A HALT
159 5111 26 07          BNE     RUNA        BR YES
160 5113 7D 5ADD        TST     SCYFLG      TEST FOR SINGLE CYCLE
161 5116 26 11          BNE     RUNB
162 5118 20 DE          BRA     RUNC
163 511A CE 5A68 RUNA   LDX     #HTMSG      PRINT HALT MESSAGE
164 511D BD E07E        JSR     PDATA1
165 5120 CE 5ABF        LDX     #EPC        PRINT HALT ADDRESS
166 5123 BD E0C8        JSR     OUT4HS
167 5126 BD 500B        JSR     CRLF
168 5129 7E 50B8 RUNB   JMP     REG         SINGLE CYCLE & HALT EXIT

171 512C 7C 5ADD SCYC   INC     SCYFLG      TURN SINGLE CYCLE ON
172 512F 39             RTS

174 5130 7F 5ADD SCYF   CLR     SCYFLG      TURN SINGLE CYCLE OFF
175 5133 39             RTS

177 5134 7C 5AE5 ECHN   INC     ECHOFG      TURN TTY ECHO ON
178 5137 39             RTS

180 5138 7F 5AE5 ECHF   CLR     ECHOFG      TURN TTY ECHO OFF
181 513B 39             RTS

183 513C        CLREGS  EQU     *
184 513C 7F 5ABF        CLR     EPC
185 513F 7F 5AC0        CLR     EPC+1
186 5142 5F             CLR B
187 5143 7F 5AB0        CLR     C
188 5146 7F 5AB1        CLR     D
189 5149 7F 5AB2        CLR     E
190 514C 7F 5AB3        CLR     H
191 514F 7F 5AB4        CLR     L
192 5152 7F 5AB6        CLR     ECC
193 5155 7F 5AB5        CLR     ACC
194 5158 CE 5AD6        LDX     #STACK
195 515B 08             INX
196 515C FF 5AB7        STX     EMSTK
197 515F 39             RTS

199 5160        REGD    EQU     *           DISPLAY REGISTERS
200 5160 BD 500B        JSR     CRLF
201 5163 CE 5AAF        LDX     #REGS
202 5166 BD E0CA        JSR     OUT2HS      B REG
203 5169 BD E0CA        JSR     OUT2HS      C REG
204 516C BD E0CA        JSR     OUT2HS      D REG
205 516F BD E0CA        JSR     OUT2HS      E REG
206 5172 BD E0CA        JSR     OUT2HS      H REG
207 5175 BD E0CA        JSR     OUT2HS      L REG
208 5178 BD E0CA        JSR     OUT2HS      ACC
209 517B BD E0CA        JSR     OUT2HS      CC
210 517E BD 500B        JSR     CRLF
211 5181 CE 5AB7        LDX     #EMSTK
212 5184 BD E0C8        JSR     OUT4HS      EMULATOR STACK POINTER
213 5187 CE 5ADE        LDX     #OPCD
214 518A BD E0CA        JSR     OUT2HS      OP CODE
215 518D CE 5ABF        LDX     #EPC
216 5190 7E E0C8        JMP     OUT4HS      PC
```

```
218 5173          SETPC  EQU    *          SET 8080 PROGRAM COUNTER
219 5173 CE 5A84         LDX    #PCMSG
220 5176 BD E07E         JSR    PDATA1
221 5179 BD E047         JSR    BADDR
222 517C FF 5ABF         STX    EPC
223 517F 7E 500B         JMP    CRLF

225              *  EXECUTE ON INSTRUCTION AND RETURN
226              *  TO RUN ROUTINE TO DETERMINE IF ANOTHER
227              *  INSTRUCTION IS TO BE EXECUTED
228 5182 FE 5ABF IFETCH LDX    EPC        GET PROGRAM COUNTER
229 5185 A6 00           LDA A  0,X        GET 1ST BYTE OF INSTRUCTION
230 5187 B7 5ADE         STA A  OPCD
231 518A 16              TAB              SAVE CODE IN B-REG
232 518B 08              INX              INCREMENT PROGRAM COUNTER
233 518C FF 5ABF         STX    EPC
234 518F FF 5ABD         STX    NEWPC
235 5192 17              TBA              GET FRESH INSTRUCTION BYTE
236 5193 84 38           AND A  #$38       MASK FOR DESTINATION
237 5195 44              LSR A            RIGHT JUSTIFY DESTINATION
238 5196 44              LSR A
239 5197 44              LSR A
240 5198 B7 5ABA         STA A  DDD        SAVE DESTINATION
241 519B 17              TBA              GET SOURCE FIELD
242 519C 84 07           AND A  #$07       MASK FOR SOURCE
243 519E B7 5AB9         STA A  SSS        SAVE SOURCE
244 51A1 17              TBA
245 51A2 84 C0           AND A  #$C0       TEST D7,D6
246 51A4 81 C0           CMP A  #$C0       = 11?
247 51A6 27 11           BEQ    CD11       YES
248 51A8 81 40           CMP A  #$40       = 01?
249 51AA 27 07           BEQ    CD01       YES
250 51AC 81 80           CMP A  #$80       = 10?
251 51AE 27 06           BEQ    CD10       YES
252 51B0 7E 54DA         JMP    RT00

254 51B3          CD01   EQU    *
255 51B3 7E 51BC         JMP    RT01

257 51B6          CD10   EQU    *
258 51B6 7E 5724         JMP    RT10

260 51B9          CD11   EQU    *
261 51B9 7E 521D         JMP    RT11

263 51BC          RT01   EQU    *          MOVE TYPE: MOV R1,R2
264 51BC CE 5AAF         LDX    #REGS
265 51BF 86 06           LDA A  #$6
266 51C1 B1 5AB9         CMP A  SSS        MEMORY TYPE MOVE OR HALT?
267 51C4 27 22           BEQ    RM         YES = REG TO MEM OR HALT
268 51C6 B1 5ABA         CMP A  DDD        MEMORY TYPE MOVE?
269 51C9 27 3D           BEQ    MR         YES = MEM TO REG
270 51CB B6 5AB9         LDA A  SSS        GET SOURCE REG
271 51CE 81 07           CMP A  #7
272 51D0 26 01           BNE    *+3
273 51D2 4A              DEC A
274 51D3 B7 51D7         STA A  RT01A+1    BUILD INSTRUCTION
275 51D6 E6 00   RT01A   LDA B  0,X        GET SOURCE DATA
276 51D8 B6 5ABA         LDA A  DDD        GET SOURCE REG
277 51DB 81 07           CMP A  #7
278 51DD 26 01           BNE    *+3
279 51DF 4A              DEC A
280 51E0 B7 51E4         STA A  RT01B+1    BUILD INSTRUCTION
281 51E3 E7 00   RT01B   STA B  0,X        STORE DATA IN DESTINATION
282 51E5 7E 5378         JMP    END

284 51E8          RM     EQU    *          MEM TO REG OR HALT
285 51E8 B1 5ABA         CMP A  DDD        HALT?
286 51EB 27 18           BEQ    JHALT      YES
287 51ED FE 5AB3         LDX    H          GET MEMORY ADDRESS IN H-L
288 51F0 E6 00           LDA B  0,X        GET MEMORY DATA
289 51F2 CE 5AAF         LDX    #REGS      POINT TO REGISTERS
290 51F5 B6 5ABA         LDA A  DDD        GET REG NUMBER

291 51F8 81 07           CMP A  #7
292 51FA 26 01           BNE    *+3
293 51FC 4A              DEC A
294 51FD B7 5201         STA A  RM1+1
295 5200 E7 00   RM1     STA B  0,X        PUT DATA IN REGISTER
296 5202 7E 5378         JMP    END

298 5205 7E 52D7 JHALT   JMP    HALT       HALT INSTRUCTION

300 5208         MR      EQU    *          REG TO MEM
301 5208 B6 5AB9         LDA A  SSS        GET REG NUMBER
302 520B 81 07           CMP A  #7
303 520D 26 01           BNE    *+3
304 520F 4A              DEC A
305 5210 B7 5214         STA A  MR1+1
306 5213 A6 00   MR1     LDA A  0,X        GET REG DATA
307 5215 FE 5AB3         LDX    H          GET MEM ADDRESS IN H-L
308 5218 A7 00           STA A  0,X        STORE REG IN MEM
309 521A 7E 5378         JMP    END

311 521D         RT11    EQU    *
312 521D 17              TBA              GET FRESH INSTRUCTION BYTE
313 521E 84 3F           AND A  #$3F       USE SSS AND DDD ONLY
314 5220 48              ASL A            TIMES 2
315 5221 B7 5228         STA A  RT11A+1    BUILD INSTRUCTION
316 5224 CE 522F         LDX    #T3XX      POINT TO BRANCH TABLE
317 5227 EE 00   RT11A   LDX    0,X
318 5229 B6 5AB6         LDA A  ECC        GET 8080 CONDITION CODE
319 522C 06              TAP
320 522D 6E 00           JMP    0,X        EXECUTE 8080 INSTRUCTION OF 3XX TYPE

322 522F         T3XX    EQU    *
323 522F 52 AF           FDB    C300
324 5231 52 FA           FDB    POP
325 5233 54 25           FDB    C302
326 5235 53 39           FDB    JUMP       C303
327 5237 54 06           FDB    C304
328 5239 52 DD           FDB    PUSH
329 523B 54 43           FDB    C306
330 523D 53 42           FDB    RST
331 523F 52 B3           FDB    C310
332 5241 53 60           FDB    RET
333 5243 54 21           FDB    C312
334 5245 52 CB           FDB    NEM
335 5247 54 02           FDB    C314
336 5249 53 17           FDB    CALL
337 524B 54 54           FDB    C316
338 524D 53 42           FDB    RST
339 524F 52 B7           FDB    C320
340 5251 52 FA           FDB    POP
341 5253 54 1D           FDB    C322
342 5255 57 E8           FDB    OUT
343 5257 53 FE           FDB    C324
344 5259 52 DD           FDB    PUSH
345 525B 54 5A           FDB    C326
346 525D 53 42           FDB    RST
347 525F 52 BB           FDB    C330
348 5261 52 CB           FDB    NEM
349 5263 54 19           FDB    C332
350 5265 57 B8           FDB    IN
351 5267 53 FA           FDB    C334
352 5269 52 CB           FDB    NEM
353 526B 54 60           FDB    C336
354 526D 53 42           FDB    RST
355 526F 52 CB           FDB    NEM
356 5271 52 FA           FDB    POP
357 5273 52 CB           FDB    NEM
358 5275 53 B2           FDB    C343
359 5277 52 CB           FDB    NEM
360 5279 52 DD           FDB    PUSH
361 527B 54 66           FDB    C346
362 527D 53 42           FDB    RST
363 527F 52 CB           FDB    NEM
```

```
 364 5281 53 F1           FDB    C351
 365 5283 52 CB           FDB    NEM
 366 5285 53 D2           FDB    C353
 367 5287 52 CB           FDB    NEM
 368 5289 52 CB           FDB    NEM
 369 528B 54 6C           FDB    C356
 370 528D 53 42           FDB    RST
 371 528F 52 BF           FDB    C360
 372 5291 52 FA           FDB    POP
 373 5293 54 29           FDB    C362
 374 5295 54 8A           FDB    C363
 375 5297 54 0A           FDB    C364
 376 5299 52 DD           FDB    PUSH
 377 529B 54 75           FDB    C366
 378 529D 53 42           FDB    RST
 379 529F 52 C3           FDB    C370
 380 52A1 53 E7           FDB    C371
 381 52A3 54 2D           FDB    C372
 382 52A5 54 84           FDB    C373
 383 52A7 54 0E           FDB    C374
 384 52A9 52 CB           FDB    NEM
 385 52AB 54 7E           FDB    C376
 386 52AD 53 42           FDB    RST

 388                      *
 389                      *  RETURN ON CONDITION CODE
 390                      *
 391 52AF 26 17   C300    BNE    JRET        RETURN ON NO ZERO
 392 52B1 20 12           BRA    JENDA
 393 52B3 27 13   C310    BEQ    JRET        RETURN ON ZERO
 394 52B5 20 0E           BRA    JENDA
 395 52B7 24 0F   C320    BCC    JRET        RETURN ON NO CARRY
 396 52B9 20 0A           BRA    JENDA
 397 52BB 25 0B   C330    BCS    JRET        RETURN ON CARRY
 398 52BD 20 06           BRA    JENDA
 399 52BF 2A 07   C360    BPL    JRET        RETURN ON POSITIVE
 400 52C1 20 02           BRA    JENDA
 401 52C3 2B 03   C370    BMI    JRET        RETURN ON MINUS
 402 52C5 7E 5378 JENDA   JMP    END
 403 52C8 7E 5360 JRET    JMP    RET

 405 52CB         NEM     EQU    *           NOT EMULATED
 406 52CB BD 500B         JSR    CRLF
 407 52CE CE 5A88         LDX    #NEMMSG
 408 52D1 BD E07E         JSR    PDATA1
 409 52D4 BD 500B         JSR    CRLF
 410 52D7 7C 5ADC HALT    INC    HTFLG       SET HALT FLAG
 411 52DA 7E 5378         JMP    END

 413 52DD         PUSH    EQU    *
 414 52DD BF 5AC1         STS    MSTK
 415 52E0 BE 5AB7         LDS    EMSTK
 416 52E3 34              DES
 417 52E4 F6 5ABA         LDA B  DDD
 418 52E7 CE 5AAF         LDX    #REGS
 419 52EA F7 52F2         STA B  PUSHA+1
 420 52ED 5C              INC B
 421 52EE F7 52F5         STA B  PUSHB+1
 422 52F1 A6 00   PUSHA   LDA A  0,X         H
 423 52F3 36              PSH A
 424 52F4 A6 00   PUSHB   LDA A  0,X         L
 425 52F6 36              PSH A
 426 52F7 7E 5371         JMP    END1

 428 52FA         POP     EQU    *
 429 52FA BF 5AC1         STS    MSTK
 430 52FD BE 5AB7         LDS    EMSTK
 431 5300 34              DES
 432 5301 F6 5ABA         LDA B  DDD
 433 5304 CE 5AAF         LDX    #REGS
 434 5307 F7 5313         STA B  PULLB+1
 435 530A 5C              INC B
 436 530B F7 5310         STA B  PULLA+1

 437 530E 32              PUL A
 438 530F A7 00   PULLA   STA A  0,X         L
 439 5311 32              PUL A
 440 5312 A7 00   PULLB   STA A  0,X         H
 441 5314 7E 5371         JMP    END1

 443 5317         CALL    EQU    *
 444 5317 BD 53A1         JSR    PCP2        PC+2
 445 531A BF 5AC1         STS    MSTK
 446 531D BE 5AB7         LDS    EMSTK
 447 5320 34              DES
 448 5321 B6 5ABD         LDA A  NEWPC
 449 5324 36              PSH A
 450 5325 B6 5ABE         LDA A  NEWPC+1
 451 5328 36              PSH A
 452 5329 FE 5ABF JUMPX   LDX    EPC
 453 532C A6 01           LDA A  1,X         GET HIGH PART OF NEW PC
 454 532E B7 5ABD         STA A  NEWPC
 455 5331 A6 00           LDA A  0,X         GET LOW PART OF NEW PC
 456 5333 B7 5ABE         STA A  NEWPC+1
 457 5336 7E 5371         JMP    END1

 459 5339         JUMP    EQU    *
 460 5339 BF 5AC1         STS    MSTK
 461 533C BE 5AB7         LDS    EMSTK
 462 533F 34              DES                MAKE 6800 STACK = 8080
 463 5340 20 E7           BRA    JUMPX

 465 5342         RST     EQU    *           RESTART TYPE INSTRUCTION
 466 5342 BF 5AC1         STS    MSTK
 467 5345 BE 5AB7         LDS    EMSTK
 468 5348 34              DES                MAKE 6800 STACK = 8080
 469 5349 B6 5ABF         LDA A  EPC
 470 534C 36              PSH A
 471 534D B6 5AC0         LDA A  EPC+1
 472 5350 36              PSH A
 473 5351 B6 5ABA         LDA A  DDD
 474 5354 48              ASL A
 475 5355 48              ASL A
 476 5356 48              ASL A
 477 5357 7F 5ABD         CLR    NEWPC
 478 535A B7 5ABE         STA A  NEWPC+1
 479 535D 7E 5371         JMP    END1

 481 5360         RET     EQU    *           RETURN TYPE INSTRUCION
 482 5360 BF 5AC1         STS    MSTK
 483 5363 BE 5AB7         LDS    EMSTK
 484 5366 34              DES                MAKE 6800 STACK = 8080
 485 5367 32              PUL A
 486 5368 B7 5ABE         STA A  NEWPC+1
 487 536B 32              PUL A
 488 536C B7 5ABD         STA A  NEWPC
 489 536F 20 00           BRA    END1

 491 5371         END1    EQU    *           FINISH UP EMULATED INSTRUCTION
 492 5371 31              INS                MAKE 8080 STACK = 6800
 493 5372 BF 5AB7         STS    EMSTK
 494 5375 BE 5AC1         LDS    MSTK
 495 5378 FE 5ABD END     LDX    NEWPC
 496 537B FF 5ABF         STX    EPC
 497 537E 8D 03           BSR    CRYADJ      SPECIAL ADJUSTMENT OF CARRY & HALF CARR
 498 5380 7E 5809         JMP    INTST       INTERRUPT TEST

 500 5383         CRYADJ  EQU    *           ADJUSTMENT OF CARRY & HALF CARRY
 501 5383 B6 5AB6         LDA A  ECC
 502 5386 7D 5AE0         TST    CYAC
 503 5389 26 06           BNE    CRY1
 504 538B 7D 5ADF         TST    CY
 505 538E 26 0D           BNE    CRY2
 506 5390 39              RTS

 508 5391 84 DE   CRY1    AND A  #$DE
 509 5393 B7 5AB6 CRY3    STA A  ECC
```

```
510 5396 7F 5ADF         CLR    CY
511 5399 7F 5AE0         CLR    CYAC
512 539C 39              RTS

514 539D 84 FE   CRY2    AND A  #$FE
515 539F 20 F2           BRA    CRY3

517              *  INCREMENT 8080 PROGRAM
518              *  COUNTER +2
519 53A1 FE 5ABF PCP2    LDX    EPC
520 53A4 08              INX
521 53A5 08              INX
522 53A6 FF 5ABD         STX    NEWPC
523 53A9 39              RTS

525 53AA FE 5ABF PCP1    LDX    EPC
526 53AD 08              INX
527 53AE FF 5ABD         STX    NEWPC
528 53B1 39              RTS

530 53B2         C343    EQU    *          XTHL-EXCHANGE TOP OF STACK, H&L
531 53B2 BF 5AC1         STS    MSTK
532 53B5 BE 5AB7         LDS    EMSTK
533 53B8 34              DES                MAKE 6800 STACK = 8080
534 53B9 32              PUL A
535 53BA B7 5ABC         STA A  TMP+1
536 53BD 32              PUL A
537 53BE B7 5ABB         STA A  TMP
538 53C1 B6 5AB3         LDA A  H
539 53C4 36              PSH A
540 53C5 B6 5AB4         LDA A  L
541 53C8 36              PSH A
542 53C9 FE 5ABB         LDX    TMP
543 53CC FF 5AB3         STX    H
544 53CF 7E 5371         JMP    END1

546 53D2         C353    EQU    *          XCHG-EXCHANGE D&L, H&L
547 53D2 FE 5AB3         LDX    H
548 53D5 FF 5ABB         STX    TMP
549 53D8 FE 5AB1         LDX    D
550 53DB FF 5AB3         STX    H
551 53DE FE 5ABB         LDX    TMP
552 53E1 FF 5AB1         STX    D
553 53E4 7E 5378         JMP    END

555 53E7         C371    EQU    *          SPHL-H&L TO STACK POINTER
556 53E7 FE 5AB3         LDX    H
557 53EA 08              INX
558 53EB FF 5AB7         STX    EMSTK
559 53EE 7E 5378         JMP    END

561 53F1         C351    EQU    *          PCHL-H&L TO PROGRAM COUNTER
562 53F1 FE 5AB3         LDX    H
563 53F4 FF 5ABD         STX    NEWPC
564 53F7 7E 5378         JMP    END

566              *
567              *  CALL ON CONDITION TYPE INSTRUCTIONS
568              *
569 53FA 25 1A   C334    BCS    JCALL      CC-CALL ON CARRY
570 53FC 20 12           BRA    JENDB
571 53FE 24 16   C324    BCC    JCALL      CNZ-CALL ON NO CARRY
572 5400 20 0E           BRA    JENDB
573 5402 27 12   C314    BEQ    JCALL      CZ-CALL ON ZERO
574 5404 20 0A           BRA    JENDB
575 5406 26 0E   C304    BNE    JCALL      CNZ-CALL ON NO ZERO
576 5408 20 06           BRA    JENDB
577 540A 2A 0A   C364    BPL    JCALL      CP-CALL ON POSITIVE
578 540C 20 02           BRA    JENDB
579 540E 2B 06   C374    BMI    JCALL      CM-CALL ON MINUS
580 5410         JENDB   EQU    *
581 5410 BD 53A1         JSR    PCP2
582 5413 7E 5378         JMP    END
583 5416 7E 5317 JCALL   JMP    CALL

585              *
586              *  JUMP ON CONDITION TYPE INSTRUCTIONS
587              *
588 5419 25 1A   C332    BCS    JUMPA      JC-JUMP ON CARRY
589 541B 20 12           BRA    JENDC
590 541D 24 16   C322    BCC    JUMPA      JNC-JUMP ON NO CARRY
591 541F 20 0E           BRA    JENDC
592 5421 27 12   C312    BEQ    JUMPA      JZ-JUMP ON ZERO
593 5423 20 0A           BRA    JENDC
594 5425 26 0E   C302    BNE    JUMPA      JNZ-JUMP ON NO ZERO
595 5427 20 06           BRA    JENDC
596 5429 2A 0A   C362    BPL    JUMPA      JP-JUMP ON POSITIVE
597 542B 20 02           BRA    JENDC
598 542D 2B 06   C372    BMI    JUMPA      JM-JUMP ON MINUS

600 542F         JENDC   EQU    *
601 542F BD 53A1         JSR    PCP2
602 5432 7E 5378         JMP    END
603 5435 7E 5339 JUMPA   JMP    JUMP

605 5438 FE 5ABF I3X6    LDX    EPC
606 543B F6 5AB5         LDA B  ACC
607 543E B6 5AB6         LDA A  ECC
608 5441 06              TAP
609 5442 39              RTS

611 5443 8D F3   C306    BSR    I3X6
612 5445 EB 00           ADD B  0,X         ADI - ADD IMMEDIATE TO ACC
613 5447         JENDD   EQU    *
614 5447 07              TPA
615 5448 B7 5AB6         STA A  ECC
616 544B F7 5AB5         STA B  ACC
617 544E BD 53AA         JSR    PCP1
618 5451 7E 5378         JMP    END

620 5454 8D E2   C316    BSR    I3X6        ACI-ADD IMMEDIATE TO A WITH CARRY
621 5456 E9 00           ADC B  0,X
622 5458 20 ED           BRA    JENDD
623 545A 8D DC   C326    BSR    I3X6        SUI-SUBTRACT IMMEDIATE FROM A
624 545C E0 00           SUB B  0,X
625 545E 20 E7           BRA    JENDD
626 5460 8D D6   C336    BSR    I3X6        SUBI-SUB IMM FROM A WITH BORROW
627 5462 E2 00           SBC B  0,X
628 5464 20 E1           BRA    JENDD
629 5466 8D D0   C346    BSR    I3X6        ANI-AND IMM WITH A
630 5468 E4 00           AND B  0,X
631 546A 20 DB           BRA    JENDD
632 546C 8D CA   C356    BSR    I3X6        XRI-EXCLUSIVE OR IMM W/A
633 546E 7C 5AE0         INC    CYAC
634 5471 E8 00           EOR B  0,X
635 5473 20 D2           BRA    JENDD
636 5475 8D C1   C366    BSR    I3X6        ORI-OR IMM WITH A
637 5477 7C 5AE0         INC    CYAC
638 547A EA 00           ORA B  0,X
639 547C 20 C9           BRA    JENDD
640 547E 8D B8   C376    BSR    I3X6        CPI-COMPARE IMM WITH A
641 5480 F1 00           CMP B  0,X
642 5482 20 C3           BRA    JENDD

644 5484 7C 5AE1 C373    INC    EIFLG       EI-ENABLE INTERRUPTS
645 5487 7E 5378         JMP    END

647 548A 7F 5AE1 C363    CLR    EIFLG       DI-DISABLE INTERRUPTS
648 548D 7E 5378         JMP    END

650 5490         C027    EQU    *           SHIFT INSTRUCTIONS
651 5490 79 5AB5         ROL    ACC
652 5493 07      C027A   TPA
653 5494 84 01   C027B   AND A  #1          USE CARRY COND ONLY
654 5496 C6 FE           LDA B  #$FE         CHANGE CARRY IN COND CODE ONLY
655 5498 F4 5AB6         AND B  ECC
656 549B 1B              ABA
657 549C B7 5AB6         STA A  ECC
658 549F 7E 5378         JMP    END
```

```
661 54A2 76 5AB5 C037   ROR     ACC
662 54A5 20 EC          BRA     C027A
663 54A7 79 5AB5 C007   ROL     ACC
664 54AA 07       C007A TPA
665 54AB 36             PSH A
666 54AC 84 01          AND A   #1
667 54AE C6 FE          LDA B   #$FE
668 54B0 F4 5AB5        AND B   ACC
669 54B3 1B             ABA
670 54B4 B7 5AB5        STA A   ACC
671 54B7 32             PUL A
672 54B8 20 DA          BRA     C027B
673 54BA 76 5AB5 C017   ROR     ACC
674 54BD 20 EB          BRA     C007A

676 54BF BD 56ED C072   JSR     ADREXC      LOAD & SSTORE ACC IMM
677 54C2 A6 00          LDA A   0,X
678 54C4 B7 5AB5        STA A   ACC
679 54C7 BD 53A1 C072A  JSR     PCP2
680 54CA 7E 5378        JMP     END

682 54CD B6 5AB5 C062   LDA A   ACC
683 54D0 FE 5ABF        LDX     EPC
684 54D3 A7 00          STA A   0,X
685 54D5 20 F0          BRA     C072A
686 54D7 7E 5378 C000   JMP     END         NO OP

688 54DA        RT00    EQU     *           D7,D6 = 00
689 54DA 17             TBA                 GET FRESH INSTRUCTION BYTE
690 54DB 84 3F          AND A   #$3F        USE SSS AND DDD ONLY
691 54DD 48             ASL A               TIMES 2 FOR TABLE ADDRESS
692 54DE B7 54E9        STA A   RT00A+1     BUILD INSTRUCTION
693 54E1 CE 54EC        LDX     #T0XX       POINT TO BRANCH TABLE
694 54E4 B6 5AB6        LDA A   ECC         GET COND CODE FOR
695             *   TEST CODE TYPE INSTRUCITON
696 54E7 06             TAP
697 54E8 FE 00  RT00A   LDX     0,X         GET BRANCH ADDRESS
698 54EA 6E 00          JMP     0,X         EXECUTE 8080 INSTRUCTION OF 0XX TYPE

700 54EC        T0XX    EQU     *
701 54EC 54 D7          FDB     C000
702 54EE 56 D1          FDB     C001        LXI
703 54F0 56 56          FDB     C002
704 54F2 55 E1          FDB     C003
705 54F4 55 6F          FDB     INCR
706 54F6 55 8E          FDB     DECR
707 54F8 55 9E          FDB     MVIR
708 54FA 54 A7          FDB     C007
709 54FC 52 CB          FDB     NEM
710 54FE 56 46          FDB     DAD
711 5500 56 6A          FDB     C012
712 5502 55 FF          FDB     C013
713 5504 55 6F          FDB     INCR
714 5506 55 8E          FDB     DECR
715 5508 55 9E          FDB     MVIR
716 550A 54 BA          FDB     C017
717 550C 52 CB          FDB     NEM
718 550E 56 D9          FDB     C021        LXI
719 5510 56 5D          FDB     C022
720 5512 55 EB          FDB     C023
721 5514 55 6F          FDB     INCR
722 5516 55 8E          FDB     DECR
723 5518 55 9E          FDB     MVIR
724 551A 54 90          FDB     C027
725 551C 52 CB          FDB     NEM
726 551E 56 46          FDB     DAD
727 5520 56 71          FDB     C032
728 5522 56 09          FDB     C033
729 5524 55 6F          FDB     INCR
730 5526 55 8E          FDB     DECR
731 5528 55 9E          FDB     MVIR
732 552A 54 A2          FDB     C037
733 552C 52 CB          FDB     NEM

734 552E 56 E1          FDB     C041        LXI
735 5530 56 AF          FDB     C042
736 5532 55 F5          FDB     C043
737 5534 55 6F          FDB     INCR
738 5536 55 8E          FDB     DECR
739 5538 55 9E          FDB     MVIR
740 553A 56 7E          FDB     C047
741 553C 52 CB          FDB     NEM
742 553E 56 46          FDB     DAD
743 5540 56 C2          FDB     C052
744 5542 56 13          FDB     C053
745 5544 55 6F          FDB     INCR
746 5546 55 8E          FDB     DECR
747 5548 55 9E          FDB     MVIR
748 554A 56 94          FDB     C057
749 554C 52 CB          FDB     NEM
750 554E 55 CE          FDB     C061
751 5550 54 CD          FDB     C062
752 5552 55 D7          FDB     C063
753 5554 55 6F          FDB     INCR
754 5556 55 8E          FDB     DECR
755 5558 55 B4          FDB     C066
756 555A 56 9A          FDB     C067
757 555C 52 CB          FDB     NEM
758 555E 56 52          FDB     C071
759 5560 54 BF          FDB     C072
760 5562 56 1D          FDB     C073
761 5564 55 6C          FDB     C074
762 5566 55 8B          FDB     C075
763 5568 55 9B          FDB     C076
764 556A 56 A6          FDB     C077

766 556C 7A 5ABA C074   DEC     DDD         POINT TO CORRECT REGISTER
767 556F B6 5ABA INCR   LDA A   DDD         INCREMENT REGISTERS
768 5572 B7 5579        STA A   INCRA+1
769 5575 CE 5AAF        LDX     #REGS
770 5578 6C 00    INCRA INC     0,X
771 557A 07       INCRB TPA
772 557B 84 FE          AND A   #$FE        CHANGE ALL BUT CARRY FLAG
773 557D F6 5AB6        LDA B   ECC
774 5580 C4 01          AND B   #1          SAVE OLD CARRY CONDITION
775 5582 27 01          BEQ     INCRC
776 5584 4C             INC A               TURN ON CARRY
777 5585 B7 5AB6 INCRC  STA A   ECC
778 5588 7E 5378        JMP     END

780 558B 7A 5ABA C075   DEC     DDD
781 558E B6 5ABA DECR   LDA A   DDD         DECREMENT REGISTERS
782 5591 B7 5598        STA A   DECRA+1
783 5594 CE 5AAF        LDX     #REGS
784 5597 6A 00    DECRA DEC     0,X
785 5599 20 DF          BRA     INCRB
786 559B 7A 5ABA C076   DEC     DDD
787 559E FE 5ABF MVIR   LDX     EPC         MOVE IMM REGISTERS
788 55A1 E6 00          LDA B   0,X
789 55A3 CE 5AAF        LDX     #REGS
790 55A6 B6 5ABA        LDA A   DDD
791 55A9 B7 55AD        STA A   MVIRA+1
792 55AC E7 00    MVIRA STA B   0,X
793 55AE BD 53AA MVIRB  JSR     PCP1
794 55B1 7E 5378        JMP     END

796 55B4 FE 5ABF C066   LDX     EPC
797 55B7 E6 00          LDA B   0,X
798 55B9 FE 5AB3        LDX     H
799 55BC E7 00          STA B   0,X
800 55BE 20 EE          BRA     MVIRB

802 55C0        C064    EQU     *           INCREMENT MEMORY
803 55C0 FE 5AB3        LDX     H
804 55C3 6C 00          INC     0,X
805 55C5 20 B3          BRA     INCRB
```

```
807 55C7          C065   EQU   *          DECREMENT MEMORY
808 55C7 FE 5AB3         LDX   H
809 55CA 6A 00           DEC   0,X
810 55CC 20 AC           BRA   INCRB

812 55CE          C061   EQU   *          LOAD IMM STACK POINTER
813 55CE BD 56ED         JSR   ADREXC
814 55D1 FF 5AB7         STX   EMSTK
815 55D4 7E 56E7         JMP   LXIB

817 55D7 FE 5AB7 C063    LDX   EMSTK
818 55DA 08              INX
819 55DB FF 5AB7         STX   EMSTK
820 55DE 7E 5378         JMP   END

822 55F1 FE 5AAF C003    LDX   BREG
823 55F4 08              INX
824 55F5 FF 5AAF         STX   BREG
825 55F8 7E 5378         JMP   END

827 55EB FE 5AB1 C023    LDX   D
828 55FE 08              INX
829 55EF FF 5AB1         STX   D
830 55F2 7E 5378         JMP   END

832 55F5 FE 5AB3 C043    LDX   H
833 55F8 08              INX
834 55F9 FF 5AB3         STX   H
835 55FC 7E 5378         JMP   END

837 55FF FE 5AAF C013    LDX   BREG
838 5602 09              DEX
839 5603 FF 5AAF         STX   BREG
840 5606 7E 5378         JMP   END

842 5609 FE 5AB1 C033    LDX   D
843 560C 09              DEX
844 560D FF 5AB1         STX   D
845 5610 7E 5378         JMP   END

847 5613 FE 5AB3 C053    LDX   H
848 5616 09              DEX
849 5617 FF 5AB3         STX   H
850 561A 7E 5378         JMP   END
853 561D FE 5AB7 C073    LDX   EMSTK
854 5620 09              DEX
855 5621 FF 5AB7         STX   EMSTK
856 5624 7E 5378         JMP   END
857 5627 CE 5AAF DADR    LDX   #REGS
858 562A E6 00   DAD1    LDA B 0,X        LOW SORCE
859 562C A6 00   DAD2    LDA A 0,X        HIGH SOURCE
860 562E AB 05           ADD A 5,X
861 5630 E9 04           ADC B 4,X
862 5632 36              PSH A
863 5633 37              PSH B
864 5634 07              TPA
865 5635 84 01           AND A #1
866 5637 BA 5AB6         ORA A ECC
867 563A B7 5AB6         STA A ECC
868 563D 32              PUL A
869 563E A7 04           STA A 4,X
870 5640 32              PUL A
871 5641 A7 05           STA A 5,X
872 5643 7E 5378         JMP   END

874 5646 F6 5ABA DAD     LDA B DDD        ADD REG PAIRS
875 5649 F7 562D DAD5    STA B DAD2+1
876 564C 5A              DEC B
877 564D F7 562B         STA B DAD1+1
878 5650 20 D5           BRA   DADR
879 5652 C6 09   C071    LDA B #9
880 5654 20 F3           BRA   DAD5
881 5656 CE 5AAF C002    LDX   #REGS
882 5659 EE 00           LDX   0,X
883 565B 20 05           BRA   C022A
884 565D CE 5AAF C022    LDX   #REGS
885 5660 EE 02           LDX   2,X
886 5662 B6 5AB5 C022A   LDA A ACC
887 5665 A7 00           STA A 0,X
888 5667 7E 5378         JMP   END

890 566A CE 5AAF C012    LDX   #REGS
891 566D EE 00           LDX   0,X
892 566F 20 05           BRA   C032A
893 5671 CE 5AAF C032    LDX   #REGS
894 5674 EE 02           LDX   2,X
895 5676 A6 00   C032A   LDA A 0,X
896 5678 B7 5AB5         STA A ACC
897 567B 7E 5378         JMP   END

899 567E B6 5AB5 C047    LDA A ACC        DECIMAL ADJUST
900 5681 36              PSH A
901 5682 B6 5AB6         LDA A ECC
902 5685 06              TAP
903 5686 32              PUL A
904 5687 19              DAA
905 5688 36              PSH A
906 5689 07              TPA
907 568A B7 5AB6         STA A ECC
908 568D 32              PUL A
909 568E B7 5AB5         STA A ACC
910 5691 7E 5378         JMP   END

912 5694 73 5AB5 C057    COM   ACC
913 5697 7E 5378         JMP   END

915 569A B6 5AB6 C067    LDA A ECC        SET CARRY
916 569D 06              TAP
917 569E 0D       C067B  SEC
918 569F 07       C067A  TPA
919 56A0 B7 5AB6         STA A ECC
920 56A3 7E 5378         JMP   END

922 56A6 B6 5AB6 C077    LDA A ECC        COMPLIMENT CARRY
923 56A9 06              TAP
924 56AA 24 F2           BCC   C067B
925 56AC 0C              CLC
926 56AD 20 F0           BRA   C067A

928 56AF          C042   EQU   *          STORE H & L DIRECT
929 56AF BD 56ED         JSR   ADREXC     PUT 8080 ADDRESS IN X-REG
930 56B2 B6 5AB3         LDA A H
931 56B5 A7 01           STA A 1,X
932 56B7 B6 5AB4         LDA A L
933 56BA A7 00           STA A 0,X
934 56BC BD 53A1 C042A   JSR   PCP2
935 56BF 7E 5378         JMP   END

937 56C2          C052   EQU   *          LOAD H&L DIRECT
938 56C2 BD 56ED         JSR   ADREXC     PUT 8080 ADDRES IN X-REG
939 56C5 A6 01           LDA A 1,X
940 56C7 B7 5AB3         STA A H
941 56CA A6 00           LDA A 0,X
942 56CC B7 5AB4         STA A L
943 56CF 20 EB           BRA   C042A
944                *
945                * LOAD IMMEDIATE REGISTER PAIR
946                *
947 56D1 BD 56ED C001    JSR   ADREXC
948 56D4 FF 5AAF         STX   BREG
949 56D7 20 0E           BRA   LXIB
950 56D9 BD 56ED C021    JSR   ADREXC
951 56DC FF 5AB1         STX   D
952 56DF 20 06           BRA   LXIB
953 56E1 BD 56ED C041    JSR   ADREXC
954 56E4 FF 5AB3         STX   H
```

```
 955 56F7 BD 53A1 LXIB   JSR    PCP2
 956 56FA 7E 5378        JMP    END

 958              * GET 8080 ADDRESS IN LOW-HIGH FORMAT
 959              * AND EXCHANGE IT INTO 6800 HIGH-LOW
 960              * FORMAT IN 6800 X-REG & TMP
 961              *
 962 56ED FE 5ABF ADREXC LDX    EPC
 963 56F0 A6 00          LDA A  0,X        8080 LOW
 964 56F2 B7 5ABC        STA A  TMP+1      6800 LOW
 965 56F5 A6 01          LDA A  1,X        8080 HIGH
 966 56F7 B7 5ABB        STA A  TMP        6800 HIGH
 967 56FA FE 5ABB        LDX    TMP
 968 56FD 39             RTS

 970 56FE          RA    EQU    *          ADD REGISTER TO A
 971 56FE B6 5AB9        LDA A  SSS
 972 5701 81 07          CMP A  #7
 973 5703 26 02          BNE    RA1
 974 5705 86 06          LDA A  #$6        POINT TO CORRECT POSITON FOR REG 7
 975 5707          RA1   EQU    *
 976 5707 B7 5717        STA A  XCT+1
 977 570A CE 5AAF        LDX    #REGS
 978 570D B6 5AB5        LDA A  ACC
 979 5710 36             PSH A
 980 5711 B6 5AB6        LDA A  ECC
 981 5714 06             TAP
 982 5715 32             PUL A

 984 5716          XCT   RMB    2          BUILD INSTRUCTION HERE

 986 5718 36             PSH A
 987 5719 07             TPA
 988 571A B7 5AB6        STA A  ECC
 989 571D 32             PUL A
 990 571E B7 5AB5        STA A  ACC
 991 5721 7E 5378        JMP    END

 993 5724          RT10  EQU    *
 994 5724 B6 5AB9        LDA A  SSS
 995 5727 81 06          CMP A  #6
 996 5729 27 4D          BEQ    C2X6
 997 572B B6 5ABA        LDA A  DDD
 998 572E 81 04          CMP A  #4
 999 5730 27 2D          BEQ    C24X
1000 5732 81 05          CMP A  #5
1001 5734 27 30          BEQ    C25X
1002 5736 81 06          CMP A  #6
1003 5738 27 33          BEQ    C26X
1004 573A 81 07          CMP A  #7
1005 573C 27 36          BEQ    C27X
1006 573E 81 01          CMP A  #1
1007 5740 27 11          BEQ    C21X
1008 5742 81 02          CMP A  #2
1009 5744 27 11          BEQ    C22X
1010 5746 81 03          CMP A  #3
1011 5748 27 11          BEQ    C23X
1012 574A 20 00          BRA    C20X
1013 574C 86 AB   C20X   LDA A  #$AB        ADD
1014 574E B7 5716 XCTA   STA A  XCT
1015 5751 20 AB          BRA    RA
1016 5753 86 A9   C21X   LDA A  #$A9        ADD WITH CARRY
1017 5755 20 F7          BRA    XCTA
1018 5757 86 A0   C22X   LDA A  #$A0        SUB
1019 5759 20 F3          BRA    XCTA
1020 575B 86 A2   C23X   LDA A  #$A2        SUB WITH BARROW
1021 575D 20 EF          BRA    XCTA
1022 575F 7C 5ADF C24X   INC    CY
1023 5762 86 A4          LDA A  #$A4        AND
1024 5764 20 E8          BRA    XCTA
1025 5766 7C 5AE0 C25X   INC    CYAC
1026 5769 86 A8          LDA A  #$A8        EXCLUSIVE OR
1027 576B 20 E1          BRA    XCTA
1028 576D 7C 5AE0 C26X   INC    CYAC
1029 5770 86 AA          LDA A  #$AA        OR, INCLUSIVE
1030 5772 20 DA          BRA    XCTA
1031 5774 86 A1   C27X   LDA A  #$A1
1032 5776 20 D6          BRA    XCTA
1033 5778 FE 5AB3 C2X6   LDX    H          LOAD HL BYTE
1034 577B A6 00          LDA A  0,X        GET MEMORY BYTE
1035 577D B7 5ABB        STA A  TMP
1036 5780 86 0C          LDA A  #12        POINT TO TMP REG
1037 5782 B7 5AB9        STA A  SSS
1038 5785 86 5ABA        LDA A  DDD
1039 5788 81 00          CMP A  #$00
1040 578A 27 C0          BEQ    C20X
1041 578C 81 01          CMP A  #$01
1042 578E 27 C3          BEQ    C21X
1043 5790 81 02          CMP A  #$02
1044 5792 27 C3          BEQ    C22X
1045 5794 81 03          CMP A  #$03
1046 5796 27 C3          BEQ    C23X
1047 5798 81 07          CMP A  #$07
1048 579A 27 D8          BEQ    C27X
1049 579C 7C 5AE0        INC    CYAC
1050 579F 81 04          CMP A  #$04
1051 57A1 27 BC          BEQ    C24X
1052 57A3 81 05          CMP A  #$05
1053 57A5 27 BF          BEQ    C25X
1054 57A7 81 06          CMP A  #$06
1055 57A9 27 C2          BEQ    C26X
1056 57AB 7E 52CB        JMP    NEM

1058              *  EMULATE 8080 INPUT PORT WITH IRQ FROM INPUT
1059              *  DEVICE. EITHER AN ACIA OR MIKBUG PIA.
1060              *
1061 57AE         GETTTY EQU    *
1062 57AE BD E1AC       JSR    INCH
1063              *
1064              * FOR ACIA, REPLACE WITH : LDAA $8009
1065              *
1066 57B1 B7 5AE3       STA A  TTYCH
1067 57B4 7C 5AE2       INC    TTYRDY
1068 57B7 3B            RTI

1070 57B8         IN    EQU    *          INPUT
1071 57B8 FE 5ABF       LDX    EPC
1072 57BB A6 00         LDA A  0,X        GET INPUT ADDRESS
1073 57BD 27 07         BEQ    PORT0I     TV STATUS
1074 57BF 81 01         CMP A  #1
1075 57C1 27 11         BEQ    PRT1I
1076 57C3 7E 52CB       JMP    NEM

1078 57C6 86 02   PORT0I LDA A  #2         INDICATE OUTPUT READY
1079 57C8 7D 5AE2       TST    TTYRDY     TTY CHARACTER READY?
1080 57CB 27 02         BEQ    TSTIN      BRANCH NO
1081 57CD 8A 20         ORA A  #$20       INDICATE INPUT CHARACTER READY
1082 57CF B7 5AB5 TSTIN  STA A  ACC
1083 57D2 20 0E         BRA    IOEND
1084 57D4 B6 5AE3 PRT1I  LDA A  TTYCH      GET TTY IN CHARACTER
1085 57D7 B7 5AB5       STA A  ACC
1086 57DA 7F 5AE2       CLR    TTYRDY
1087 57DD 7C 5AE4       INC    ECHOA      PREVENT THIS IN CHARACTER
1088              *                  FROM ECHOING IF DESIRED
1089 57E0 20 00         BRA    IOEND

1091 57E2 BD 53AA IOEND  JSR    PCP1
1092 57E5 7E 5378       JMP    END

1094 57E8         OUT   EQU    *          OUTPUT
1095 57E8 FE 5ABF       LDX    EPC
1096 57EB A6 00         LDA A  0,X
1097 57ED 81 01         CMP A  #$01
1098 57EF 27 03         BEQ    PRT1O
1099 57F1 7E 52CB       JMP    NEM
```

```
1101 57F4 B6 5AB5 PRT10  LDA A  ACC
1102 57F7 7D 5AE5        TST    ECHOFG      IS ECHO ON?
1103 57FA 26 05          BNE    ECHO        BR YES TO ECHO
1104 57FC 7D 5AE4        TST    ECHOA       NO, PRINT THIS CHARACTER?
1105 57FF 26 03          BNE    PRT10A      NO

1107 5801         ECHO   EQU    *           YES
1108 5801 BD E1D1        JSR    OUTCH
1109 5804         PRT10A EQU    *
1110 5804 7F 5AE4        CLR    ECHOA
1111 5807 20 D9          BRA    IOEND

1113 5809         INTST  EQU    *           TEST FOR INTERRUPT
1114 5809 39             RTS

1116 580A         OCTIN  EQU    *           INPUT ONE OCTAL DIGIT
1117 580A BD E1AC        JSR    INCH        1ST OCTAL DIGIT
1118 580D 8D 45          BSR    VALOCT      VALID OCTAL?
1119 580F 7D 5AEB        TST    LDXT        TEST FOR EXIT
1120 5812 26 2F          BNE    OUTX
1121 5814 7D 5AEC        TST    SPXT        SPACE OVER MEM LOC?
1122 5817 26 2A          BNE    OUTX
1123 5819 7D 5AED        TST    BKXT        BACKWARD OVER MEM LOC?
1124 581C 26 25          BNE    OUTX
1125 581E 8D 30          BSR    ASLA3       SHIFT LEFT THREE
1126 5820 B7 5AE6        STA A  OCTDIG      SAVE PARTIAL DIGIT
1127 5823 BD E1AC        JSR    INCH        2ND  OCTAL DIGIT
1128 5826 8D 2C          BSR    VALOCT      VALID OCTAL?
1129 5828 BA 5AE6        ORA A  OCTDIG      BUILD MORE OF OCTAL DIGIT
1130 582B 8D 23          BSR    ASLA3       SHIFT 3 LEFT
1131 582D B7 5AE6        STA A  OCTDIG      SAVE PARTIAL
1132 5830 BD E1AC        JSR    INCH        3RD OCTAL DIGIT
1133 5833 8D 1F          BSR    VALOCT      VALID OCTAL?
1134 5835 BA 5AE6        ORA A  OCTDIG      COMPLETED OCTAL DIGIT
1135 5838 B7 5AE6        STA A  OCTDIG      SAVE IT
1136 583B 86 0D          LDA A  #'
1137 583D BD E1D1        JSR    OUTCH
1138 5840 B6 5AE6        LDA A  OCTDIG
1139 5843 39      OUTX   RTS

1141 5844 7C 5AEB OCTINX INC    LDXT        SET EXIT FLAG
1142 5847 39             RTS

1144 5848 7C 5AEC OCTSP  INC    SPXT        SET SPACE OVER MEM LOC FLAG
1145 584B 39             RTS

1147 584C 7C 5AED OCTBK  INC    BKXT        BACK OVER MEM LOC FLAG SET
1148 584F 39             RTS

1150 5850 48      ASLA3  ASL A
1151 5851 48             ASL A
1152 5852 48             ASL A
1153 5853 39             RTS

1155 5854         VALOCT EQU    *
1156 5854 81 2E          CMP A  #'.
1157 5856 27 EC          BEQ    OCTINX
1158 5858 81 0D          CMP A  #'
1159 585A 27 EC          BEQ    OCTSP
1160 585C 81 2D          CMP A  #'-
1161 585E 27 EC          BEQ    OCTBK
1162 5860 81 37          CMP A  #'7
1163 5862 22 07          BHI    OCTER       YES
1164 5864 81 2F          CMP A  #$2F        LESS THAN ZERO?
1165 5866 23 03          BLS    OCTER       YES
1166 5868 84 07          AND A  #$07        STRIP
1167 586A 39             RTS

1169 586B 86 3F   OCTER  LDA A  #'?
1170 586D BD E1D1        JSR    OUTCH
1171 5870 32             PUL A              GET HI RETURN ADDRESS
1172 5871 B7 5AE7        STA A  OCTWRK
1173 5874 32             PUL A              GET LO RETURN ADDRESS
1174 5875 B7 5AE8        STA A  OCTWRK+1
1175 5878 FF 5AE9        STX    OCTX        SAVE INDEX
1176 587B FE 5AE7        LDX    OCTWRK
1177 587E 09             DEX
1178 587F 09             DEX
1179 5880 09             DEX
1180 5881 09             DEX
1181 5882 09             DEX
1182 5883 FF 5AE7        STX    OCTWRK      BACK UP TO INCH
1183 5886 B6 5AE8        LDA A  OCTWRK+1
1184 5889 36             PSH A
1185 588A B6 5AE7        LDA A  OCTWRK
1186 588D 36             PSH A
1187 588E FE 5AE9        LDX    OCTX        RETURN INDEX
1188 5891 39             RTS

1190              * BUILD AN OCTAL ADDRESS AND SAVE AS HEX
1191 5892 BD 580A OCTADD JSR    OCTIN       GET LO
1192 5895 B7 5AEF        STA A  OCADDR+1
1193 5898 BD 580A        JSR    OCTIN       GET HI
1194 589B B7 5AEE        STA A  OCADDR
1195 589E FE 5AEE        LDX    OCADDR
1196 58A1 39             RTS

1198 58A2         LDOCT  EQU    *           LOAD MEMORY WITH 8080 PRGM DATA
1199 58A2 BD 500B        JSR    CRLF
1200 58A5 7F 5AEB        CLR    LDXT        CLEAR EXIT FLAG
1201 58A8 BD 5892        JSR    OCTADD      GET START ADDRESS
1202 58AB 7D 5AEB        TST    LDXT        EXIT?
1203 58AE 26 39          BNE    LDAX        YES
1204 58B0 FF 5AF0 LDOCTA STX    LDOCTB
1205 58B3 7F 5AFC        CLR    SPXT
1206 58B6 7F 5AED        CLR    BKXT
1207 58B9 BD 500B        JSR    CRLF
1208 58BC CE 5AF0        LDX    #LDOCTB
1209 58BF BD E0C8        JSR    OUT4HS
1210 58C2 FE 5AF0        LDX    LDOCTB
1211 58C5 A6 00          LDA A  0,X
1212 58C7 BD 58ED        JSR    OUTOCT
1213 58CA 86 0D          LDA A  #'
1214 58CC BD E1D1        JSR    OUTCH
1215 58CF BD 580A        JSR    OCTIN
1216 58D2 7D 5AEB        TST    LDXT
1217 58D5 26 12          BNE    LDAX
1218 58D7 7D 5AEC        TST    SPXT
1219 58DA 26 0A          BNE    SKP
1220 58DC 7D 5AED        TST    BKXT
1221 58DF 26 09          BNE    SKPB
1222 58E1 FE 5AF0        LDX    LDOCTB
1223 58E4 A7 00          STA A  0,X
1224 58E6 08      SKP    INX
1225 58E7 20 C7          BRA    LDOCTA

1227 58E9         LDAX   EQU    *           EXIT ROUTINE
1228 58E9 39             RTS

1230 58EA 09      SKPB   DEX
1231 58EB 20 C3          BRA    LDOCTA

1233 58ED         OUTOCT EQU    *           OUTPUT OCTAL CHAR IN A REG
1234 58ED 16             TAB
1235 58EE 84 C0          AND A  #$C0
1236 58F0 8D 19          BSR    LSRA3
1237 58F2 8D 17          BSR    LSRA3
1238 58F4 8A 30          ORA A  #$30
1239 58F6 BD E1D1        JSR    OUTCH       FIRST DIGIT
1240 58F9 17             TBA
1241 58FA 84 38          AND A  #$38
1242 58FC 8D 0D          BSR    LSRA3
1243 58FE 8A 30          ORA A  #$30
1244 5900 BD E1D1        JSR    OUTCH       2ND DIGIT
1245 5903 17             TBA
1246 5904 84 07          AND A  #$07
```

```
1247 5906 8A 30          ORA A  #$30
1248 5908 7E E1D1        JMP    OUTCH       3RD DIGIT

1250 590B 44       LSRA3 LSR A
1251 590C 44             LSR A
1252 590D 44             LSR A
1253 590E 39             RTS

1255 590F          HELP  EQU    *
1256 590F 0D             FCB    $D,$A
1257 5911 41             FCC    /AB = SET ADDRESS  BREAK/
1258 5928 0D             FCB    $D,$A
1259 592A 43             FCC    /CB = CLEAR ADDRESS BREAK/
1260 5942 0D             FCB    $D,$A
1261 5944 43             FCC    /CR = CLEAR REGISTERS/
1262 5958 0D             FCB    $D,$A
1263 595A 43             FCC    /CT = CONTINUE 8080 DEBUG/
1264 5972 0D             FCB    $D,$A
1265 5974 45             FCC    /EO = TURN ECHO ON/
1266 5985 0D             FCB    $D,$A
1267 5987 45             FCC    /EF = TURN ECHO OFF/
1268 5999 0D             FCB    $D,$A
1269 599B 4C             FCC    /LO = LOAD OCTAL DATA/
1270 59AF 0D             FCB    $D,$A
1271 59B1 50             FCC    /PC = SET 8080 PROGRAM COUNTER/
1272 59CE 0D             FCB    $D,$A
1273 59D0 52             FCC    /RG = DISPLAY 8080 REGISTERS/
1274 59FB 0D             FCB    $D,$A
1275 59FD 52             FCC    /RU = RUN OR CONTINUE PROGRAM/
1276 5A09 0D             FCB    $D,$A
1277 5A0B 53             FCC    /SF = TURN SINGLE CYCLE OFF/
1278 5A25 0D             FCB    $D,$A
1279 5A27 53             FCC    /SO = TURN SINGLE CYCLE ON/
1280 5A40 0D             FCB    $D,$A
1281 5A42 53             FCC    /SPACE KEY = RU/
1282 5A50 0D             FCB    $D,$A
1283 5A52 45             FCC    /EX = EXIT TO MIKBUG/
1284 5A65 0D             FCB    $D,$A
1285 5A67 04             FCB    $4

1287 5A68 48       HTMSG FCC    /HALT PC = /
1288 5A72 04             FCB    4
1289 5A73 41       ABMSG FCC    /ADDRESS BREAK = /
1290 5AA3 04             FCB    4
1291 5AA4 20       PCMSG FCC    / = /
1292 5AA7 04             FCB    4
1293 5AA8 4E       NEMMSG FCC   /NOT EMULATED CODE/
1294 5AB9 04             FCB    4
1295 5ABA 0D       CRLFS FCB    $D
1296 5ABB 0A       LF    FCB    $A,4

1298 5ABD          HELPRT EQU   *
1299 5ABD 8D 06          BSR    HOMEUP
1300 5ABF CE 590F        LDX    #HELP
1301 5AA2 7E E07E        JMP    PDATA1

1303 5AA5          HOMEUP EQU   *
1304 5AA5 86 1B          LDA A  #$1B        (ESC *) FOR SOROC IQ120
1305 5AA7 BD E1D1        JSR    OUTCH
1306 5AAA 86 2A          LDA A  #$2A
1307 5AAC 7E E1D1        JMP    OUTCH

1309               * RAM STORAGE AREA

1311 5AAF          REGS  EQU    *
1312 5AAF          BREG  RMB    1
1313 5AB0          C     RMB    1
1314 5AB1          D     RMB    1
1315 5AB2          E     RMB    1
1316 5AB3          H     RMB    1
1317 5AB4          L     RMB    1           LOW ORDER ADDR BYTE
1318 5AB5          ACC   RMB    1           8080 ACC
1319 5AB6          ECC   RMB    1           8080 COND CODE REG
1320 5AB7 5A D7    EMSTK FDB    STACK+1     8080 STACK REG
1321 5AB9          SSS   RMB    1           SOURCE FIELD DATA
1322 5ABA          DDD   RMB    1           DESTINATION FIELD DATA
1323 5ABB          TMP   RMB    2
1324 5ABD          NEWPC RMB    2
1325 5ABF          EPC   RMB    2           EMULATED 8080 PC
1326 5AC1          MSTK  RMB    2           SAVE MONITOR STACK POINTER
1327 5AC3                RMB    19          STACK FOR TESTING 8080 INSTRUCTIONS
1328 5AD6          STACK RMB    1
1329 5AD7          MONCD RMB    2           CURRENT MONITOR INSTRUCTION
1330 5AD9          ABFLG RMB    1           ON IF TEST FOR ADDR BREAK
1331 5ADA          ABADDR RMB   2           STOP PC ADDR FOR ADDR BREAK
1332 5ADC          HTFLG RMB    1           HALT FLAG (.NE. IF ON)
1333 5ADD          SCYFLG RMB   1           SINGLE CYCLE FLAG (.NE. IF ON)
1334 5ADE          OPCD  RMB    1           OP CODE IN HEX
1335 5ADF          CY    RMB    1           CARRY CLEAR FLAG
1336 5AE0          CYAC  RMB    1           CARRY AND HALF CARRY CLEAR FLAG
1337 5AE1          EIFLG RMB    1
1338 5AE2          TTYRDY RMB   1
1339 5AE3          TTYCH RMB    1
1340 5AE4 01       ECHOA FCB    1
1341 5AE5 01       ECHOFG FCB   1
1342 5AE6          OCTDIG RMB   1
1343 5AE7          OCTWRK RMB   2
1344 5AE9          OCTX  RMB    2
1345 5AEB          LDXT  RMB    1           EXIT FLAG
1346 5AEC          SPXT  RMB    1           SPACE OVER MEM LOC FLAG
1347 5AED          BKXT  RMB    1           BACK OVER MEM LOC FLAG
1348 5AEE          OCADDR RMB   2
1349 5AF0          LDOCTB RMB   2           SAVE X-REG

1351               *
1352               *  SET UP INITIAL START-UP CONDITIONS
1353               *
1354 A043                ORG    $A043
1355 A043 10             FCB    $10         CC = DISABLE INTERRUPTS
1356 A044 00             FCB    $00         B
1357 A045 00             FCB    $00         A
1358 A046 00 00          FDB    $0000       INX
1359 A048 50 00          FDB    START       PC
1360 A04A          EXTRNL EQU   *

1362                     END
```

SYMBOL TABLE:

ABADDR	5ADA	ABFLG	5AD9	ABMSG	5A73	ACC	5AB5	ADREXC	56ED
ADRSET	50C2	ASLA3	5850	BADDR	E047	BEGIN	5011	BKXT	5AED
BREG	5AAF	C	5AB0	C000	54D7	C001	56D1	C002	5656
C003	55E1	C007	54A7	C007A	54AA	C012	566A	C013	55FF
C017	54BA	C021	56D9	C022	565D	C022A	5662	C023	55EB
C027	5490	C027A	5493	C027B	5494	C032	5671	C032A	5676
C033	5609	C037	54A2	C041	56E1	C042	56AF	C042A	56BC
C043	55F5	C047	567E	C052	56C2	C053	5613	C057	5694
C061	55CE	C062	54CD	C063	55D7	C064	55C0	C065	55C7
C066	5584	C067	569A	C067A	569F	C067B	569E	C071	5652
C072	54BF	C072A	54C7	C073	561D	C074	556C	C075	5588
C076	559B	C077	56A6	C20X	574C	C21X	5753	C22X	5757
C23X	575B	C24X	575F	C25X	5766	C26X	576D	C27X	5774
C2X6	5778	C300	52AF	C302	5425	C304	5406	C306	5443
C310	52B3	C312	5421	C314	5402	C316	5454	C320	52B7
C322	541D	C324	53FE	C326	545A	C330	52BB	C332	5419
C334	53FA	C336	5460	C343	53B2	C346	5466	C351	53F1
C353	53D2	C356	546C	C360	52BF	C362	5429	C363	548A
C364	540A	C366	5475	C370	52C3	C371	53E7	C372	542D
C373	5484	C374	540E	C376	547E	CALL	5317	CD01	51B3
CD10	51B6	CD11	51B9	CLRAB	50BE	CLREGS	511C	CRLF	500B
CRLFS	5A9A	CRY1	5391	CRY2	539D	CRY3	5393	CRYADJ	5383
CY	5ADF	CYAC	5AE0	D	5AB1	DAD	5646	DAD1	562A
DAD2	562C	DAD5	5649	DADR	5627	DDD	5ABA	DECR	558E
DECRA	5597	E	5AB2	ECC	5AB6	ECHF	5118	ECHN	5114
ECHO	5801	ECHOA	5AE4	ECHOFG	5AE5	EIFLG	5AF1	EMSTK	5AB7

```
END     5378   END1    5371   EPC     5ABF   EXTRNL  A04A   GETTTY 57AE
H       5AB3   HALT    52D7   HELP    590F   HELPRT  5A9D   HOMEUP 5AA5
HTFLG   5ADC   HTMSG   5A68   I3X6    5438   IFETCH  5182   IN     57B8
INCH    E1AC   INCR    556F   INCRA   5578   INCRB   557A   INCRC  5585
INTST   5809   IO      A000   IOEND   57E2   JCALL   5416   JENDA  52C5
JENDB   5410   JENDC   542F   JENDD   5447   JHALT   5205   JRET   52C8
JUMP    5339   JUMPA   5435   JUMPX   5329   L       5A84   LDAX   58E9
LDOCT   58A2   LDOCTA  58B0   LDOCTB  5AF0   LDXT    5AEB   LF     5A98
LSRA3   590B   LXIB    56E7   MIKBUG  E0D0   MN1     506F   MN2    507B
MN3     505D   MON4    5049   MON5    5052   MON6    5057   MONCD  5AD7
MONTBL  5080   MONTOR  5021   MONTST  505A   MR      5208   MR1    5213
MSTK    5AC1   MVIR    559E   MVIRA   55AC   MVIRB   55AE   NEM    52CB
NEMMSG  5A88   NEWPC   5ABD   NMIV    5002   OCADDR  5AEE   OCTADD 5892
OCTBK   584C   OCTDIG  5AE6   OCTER   586B   OCTIN   589A   OCTINX 5844
OCTSP   5848   OCTWRK  5AE7   OCTX    5AE9   OPCD    5ADE   OUT    57E8
OUT2HS  E0CA   OUT4HS  E0C8   OUTCH   E1D1   OUTOCT  58FD   OUTX   5843
PCMSG   5A84   PCP1    53AA   PCP2    53A1   PDATA1  E07E   PIA    8004
PIAS    8005   POP     52FA   PORT0I  57C6   PRT1I   57D4   PRT1O  57F4
PRT1OA  5804   PULLA   530F   PULLB   5312   PUSH    52DD   PUSHA  52F1
PUSHB   52F4   RA      56FE   RA1     5707   REG     5088   REGD   5140
REGS    5AAF   RET     5360   RM      51E8   RM1     5200   RST    5342
RT00    54DA   RT00A   54E8   RT01    51BC   RT01A   51D6   RT01B  51E3
RT10    5724   RT11    521D   RT11A   5227   RUN     50D8   RUNA   50FA
RUNB    5109   RUNC    50D8   SCYC    510C   SCYF    5110   SCYFLG 5ADD
SETPC   5173   SKP     58E6   SKP1    50EB   SKPB    58EA   SPXT   5AEC
SSS     5AB9   STACK   5AD6   START   5000   T0XX    54EC   T3XX   522F
TMP     5ABB   TSTIN   57CF   TTYCH   5AE3   TTYRDY  5AF2   VALOCT 5854
XCT     5716   XCTA    574F
```

```
CROSS REFERENCE- EM8080.SRC
SYMBOL  DEFINED  REFERENCED
======  =======  ==========

ABADDR  1331 :  140   152
ABFLG   1330 :  132   142   149
ABMSG   1289 :  137
ACC     1318 :  193   606   616   651   661   663   668   670   673   678
                682   886   896   899   909   912   978   990  1082  1085
               1101
ADREXC   962 :  676   813   929   938   947   950   953
ADRSET   135 :  103
ASLA3   1150 : 1125  1130
BADDR    22 :  139   221
BEGIN    38 :   30    31
BKXT    1347 : 1123  1147  1206  1220
BREG    1312 :  822   824   837   839   948
C       1313 :  187
C000     686 :  701
C001     947 :  702
C002     881 :  703
C003     822 :  704
C007     663 :  708
C007A    664 :  674
C012     890 :  711
C013     837 :  712
C017     673 :  716
C021     950 :  718
C022     884 :  719
C022A    886 :  883
C023     827 :  720
C027     650 :  724
C027A    652 :  662
C027B    653 :  672
C032     893 :  727
C032A    895 :  892
C033     842 :  728
C037     661 :  732
C041     953 :  734
C042     928 :  735
C042A    934 :  943
C043     832 :  736
```

```
SYMBOL  DEFINED  REFERENCED
======  =======  ==========

C047     899 :  740
C052     937 :  743
C053     847 :  744
C057     912 :  748
C061     812 :  750
C062     682 :  751
C063     817 :  752
C064     802 :
C065     807 :
C066     796 :  755
C067     915 :  756
C067A    918 :  926
C067B    917 :  924
C071     879 :  758
C072     676 :  759
C072A    679 :  685
C073     853 :  760
C074     766 :  761
C075     780 :  762
C076     786 :  763
C077     922 :  764
C20X    1013 : 1012  1040
C21X    1016 : 1007  1042
C22X    1018 : 1009  1044
C23X    1020 : 1011  1046
C24X    1022 :  999  1051
C25X    1025 : 1001  1053
C26X    1028 : 1003  1055
C27X    1031 : 1005  1048
C2X6    1033 :  996
C300     391 :  323
C302     594 :  325
C304     575 :  327
C306     611 :  329
C310     393 :  331
C312     592 :  333
C314     573 :  335
C316     620 :  337
C320     395 :  339
C322     590 :  341
C324     571 :  343
C326     623 :  345
C330     397 :  347
C332     588 :  349
C334     569 :  351
C336     626 :  353
C343     530 :  358
C346     629 :  361
C351     561 :  364
C353     546 :  366
C356     632 :  369
C360     399 :  371
C362     596 :  373
C363     647 :  374
C364     577 :  375
C366     636 :  377
C370     401 :  379
C371     555 :  380
C372     598 :  381
C373     644 :  382
C374     579 :  383
C376     640 :  385
CALL     443 :  336   583
CD01     254 :  249
CD10     257 :  251
CD11     260 :  247
CLRAB    132 :  105
CLREGS   183 :  109
CRLF      35 :   52    72   129   136   141   167   200   210   223   406
                409  1199  1207
```

SYMBOL	DEFINED	REFERENCED								
CRLFS	1295 :	35								
CRY1	508 :	503								
CRY2	514 :	505								
CRY3	509 :	515								
CRYADJ	500 :	497								
CY	1335 :	504	510	1022						
CYAC	1336 :	502	511	633	637	1025	1028	1049		
D	1314 :	188	549	552	827	829	842	844	951	
DAD	874 :	710	726	742						
DAD1	858 :	877								
DAD2	859 :	875								
DADS	875 :	880								
DADR	857 :	878								
DDD	1322 :	240	268	276	285	290	417	432	473	766	767
		780	781	786	790	874	997	1038			
DECR	781 :	706	714	722	730	738	746	754			
DECRA	784 :	782									
E	1315 :	189									
ECC	1319 :	192	318	501	509	607	615	655	657	694	773
		777	866	867	901	907	915	919	922	980	988
ECHF	180 :	123									
ECHN	177 :	121									
ECHO	1107 :	1103									
ECHOA	1340 :	1087	1104	1110							
ECHOFG	1341 :	177	180	1102							
EIFLG	1337 :	644	647								
EMSTK	1320 :	196	211	415	430	446	461	467	483	493	532
		558	814	817	819	853	855				
END1	491 :	426	441	457	479	489	544				
EPC	1325 :	151	165	184	185	215	222	228	233	452	469
		471	496	519	525	605	683	787	796	962	1071
		1095									
EXTRNL	1360 :	127									
GETTTY	1061 :	49									
H	1316 :	190	287	307	538	543	547	550	556	562	798
		803	808	832	834	847	849	930	940	954	1033
HALT	410 :	298									
HELP	1255 :	1300									
HELPRT	1298 :	101									
HOMEUP	1303 :	1299									
HTFLG	1332 :	148	158	410							
HTMSG	1287 :	163									
I3X6	605 :	611	620	623	626	629	632	636	640		
IFETCH	228 :	157									
IN	1070 :	350									
INCH	21 :	55	63	1062	1117	1127	1132				
INCR	767 :	705	713	721	729	737	745	753			
INCRA	770 :	768									
INCRB	771 :	785	805	810							
INCRC	777 :	775									
INTST	1113 :	498									
IO	27 :	50									
IOEND	1091 :	1083	1089	1111							
JCALL	583 :	569	571	573	575	577	579				
JENDA	402 :	392	394	396	398	400					
JENDB	580 :	570	572	574	576	578					
JENDC	600 :	589	591	593	595	597					
JENDD	613 :	622	625	628	631	635	639	642			
JHALT	298 :	286									
JRET	403 :	391	393	395	397	399	401				
JUMP	459 :	326	603								
JUMPA	603 :	588	590	592	594	596	598				
JUMPX	452 :	463									
L	1317 :	191	540	932	942						
LDAX	1227 :	1203	1217								
LDOCT	1198 :	107									
LDOCTA	1204 :	1225	1231								
LDOCTB	1349 :	1204	1208	1210	1222						
LDXT	1345 :	1119	1141	1200	1202	1216					
LF	1296 :	68									

SYMBOL	DEFINED	REFERENCED								
LSRA3	1250 :	1236	1237	1242						
LXIB	955 :	815	949	952						
MIKBUG	18 :	33	125							
MN1	88 :	81	84							
MN2	96 :	90								
MN3	79 :	95								
MON4	70 :	66								
MON5	74 :	58								
MON6	76 :	62								
MONCD	1329 :	59	70	79	82					
MONTBL	99 :	78								
MONTOR	51 :	73	75							
MONTST	78 :	71								
MR	300 :	269								
MR1	306 :	305								
MSTK	1326 :	414	429	445	460	466	482	494	531	
MVIR	787 :	707	715	723	731	739	747			
MVIRA	792 :	791								
MVIRB	793 :	800								
NEM	405 :	334	348	352	355	357	359	363	365	367	368
		384	709	717	725	733	741	749	757	1056	1076
		1099									
NEMMSG	1293 :	407									
NEWPC	1324 :	234	448	450	454	456	477	478	486	488	522
		527	563								
NMIV	31 :	76									
OCADDR	1348 :	1192	1194	1195							
OCTADD	1191 :	1201									
OCTBK	1147 :	1161									
OCTDIG	1342 :	1126	1129	1131	1134	1135	1138				
OCTER	1169 :	1163	1165								
OCTIN	1116 :	1191	1193	1215							
OCTINX	1141 :	1157									
OCTSP	1144 :	1159									
OCTWRK	1343 :	1172	1174	1176	1182	1183	1185				
OCTX	1344 :	1175	1187								
OPCD	1334 :	213	230								
OUT	1094 :	342									
OUT2HS	23 :	202	203	204	205	206	207	208	209	214	
OUT4HS	25 :	166	212	216	1209						
OUTCH	20 :	54	97	1108	1137	1170	1214	1239	1244	1248	1305
		1307									
OUTOCT	1233 :	1212									
OUTX	1139 :	1120	1122	1124							
PCMSG	1291 :	219									
PCP1	525 :	617	793	1091							
PCP2	519 :	444	581	601	679	934	955				
PDATA1	19 :	36	69	138	164	220	408	1301			
PIA	24 :										
PIAS	26 :	40									
POP	428 :	324	340	356	372						
PORT0I	1078 :	1073									
PRT1I	1084 :	1075									
PRT1O	1101 :	1098									
PRT1OA	1109 :	1105									
PULLA	438 :	436									
PULLB	440 :	434									
PUSH	413 :	328	344	360	376						
PUSHA	422 :	419									
PUSHB	424 :	421									
RA	970 :	1015									
RA1	975 :	973									
REG	128 :	113	168								
REGD	199 :	130									
REGS	1311 :	201	264	289	418	433	769	783	789	857	881
		884	890	893	977						
RET	481 :	332	403								
RM	284 :	267									
RM1	295 :	294									
RST	465 :	330	338	346	354	362	370	378	386		

```
SYMBOL  DEFINED REFERENCED
======  ===#### ==========

RT00     688 :  252
RT00A    697 :  692
RT01     263 :  255
RT01A    275 :  274
RT01B    281 :  280
RT10     993 :  258
RT11     311 :  261
RT11A    317 :  315
RUN      145 :   74   111
RUNA     163 :  154   159
RUNB     168 :  161
RUNC     146 :  162
SCYC     171 :  115
SCYF     174 :  119
SCYFLG  1333 :  160   171   174
SETPC    218 :  117
SKP     1224 : 1219
SKP1     156 :  150   153
SKPB    1230 : 1221
SPXT    1346 : 1121  1144  1205  1218
SSS     1321 :  243   266   270   301   971   994  1037
STACK   1328 :  194  1320
START     30 : 1359
TOXX     700 :  693
T3XX     322 :  316
TMP     1323 :  535   537   542   548   551   964   966   967  1035
TSTIN   1082 : 1080
TTYCH   1339 : 1066  1084
TTYRDY  1338 : 1067  1079  1086
VALOCT  1155 : 1118  1128  1133
XCT      984 :  976  1014
XCTA    1014 : 1017  1019  1021  1024  1027  1030  1032


TOTAL SYMBOLS:   266



.TAPE PUNCH 5000 5AE5 A043 A049


S113500052OD07200020464442204D5649520D07BB
S1135010210020464442204330323700D072002020
S113502046444220 4E454D0D07230020464442206D
S1135030441440D07240020464442204330333287
S1135040OD07250020464442204330333300D072604
S1135050002046444220494E43520D072700204673
S1135060442204445435200D07280020464442203O
S113507040564952OD07290020464442204330 33FF
S1135080370D07300020464442204E454D0D073170
S1135090002046442204330343120 4C58490D0707
S11350A032002046444220433034320D073002 07E
S11350804644422043303433 0D07340020464442F2
S11350C020494E43520D07350020464442204445B2
S11350D043520D07360020464442204D5649520D96
S11350E007370020464442204330343 70D07380048
S11350F020464442204E454D0D073900204644428 7
S1135100204441440D07400020464442204330 35AA
S113511037 0D0741002046444220433035330D0709
S1135120420020464442204 94E43520D074300208A
S1135130464442204445435200074400204644421D
S1135140204D5649520D0745002046444220433025
S1135150353 70D0746002046444220 4E454D0D0785
S1135160470020464442204330363 10D0748002092
S1135170464442204330363 20D074900204644421B
S1135180204330363330D0750002046444220494E18
S1135190435200D0751002046444220444543520DDA
S11351A007520020464442204330363 60D07530050
S11351B020464442204330363 70D0754002046 44ED
S11351C042204E454D0D0755002046444220433081
S11351D037310D0756002046444220433037320D04
S11351E007570020464442204330373300D07580008
S11351F020464442204330373 40D075900204644AA
S1135200422043303735 0D0760002046444220 4396
S1135210303736 0D0761002046444220433037378B
S1135220 0D0762002053504320310D0763004330C3
S1135230373420444543204444442050 4F494E547D
S1135240205 44F20434F525245435420524547491F
S1135250535445520D07640049 4E4352204C444177
S1135260412044444420494E4352454D454E542028
S1135270524547495354455253 0D07650020535432
S1135280414120494E435241283 10D076600204CC9
S1135290445820235245475 30D076700494E435253
S11352A04120494E4320302C580D076800494E4395
S11352B052422054504 10D07690020414E44412080
S11352C023244645204348414E474520414C4C2029
S11352D042555420434152525920464C41470D07F0
S11352E07000204C44414220454343 0D0771002087
S11352F0414E444220233120534156452 04F4C44D3
S1135300204341525259204 34F4E444954494F4E31
S113531 00D0772002042455120494E4352430D0768
S1135320730020494E43412020545552 4E204F4E85
S1135330204341525259 0D077400494E435243 0B1
S1135340535441412045434 30D0775002 04A4D50B5
S113535020454E440D0776002053504320310D075D
S1135360770043303735204445432044444 40D07F7
S1135370800444543522 04C4441412044444420 55
S1135380444543524 54D454E542052454749535494
S1135390452530D0779002053544141204445435D
S11353A052412B310D078000204C44582023524594
S11353B047530D0781004445435241204445 3204F
S11353C0302C580D0782002042524120494E43524E
S11353D0420D0783004330373620444543204444 7C
S11353E0440D0784004D564952204C4458204550E2
S11353F043204D4F56452049 4D4D20524547495372
S1135400544552530D078500204C4441422030 2C12
S1135410580D078600204C44582023524547530D0D
S1135420078700204C444141204444440D07880030
S1135430205354414120 4D56495241 2B310D078987
S1135440004D5649524120535441422030 2C580DAE
S1135450079004D56495242204A535220504350 1F
S1135460310D079100204A4D5020454E440D0792BE
S1135470002053504320310D079300433036362028
S113548040445820455043 0D07940020 4C4441425D
S1135490203 02C580D079500204C44582048 0D0707
S1135 4A0960020535441422030 2C580D0797002079
S1135 4B042524120 4D56495242 0D07980020535004
S1135 4C0432031 0D0799004330363420455155208F
S1135 4D02A20494E4352454D454E54204D454D4F8B
S1135 4E052590D080000204C44582048 0D08010072
S1135 4F020494E4320302C580D08020020425241CF
S1135500204 94E4352420D0803002053504320319A
S1135510 0D0804004330363520455155202A2044D7
S1135520454352454D454E54204D454D4F52590D1E
S1135530080500204C44582048 0D08060020444526
S1135540320302C580D0807002042524120494E7B
S1135550435242 0D0808002053504320310D08090F
S1135560004330363120455155202A204C4F4144C8
S1135570204 94D4D20535441434820504F494E54E4
S1135580455 20D0810002044535220414452455888
S1135590430D0811002053545820454D53544B0DCF
S1135 A0081200204A4D50204C5849420D0813005F
S1135 B020535043203 10D0814004330363204C1F
S1135 C044582045 4D53544B0D08150020494E585E
S1135 D00D0816002053545820454D53544B0D08C4
S1135 E01700204A4D5020454E440D0818002053 02
S1135 F050432031 0D0819004330303320 4C445887
S1135600204252454 70D0820002049 4E580D0821DC
S1135610002053545820425245470D082200204A86
S1135620 4D5020454E440D0823002053504320 3153
S1135630 0D0824004330323320 4C44582044 0D08D4
S1135640250020494E580D0826002053545820 4464
S1135650 0D0827002 04A4D5020454E440D082800CF
```

```
S1135660205350432031OD0829004330343320 4C5B
S113567044582048OD0830002049 4E580D0831008B
S113568020535458204 8OD0832002044D502045DC
S11356904E440D0833002053504320310D0834008C
S11356A043303133204C44582042524547OD08358D
S11356B000204445580D0836002053545820425 2C7
S11356C045470D0837002044D5020454E440D08EB
S11356D038002053504320310D0839004330333310
S11356E0204C44582044OD0840002044455 80D08DF
S11356F041002053545820440D0842002044D5084
S113570020454E440D0843002053504320310D08DA
S1135710440043303533204C44582048OD0845009C
S1135720204445580D0846002053545820480D087D
S1135730470020 4A4D5020454E440D0848002053 50
S1135740504320310D0849002053504320310D08A7
S1135750500043303733204C44582045 4D53544B6C
S113576000851002044455 80D0852002053545848
S1135770204 54D53544B0D0853002044D502045AD
S113578044E440D0854004441445220 4C44582023B4
S1135790524547530D0855004441443120 4C44417F
S11357A042203020C58204C4F5720534F5243450D24
S11357B0085600444144322 04C44414120302C5886
S11357C020484947482053 4F55524345 0D0857003B
S11357D020414444412035 2C580D0858002041448 0
S11357E043422034 2C580D0859002050534841 0D91
S11357F00860002050534842 0D0861002054504175
S113580000D0862002041 4E44412023310D086300FD
S1135810204F52414120454343 0D0864002053541 6
S113582041412045434 30D0865002050554C410D2E
S1135830086600205354414120342C580D08670059
S113584020 50554C410D0868002053544141203 5E7
S113585020580D835EC2B8027C025E001C803C00A2
S1135860548412145400 5E00BA401C003EC13E0031
S1135870780456 8C7E10BC189D0054005A009090F9
S113588090F3E352F1F7D433F65BFAD3DC33CCD7F5C
S1135890 95FF87DF815F81BF877F77BF0DBF25BFFE
S11358A005FF275F817F6D7F953FC5EF6F7D073CC7
S11358B0157FD77F157DA37DBB7D2F6EC77FA37F0B
S11358C07C00BC0E7F807C80DE103D901C10F48434
S11358D07C8336027C4456083E843E183C045E14A5
S11358E0BEC27E1E1CD27E9854107E431C025E00F3
S11358F05C045E803C105C04961014848A003100C1
S1135900AC9DDDFF11EE953FC5B5CF3F8D7F853F43
S1135910A9FD91FF956DC1778DFFB11F03FF053F71
S113592028 5D813FABBF0FF7453F911F011F831FC8
S1135930235FC5BFA7BF87F5A53F047707FD0F1DEC
S1135940DE083E9476103C045C243C1C76163E86AD
S11359507C801C125D505446740A78805E065C029A
S1135960 3E401C057DE11F045C023C905E1C361425
S1135970 3481F480DF005C01FD047E8434809C84E7
S1135980877FA7750775871F2B7F0B5F8F7F823DEE
S113599087750B9FD77D477D83FCC53F553FA55F2A
S11359A01CFF857F157FB15F0D3FE17745BF29FF60
S11359B0015F856D873D21EF0375E7BF0D5F0D9F87
S11359C07E945C149E447E441E903C803E947D8173
S11359D07E0C1C147504FC849C841C061E06340274
S11359E03E845E107C52FC009684FF125E087E901A
S11359F0DD101C141C007C045C912C1018007E101B
S1135A00AB7F2F9F0E7F8E7FDB5F475D89FD895BB8
S1135A10E53FAD2FD97DE91DAF75057505FF1D7FE8
S1135A20E57F03750D1F0FFF045D4F0D055F0F3FED
S1135A30077F13BF073FC1FD153FED7D079F8F0D06
S1135A401C94340254017691 5C807C1218007C8092
S1135A50DC005D84B8823E047C1434407E00B8804F
S1135A6014805C001E807E005E001C00BE8036B088
S1135A701C803C1074005E1074045C043998 1C147F
S1135A80FD5507BFBD3F2B7FAF1D8F7D0DF525DD78
S1135A9005B5BD7F81FF053F51BF451D451F003F33
S1135AA0257F097F193D016D013FC1DD04BD41BF63
S1135AB089FFA175AD6D857FA3BF071D0D3D3B5DBE
S1135AC0DEEC3E8136025405481DE9857809FD2AA
S1135AD0FE0076007C82F440BE00CE845D887C9417
S1095AE0D6803E707E003A

S10AA0433901A95B4C728393

S9

.H                                          Actual Sample Execution

.JUMP 5000

:PC = $0000          Set the 8080 PC to
                     start of BASIC

:RG                  Examine the registers

00 3C 1F 37 00 00 00 D4
1FFC CD 0000

:RU                          Now do RUN
OK                           BASIC is running
>LIST

OK                                          enter a simple program
>10 FOR I= 1 TO 5
>20 PRINT *THE EMULATOR IS WORKING*
>30 NEXT I
>40 STOP
>LIST
  10 FOR I= 1 TO 5
  20 PRINT *THE EMULATOR IS WORKING*
  30 NEXT I
  40 STOP

OK                                           RUN it
>RUN
THE EMULATOR IS WORKING
THE EMULATOR IS WORKING
THE EMULATOR IS WORKING
THE EMULATOR IS WORKING
THE EMULATOR IS WORKING

OK
>
OK
>
OK                                          Another sample program
>20 PRINT I
>LIST
  10 FOR I= 1 TO 5
  20 PRINT I
  30 NEXT I
  40 STOP

OK
>RUN
     1
     2
     3
     4
     5

OK
```

A CONSUMER COMPLAINT

Mr. Maury Goldberg
MiniMicroMart
1618 James St.
Syracuse, NY 13203

Dear Mr. Goldberg: *Dated: 78 May 2*

We have received the enclosed request for assistance from one of our readers regarding an order placed with you. Since the complaint seems reasonable enough on the surface, we felt it would be in his interest as well as that of the home computing community as a whole if we inquired on his behalf.

Enclosed you will also find a copy of *Dr. Dobb's* policy statement regarding consumer complaints. May I expect a reply from you within three weeks of the date of this letter? Thank you for your help in making the home computing market a pleasant one for all concerned.

Sincerely,
Tom Williams
Editor

Dear Dr. Dobb's, *Received: 78 April 28*

I'm writing in hopes that you can help me resolve a problem which exists between myself and MiniMicroMart. I had hoped that it wouldn't get this far, but it appears that I've done all I can and it's time for outside help. I'm hoping that you can aid me.

On January 5, 1978, I sent an order for three circuit boards and some documentation for a total of $84.85.

I received a package from MMM within two weeks. However, it was only a partial shipment. They showed one board (their #CMOD 8-2P-100) as backordered, with estimated delivery of "Indefinite approx. 4 weeks maybe the 8-22 bd.". This statement was on their work order/packing list, also showing my invoice #8222, dated January 12, 1978. There was no definite shipping date as required by Federal Trade Commission mail order rules, no offer of refund and no postage paid reply medium, also required by the FTC. Oh well I thought, two out of three isn't bad, and I settled down to wait four weeks for the third board.

Unfortunately, there was another problem with that shipment. I had ordered their MOD-8 system manual that goes with the boards I purchased. I found however that the MOD-8 applies only to 8008 based computers, so I sent it back to MMM, asking that it be exchanged for their 8080 manual (this request, along with the 8008 manual was sent the day after I received the shipment containing the boards and manuals). At the same time, I ordered a copy of their Monitor-80 program listing, enclosing a check for $6.00 to cover the cost of the Monitor and shipping on both the Monitor listing the 8080 manual. The check was cancelled (so they obviously received the MOD-8 manual and Monitor-80 order) but there was no sign of the documentation, much less the board.

After 45 days (no FTC acknowledgement of my order, no backorder notification, no offer of refund, no postage paid return medium) I wrote, asking what had become of my Monitor, Manual and circuit board. No response.

One month later, 12 weeks since my original order, I wrote once again. No response.

Two weeks later, I wrote again, this time sending the letter Certified mail, with a Return Receipt, informing them that if I didn't hear from them, I was going to file a formal mail fraud complaint against them with the FTC. The Return Receipt came back, but nothing else. Apparently they aren't afraid of the Federal Government.

So as it stands, it has been nearly four months since I ordered three circuit boards and three pieces of documentation and all I have is two useless boards and one piece of data, and a large bill for envelopes and postage.

I have contacted the FTC for a copy of their complaint form, but as we know, the wheels of justice turn slowly, so I am turning to you for help, *Dr. Dobb's*.

I know that your world is very busy and that you have your hands full most of the time with the Journal, but I think you would be doing a service, not only for me, but for all computer hobbyists if you would check this out and see what's with this company.

It's bad enough that I've wasted four months on a company with policies like those of MiniMicroMart, but I think that worst part of this entire episode is being totally ignored by a company which takes your money and delivers very little. A company which also apparently views government regulations with disdain.

I should add, as a footnote, that I'm not interested in a refund, I want the circuit board and documentation I paid for.

Thanks for your time,
Robert J. Retelle

2005 Whittaker Rd.
Ypsilanti, MI 48197

Editor's Note:
Mr. Retelle eventually received his documentation but never his third board. Dr. Dobb's *has received no written response from MiniMicroMart.*

On July 26, Dr. Dobb's *contacted Mr. Maury Goldberg by phone. We were informed that the company manufacturing the boards had gone out of business and that he could only offer Mr. Retelle a refund.* – *TRW*

NEW GRAPHICS SOFTWARE FOR SWTPC

News Release *Received: 78 April 20*

GRAPH#1 and GRAPH#2 are utility packages enabling convenient use of the SWTPC GT-61 graphics display with a 6800 computer. GRAPH#1 is for use with a machine language program while GRAPH#2 is for use through SWTPC 8K BASIC version 2. When using GRAPH#2, the user can program the GT-61 display directly from a BASIC program without any machine language programming.

Both versions can display, as well as erase points, lines, and the complete upper-case ASCII character set. Additional capability for user defined characters (such as chess symbols, or even space ships) adds to the ease of use and versatility of GRAPH. Each package is priced at $5.50 (for paper tape), or $6.50 (Kansas City Standard Cassette) postage paid. Package includes either paper tape or cassette, and a 22 page instruction manual with sample programs and a full source listing of GRAPH. Contact: Applied Microcomputer Systems, Box 68, Silver Lake, NH; (603) 367-8004.

THE LAST OF THE FIRST

BY AUBREY DAHL

The Univac II was a marvelous collection of 5,200 vacuum tubes, 18,000 crystal diodes, 184,000 magnetic cores, hundreds of feet of plumbing, and countless miles of wires. It's a wonder that it worked at all; but work it did – for more than 18 years.

The year 1958 was not exactly a banner year in the course of human events, but it did usher in a computing classic when Sperry Rand first presented the Universal Automatic Computer Model II (Univac II) to the world.

It was not only one of the most powerful computers of that time, it was a veritable workhorse, persevering in varied environments on many different applications, and working on and on over the years far beyond the time of its expected obsolescence.

But, inevitably, even reluctantly, the end came.

What is believed to be the last existing functional Univac II in the world finally stopped reading, writing, and computing this March when it was retired after more than 18 years of reliable, sometimes even incredible, service.

The last 13 of those 18 years were devoted to the processing of accounting data for Computer Data Corp. an enterprising Gardena, California service bureau, where the first generation machine was kept alive with third generation software developed by the firm.

Univac II was able to support the accounting needs of an extraordinary number of Computer Data customers, more than 1,000 who probably didn't realize or care that their work was being done on the clanking antique.

Computer Data's president, Thomas Pryor, bought the machine from Pacific Finance in Los Angeles in 1964 when, at a youthful 5 years old, it *already* was obsolete. But the price was right and Pryor had a need and a fondness for the wondrous instrument.

Reprinted With Permission of DATAMATION © magazine.
© Copyright by TECHNICAL PUBLISHING COMPANY,
A Division of DUN-DONNELLEY PUBLISHING CORPORATION,
A DUN & BRADSTREET COMPANY, 1978-all rights reserved.

The cost of the computer was minimal compared to the expense incurred during installation. Much larger in size than an IBM 370/168 and less powerful than a present-day mini (in fact, it is being replaced by one – the Sperry-Univac V76), the Univac II was moved by cutting out a wall in the building that housed it, and carting it across town to Gardena, where a building had finally been located that had a back door large enough to accommodate its unwieldy size. It wouldn't have been much more trouble to disassemble, move, and reassemble a two-bedroom house.

When Sperry Rand first built the Univac II, it was hailed as an impressive achievement in the emerging science of automated data processing. And, indeed, it was, even though only about 32 were ever built.

Think back. 1958 was also the year of the hula hoop.

And it was only a year after the introduction of the Edsel.

The Univac II had much greater speed (160usec add time versus 525) and memory (10,000 12-bit words of core versus 1,000 words of mercury delay line) than its predecessor, the Univac I, which is often acknowledged as the first commercial computer.

Characterized by tube technology, the II boasted more than 5,000 vacuum tubes, or "valves," uncounted miles of wiring, an immense water cooling system, an enormous, tricky console, and a central processor that was 9 feet high, 10 feet wide, and 14 feet long.

A Walk-In CPU

In fact, the cpu was so large that it had a door on it and interior space enough for a person to stand up and move around in it. The computer and its peripherals, which together weighed more than 8 tons, required a room with about 2,000 square feet of floor space. Its water cooling equipment was nearly as big. Input was in the form of ½-inch magnetic metallic foil tape on reels that weighed about 2½ pounds – each – and were, as one might expect, very durable. The foil tape is

no longer made, but the original tapes worked fine right up until the end. When a bad spot appeared on the tape, it was possible to simply punch a hole in it at that point and continue to use it; the tape drives would skip over the hole.

The II was not basically a punched card machine, as were most of its brethren. That magnetic foil tape was the input medium, too. Data entry was done, as it had been on the Univac II, using key-to-tape devices – long before the Mohawk Data Sciences engineers and others put their minds to the matter.

Unlike newer machines, Univac II had a radio speaker that was used to "listen" to its operations when troubleshooting the system. Any deviation in the sound of the noisy computer, and the bug usually was spotted. Inventive programmers took advantage of this feature, in dull moments, and wrote routines to make it play music. What modern day machine would be able to sing "Auld Lang Syne" at its unplugging ceremony as the II did?

According to various users of the old machine, it was the most complicated and overdesigned piece of equipment ever built – and one of the most dependable.

It was in 1958 that the first U.S. earth satellite, the Explorer I, went into orbit and discovered the Van Allen radiation belt circling the globe. Globe? That same year, another space probe, the Vanguard I, determined that the earth is slightly pear-shaped.

And Tim Tam won the Kentucky Derby.

Other users of the Univac II system included the U.S. Navy Electronics Supply Office, the Department of Agriculture Commodity Stabilization Service, U.S. Steel, Metropolitan Life, and Pacific Mutual Life.

These organizations bought their systems new at a price of around $1½ million each. Tom Pryor bought his secondhand, after it had been used for five years by Pacific Finance. He estimated that with knocking out a building wall, moving, and the setting up – which included installing an $80,000 water cooling system – the machine cost him $250,000. That's about $20,000 a year for 13 years service he's enjoyed – dirt cheap.

The "high speed" printer that worked with the computer was actually a Univac I device that was five years older than the Univac II. It was capable of a then-incredible 600 lines per minute on alphanumerics and was good for 800 lpm when pushed a little. It was still capable of this performance at the end.

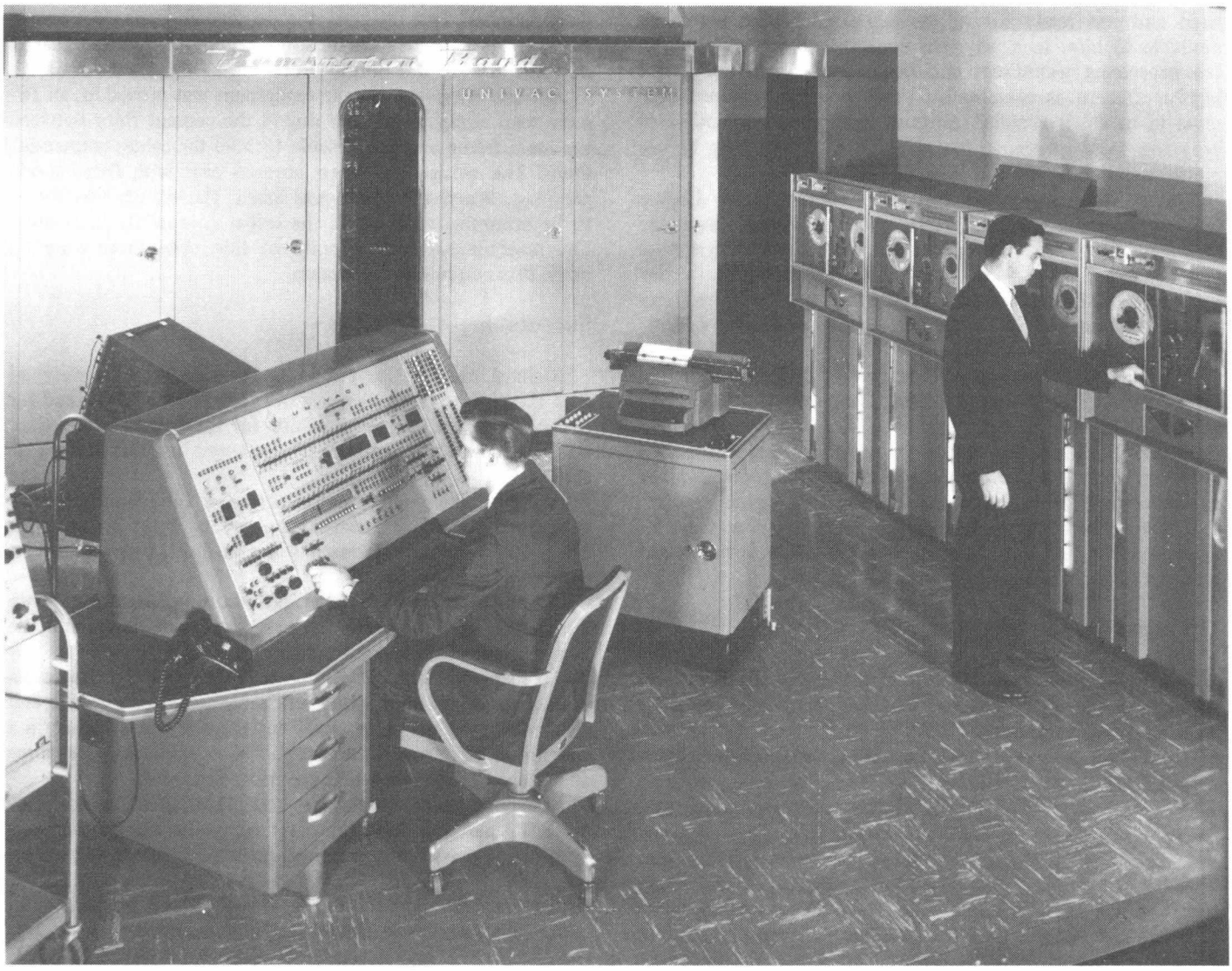


Photo courtesy Sperry Univac Corp.

The Software Lives On

The programming language used with the computer was, appropriately enough, machine language. But the third generation software developed by Computer Data over the years enabled Univac II to perform accounting applications that included payroll, accounts payable and receivable, invoicing, and inventory control in a kind of "pipelined" batch mode.

Computer Data had developed, among other functions, a monitor that allowed each customer's applications to be handled separately, but processed all like operations sequentially in what might be called a "sub-batch."

Although all the accounting programs are integrated with each other, they are structured into standalone modules. The specific requirements of each client are accommodated by the coding of selected options in each module. These options are recorded by entering what is equivalent to a company profile on each firm whose work is to be processed. The programs then perform as if they are custom programs especially designed for that particular company, except that all of the invoicing for all of the clients is done in one pass, then all of the accounts payable, etc.

Because of the modularity, a user could begin with less than the full system and know that when he developed the need and was ready for it, the expanded system would be available to him. In a way, the concept was a cross between tab processing techniques and batch processing. Yet it was highly efficient in that, unlike batch or even multiprogramming in batch, it avoided continual loading and unloading of programs. And of course the machine was built long before multiprogramming was introduced.

The software has been converted to the Sperry-Univac (nee Varian Data Machines) V76, and will be made available to installations using Varian computers in accounting applications – like an old man teaching his grandchildren a few tricks. All of the accounting systems will be offered by Computer Data; they may be offered by Sperry-Univac's Mini-Computer Operation as well.

The U.S. was approaching the end of a decade that had been relatively calm and peaceful, even though controversy continued to flare over the continued development of nuclear weapons. The so-called minorities were asserting themselves in a way that would ultimately result in the publication of a book named "Roots" and a new awareness of human legacies.

The machine was an incredibly reliable worker from its beginning. "It had only two down times of more than eight hours in 12 years of operation," said Pryor.

Through Fire and Flood

But those down times were certainly singular. In one instance, the machine began to function sporadically and thorough troubleshooting failed to turn up the cause of the malfunction (the computer wasn't operating well enough to execute the fault-finding routines).

Then someone thought he smelled smoke coming from the top of the cpu. He climbed a ladder (remember, the cpu is 9 feet tall) and found that there was cable erosion from what he later called "tired insulation" – the cable, which had been about a full inch thick, had frayed away over the years to a 1/4-inch strand. Still the machine had run on. And so the cable was cut and spliced, fresh insulation material was wrapped around it, power was turned back on, and the Univac II again operated with its customary reliability.

Arnold Palmer was the PGA's leading money winner in 1958 with $42,507. Skirts were getting shorter and the nuclear powered submarine USS Nautilus made the first under-ice crossing of the North Pole on its way from Hawaii to Iceland.

Another time at Computer Data the big machine went down due to a plumbing problem. When the door to the mainframe was opened, water gushed out onto the floor.

The computer, Uniservo II tape drives, and power supply all were cooled by a system that pumped water chilled to 50° at a rate of 130 gallons a minute through copper piping. One of the pipes that ran under the floor of the computer had sprung a leak and water was jetting into the air like a small geyser. The first step taken was to cover-up the leak with whatever material was at hand – "keep operating" was the watchword.

Water or no water, only *after* the shift was ended was the extent of the damage surveyed. The floor was soaked and the drainage hole under the floor of the cpu was under water. There was water, water everywhere. An ordinary hand pump was purchased from a nearby builder's supply outlet, hoses were attached, and the pumping proceeded. Fans were turned on and placed to help the moisture evaporate. And in a matter of hours, the job was finished.

Almost. When the Univac equipment was moved in, an 18-inch-deep trough had been dug in the cement floor between the main frame and the console to hold the cables connecting them. The trough had been covered over with fitted wood planking. When the cover was lifted, the trough was found to be brimming with water, the cables completely inundated. The machine had run in spite of this. And there wasn't a single short circuit in the system.

Succumbing to Old Age

Richard Means, Computer Data's engineer who watched over the Univac II for the past 13 years, and who worked for Sperry Rand on a similar machine for three years before that, said that it was a "phenomenal" computer in terms of dependability.

"I have taken components out of it for maintenance or whatever and it just wasn't supposed to keep on working, but somehow, it did. It made some kind of inner adjustment and stayed on. Incredible!"

However, Means added that each morning he came to work during the months before the computer was phased out, he kept his fingers crossed and said a little prayer.

To help maintain his machine, Pryor bought Univac components as spare parts when other machines came to the end of service over the years, and kept them in rented space in a warehouse next to his company. The warehouse has become an archive of first generation computer hardware.

President Eisenhower signed the Presidential Pension Law in 1958, and sent Vice President Nixon on a good-will tour of South America, where he was mobbed in Caracas, Venezuela.

And the Supreme Court reordered school integration in Little Rock, Arkansas. The more things change, etc.

A change in business requirements in the mid-1970's and the continuing growth of the company led Computer Data to the installation of an on-line system that could also continue the firm's time-tested batch operation.

Cost considerations and performance were major factors in the decision to make the conversion to the V76 minicomputer, which is currently priced at about half of what the secondhand Univac II had cost and provides a great deal more capacity, speed, and power.

The V76 minicomputer has two 64K 16-bit memory modules. The mini has semi-conductor memories instead of core, and two 92mb discs. According to Pryor, the continuing drop in the price of discs was another reason for the conversion.

Another cost factor was the electricity bill. It was running about $2,500 a month, with the Univac II accounting for about 80% of it. When the V76 was added during the changeover, Pryor said that he hadn't noticed an increase of even $100 in the monthly bill. Now that the V76 is being used exclusively, the savings are, at the least, substantial.

There were other, more particular, grounds for making the change. Even with the extensive spare parts inventory, maintenance grew increasingly difficult. And qualified technicians understandably became scarce.

Pryor explained that engineers trained in tube technology for computers had become a vanishing breed because tube logic had not been taught to undergraduate engineers in recent years. Thus, as the need for a certain proficiency diminishes, so does the proficiency, itself.

1958 was the first year that airlines carried more transatlantic passengers than ships and the year of the first jet service across the Atlantic, by BOAC. Charles de Gaulle became the President of France, and President Eisenhower sent 5,000 Marines to Lebanon.

And the John Birch Society was founded.

And Althea Gibson repeated as national woman's tennis singles champion at Forest Hills.

Scrap or Souvenirs?

There is, of course, a nostalgic concern that the last functioning Univac II has been consigned to history. When asked why he had used the old workhorse for so long, Pryor answered, "Because I owned it and it worked, and I wanted to wait until minicomputers became usable and could provide the right kind of accounting software support to go with our own system programming."

The ultimate disposition of the venerable computer is still a matter of some conjecture. Pryor thought he might rent warehouse space for it and the spare parts and preserve what is now a relic for future interest, if he finds it economically feasible.

On the other hand, he might, as he said, just save some of it and sell the rest for scrap metal or souvenirs.

Dick Means had some thoughts on that subject. "The Varian operation undoubtedly will bring in new business with its advanced technology, but it wasn't that the demands were too great on the Univac – it was just over the hill."

He smiled. "As for reducing the machine to scrap, you have to remember that it's Mr. Pryor's baby – he could never watch anyone take a hammer to it."

The Pioneer moon rocket was launched in 1958. It didn't reach the moon but it achieved an altitude of 79,000 miles, 30 times as high as any previous man-made object.

The Univac II family set its records as well. Its last member was operating productively 20 years after the series introduction, its meter having recorded 43,219.9 hours of processing, 79,637.1 hours of being powered up. Those marks may never be equalled.

'TURTLE' ROBOT FOR HOME COMPUTERS

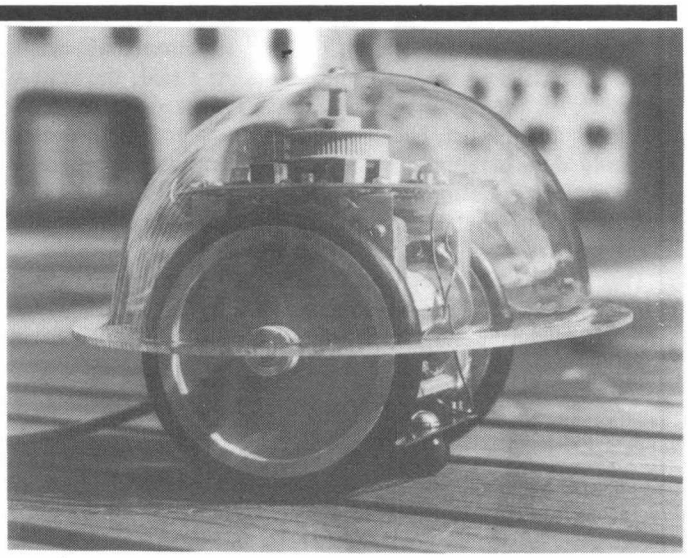
News Release *Received: 78 May 8*

The Turtle is a small home robot invented by William D. Hillis, a senior at MIT. The Terrapin™ Turtle is believed to be the first commercial offering of a home robot ready for sale. It can 'walk,' 'talk,' 'blink,' 'feel,' and draw," says Dan Hillis. Hillis is affiliated with MIT's Artificial Intelligence Laboratory which has used types of "turtles" in various projects.

The Turtle acquired its name because it is approximately the size of a box turtle, consisting of a chassis topped by a hemispherical dome 3½ inches in radius. The Turtle has two large wheels and is capable of moving six inches per second. By turning the wheels in opposite directions, the Turtle can be made to rotate exactly on the spot. Other endearing features of the Turtle include beeping its speaker and flashing its lights.

The Turtle can also "feel" and "draw." The former is accomplished by using its dome/shell as a touch sensor so that the Turtle "knows" when it bumps into an object. To draw, the Turtle lowers a pen from its chassis and moves. With these two capabilities, the Turtle can map rooms by entering them, bumping into the objects within them, and storing the configuration in computer memory. Then the Turtle can let down its pen and draw a rough representation of the floor plans.

The Turtle has lights for eyes and a speaker to emit sounds. It requires a parallel interface: one compatible with an S-100 bus is available as an accessory. Each Turtle comes with ten feet of cable and may be purchased either as a kit or fully assembled. Each kit comes with a tested, 20-page instruction manual.



Brochures are available. The Turtle sells for $300 in kit form and $500 assembled. The interface is $40. Contact David McClees, Terrapin, Inc., 33 Edinborough St. Sixth Floor, Boston, MA 02111; (617) 482-1033.

LISP FOR THE M6800

BY FRITS VAN DER WATEREN
van't Hoffstraat 140
Haarlem, NL 2014 RK
The Netherlands

LISP is an interactive list processing language and very suitable for microprocessors. LISP is very easy to learn and has a very simple syntax. It is very easy to write recursive programs and functions, because the system will stack all returns and intermediate results. When you want to define, for instance, the function for N!, in LISP this would look like:

```
(DEF FAC (N)(COND ((EQ N 0) 1)
                  (T (TIMES (N (FAC (MINUS N 1)))))
          ) )
```

"BASIC minded" people would find this an ugly notation, but in fact it is very well structured.

As you see, we define the function FAC of a variable N. The function-body is a CONDitional expression, where N is tested to be EQual to 0, then the result is 1; else (T) the result is the product of N and the 'FAC' of N − 1.

As you see, we recursively call the function FAC again, where N is assigned to N − 1 and so on. As I said, the system will stack all the intermediate results. This is done on a so-called Association list, because all structures in LISP are lists. Not only data is represented as a list structure, but programs and functions are also lists. This is one of the powers of LISP. And by means of the list notation you can directly see the internal structure of a list.

This interpreter recognizes "inner list" notation only, but it is very easy to write a program, in terms of LISP, which recognizes both "inner-" and "outer lisp" notation. We could write:

```
(DEF LISP2( ) (PROG (A)
 START (TERPRI)
       (TEREAD)
       (PRINT(QUOTE :))
       (PRINT(COND((ATOM(SETQ A (READ)))(APPLY A (READ) NIL))
                  (T (EVAL A NIL)) ))
       (GO START) ))
```

When we now call: (LISP2). We are in the just defined higher level interpreter, which has ":" as a prompt (the input routine READ will still print "*", so we get :* as a prompt now) and the system will now evaluate both:

```
CAR((A B))              which gives A and
(CAR(QUOTE (A B)))      also giving A.
```

In outer lisp notation the system does the qouting for you.

For completeness I shall give the definition of DEF:

```
(PUTPROP (QUOTE DEF)
         (QUOTE (LAMBDA (L A)
          (PUTPROP (CAR L) (CONS LAMBDA  (CDR L)) EXPR )))
         FEXPR)
```

We can now define a function by:

(DEF function-name (list of parameters) (function body))

The basic LISP interpreter consists of a READ. EVALuate and PRINT cycle, and will therefore directly execute the expression, the user has typed, and print the result. When you type, for instance:

(TIMES −3 4)

The system directly responds with: −12.

It is not my intention to give a complete description of the language, because a good description is too long for just an article. But there are some good books about LISP (see References).

This implementation of LISP is a version which is called in the literature LISP 1.5. That is standard LISP, or LISP 1, with two extensions called the PROG-feature and the FUNARG-mechanism.

Features of This Implementation

- The interpreter uses 4.5K bytes, including the initial OBLIST, and runs very well in a system with 8K bytes.
- Non compacting garbage collector.
- Advanced I/O capability.
  Implemented handlers are:
  TTY handler with an ACIA.
  READER handler with a PIA (A).
  PUNCH handler with a PIA (B).
  And it is very easy to extend this by patching the device table with an address to user written handlers.
- Very free input format.
- Easy error recovery by means of BACKSPACE, DELETE and CANCEL.
- Automatic storage allocation.
- Extended error messages.
- The interpreter recognizes 'inner lisp' notation.

Brief Function Description

Elementary functions

(CAR X)	value is the first element of list X.
(CDR X)	value is the remainder of list X.
(CONS X Y)	value is the list: (X . Y)
(QUOTE X)	value is X literally
(RPLACA X Y)	replace the 'CAR-part' of X by Y; value is X.
(RPLACD X Y)	replace the 'CDR-part' of X by Y; value is X

I/O functions

(READCH DEV)	read one single character from 'DEV'; value is this character.
(READ1 DEV)	read an atom; value is this atom.
(READ DEV)	read an S-expression; value is this expression.
(TEREAD)	flush the input buffer; value is NIL.
(PRIN1 DEV)	print atom X; value is NIL.
(PRINT X DEV)	print S-expression X; value is NIL.
(TERPRI DEV)	print a CR and LF on DEV; value is NIL.
(OPEN DEV NAME)	open a file on DEV with NAME as file name.
(CLOSE DEV)	close file on DEV.

Predicates

(ATOM X)	if X is an atom, value is T else NIL.
(NUMBER X)	if X is a number, value is T else NIL.
(NULL X)	if X is NIL, value is T else NIL.
(EQ X Y)	if X is the same as Y, value is T else NIL.
(GREATER X Y)	if X is greater than Y, value is T else NIL.
(COND (A B) (X Y))	If A is not NIL then B is evaluated; else if X is not NIL then Y is evaluated; else NIL.
(LIST A B C. . . Z)	The result is a list of all its evaluated arguments. The number of arguments may be infinite.
(EVAL X Y)	Evaluate X with Y as association-list. In fact this is the LISP-interpreter itself.
(APPLY X Y Z)	Apply the argument Y to the function with Z as initial association-list.
(PROG (A B. . . .) L (statement 1) (statement 2))	With this function we are capable of writing programs in LISP. The first argument of PROG is a list of variables used inside the PROG, and the remainder is a list of labels and statements.
(GO L)	Goto label L (literally)
(RETURN X)	Return from a PROG with the value of X.
(FUNCTION X)	This is a quotation for functions.
(ALIST)	return the current systems association-list.

Arithmetic functions

(PLUS X Y)	value is X + Y
(MINUS X Y)	value is X − Y
(TIMES X Y)	value is X * Y
(QUOTIENT X Y)	value is X/Y

Some miscellaneous functions

(SETQ X Y)	X is set to the value of Y; this is also the value of this function.
(PUTPROP X Y Z)	The property-list of atom X is extended by property Y under an indicator Z. If the indicator already exists, then its property is altered; value is Y.
(GET X Y)	Get the property saved under the indicator Y from the property-list of atom X.
(SASSOC X Y)	Lookup X on the association-list Y, when found return its value; else NIL.

How to Obtain This Implementation

Send an International Money Order to:

F. v. d. Wateren
van't Hoffstraat 140
Haarlem, NL 2014 RK
The Netherlands.

Costs:

- $10. – for an object papertape in S1-FORMAT complete documentation, and a listing of the I/O package of LISP.
- $25. – for an assembler listing with symbol table and the macro generation for the OBLIST (in GPM)
- $30. – for the above mentioned two items.
- $12.50 for cassette tape, Kansas City standard, 300 baud, including description and I/O listing.
- $32.50 for cassette and complete listing.

References

1. Berkeley, Edmund C., and Daniel B. Bobrow, *The Programming Language LISP: Its Operation and Applications*, MIT Press: Cambridge, MA, 1966.
2. Friedman, D.P., *The Little LISPer*, Science Research Associates: Palo Alto, CA, 1974.
3. Siklosky, Laurent, *Let's Talk LISP*, Prentice-Hall: Englewood Cliffs, NJ, 1976.
4. van der Poel, W.L., *The Programming Languages LISP and TRAC*, T.H. Delft: The Netherlands, 1972.
5. Waite, W.M., *Implementing Software for Nonnumeric Applications*, Prentice-Hall: Englewood Cliffs, NJ, 1972.
6. Weissman, Clark, *LISP 1.5 Primer*, Dickenson: Belmont, CA, 1976.

FORTH DUMP PROGRAMS

BY JOHN S. JAMES
1090 Miller Ave.
Berkeley, CA 94708

Here are a couple short examples to illustrate programming in Forth.

Forth is a stack-oriented programming language which is also its own operating system. Typically the complete Forth system fits in only 6K bytes of code, which includes the Forth compiler, a run-time system including I/O drivers and virtual memory, a text editor, plus an assembler, which is (optionally) used for writing new I/O drivers or time-critical code which must run at full machine speed. Forth object code is very compact and the system is interactive, yet run-time speeds can be as little as 30 percent slower than machine code (100 percent on 8-bit micros). Typical hardware is 16K bytes and a CRT; for serious development work, a floppy or other random access mass storage for source programs is necessary. For more introductory info, see my article in the May, 1978 issue of *Dr. Dobb's*.

The Forth system is like a root of a language which grows toward any application area for which it is used. Forth programs consist of short (usually one to five line) definitions of new operations, which then become part of the language. Each large application program is really a special-purpose language, greatly facilitating modification. Because object code is so compact, any general purpose operations (such as the examples below) can be added permanently to the language. Then they are automatically available to be used from the terminal as commands, or as instructions in other programs; and they are used in exactly the same way in either case.

Here is a Forth program to dump a specified number of bytes from a given memory address:

```
: DUMP 3 F =     0 DO DUP I + \@ . LOOP DROP ;
```

Examples of its use, in decimal and octal:

```
8100 10 DUMP
 32   0 114  31 214   6  18  11 140   6
8100 10 OCTAL DUMP
 40   0 162  37 326   6  22  13 214   6
```

Most Forth systems can use any number base, even above 16, in which case letters after 'F' in the alphabet are taken as digits. These number bases affect only the I/O; internal computations are binary, so there is no loss of efficiency.

Forth operation names are separated by one or more spaces. Names can be any length, and can include any combination of letters, numbers, special characters, or even invisible control characters. At compile time, the system first searches the *dictionary* of operation names. If the character string is not found, it tries to make a number out of it, using the number base which is currently in effect; if this fails an error message is given. Almost all of Forth, the system itself and the user's applications, is part of the dictionary.

In the example above, the first character string is ':' (a colon), which is the DEFINE operation. It sets up the beginning of a dictionary entry for the next word, 'DUMP'. The definition is terminated by the semicolon.

'3 F =' sets 'F', which is the output field width, to three, so that each number which is output will take at least three spaces (plus an additional space between numbers). This is done so that the output will line up well. Otherwise each number would take as many spaces as it happened to require.

At execution time, the number three is pushed onto the stack, then 'F' is executed. 'F' is a variable of type INTEGER, declared by the system and used as field width by the output routine. Forth users can declare their own variables of type INTEGER, of other system-defined types such as CONSTANT or ARRAY, or they can create their own user-defined variable types.

Any variable of type INTEGER is really a Forth operation which pushes the address of a location suitable for storing an integer onto the stack. After 'F', '=' is executed; it places the second number on the stack into the address on top of the stack; and then, like other Forth operations, it destroys its arguments on the stack.

'DO' and 'LOOP' are structured looping operations. 'DO' takes an initial and a termination value from the stack (the loop actually ends one before the termination value). In this case, when '8100 10 DUMP' is executed, the stack contains '8100' and '10' before DUMP starts to execute. ' 3 F =' makes no net change to the stack, but '0' is pushed on before DO is

executed, so DO sets up a loop of ten iterations (index values zero through nine). 'DO' destroys its two arguments on the stack, which then contains only '8100'.

Now 'DUP' makes a copy of what's on top of the stack, making the stack be:

8100
8100

'I' is an operation which pushes the value of the index of the innermost 'DO' loop onto the stack. '+' adds the top stack values, replacing them with their sum. On the first iteration, zero was added, leaving the '8100' unchanged.

'\@', or BYTE FETCH, replaces the address on the top stack value with the byte which was at that address. '.', or PRINT, prints the value on top of the stack, and then discards it. Each time around, the '8100' sitting on the stack is copied, the loop index is added to it, and the byte at that address is printed.

After the loop terminates, 'DROP' is executed. It throws away the top stack value. This is to get rid of the '8100', so that DUMP does not leave garbage on the stack after it has finished.

Incidentally, this program could also be written (a little better) as:

```
: DUMP 3 F =     OVER + SWAP DO I \@ . LOOP ;
```

'OVER' takes the second argument on the stack and (without destorying it) pushes a copy of it on to the stack. After the '+' is executed, the stack contains:

8110
8100

(assuming that '8100 10 DUMP' was executed). The rest of the working of this program is left to the reader as an exercise (SWAP reversed the top two items on the stack).

Here is a dump program which formats every 100 bytes (if in decimal) into a separate block of print-out, with the starting address printed on a separate line. (In octal, every 64 bytes is in a separate block, and similarly for other number bases.)

```
: BYTEDUMP 3 F =     OVER + SWAP DO
   I BASE @ DUP * MOD 0= IF CR CR I . CR THEN
   I \@ . LOOP ;
```

```
8000 140 OCTAL BYTEDUMP

17500
234   4 140   6  12   0  64  25 262  14 110  25 246  36 140   6  24   0 100  36
 66  15 140   6  62   0 100  36  66  15 140   6 106   0 230  36 166  14 200   4
166  21 146  21 156  24 276  36 214   6  12 102 117 130 104   0  32  37 326   6
136  14 266  27

17600
 42  37 356  27 200  37 214   6   6 107 105 122 115   0 142  32 326   6 140   6
 36   0  64  13 140   6  37   0  64  13 214   6   3 102 131 105  40   0 162  37
326   6  22  13 214   6   4 104 125 115 120   0 376  33 326   6 202  14  44  31
236  21 136  14

17700
266  27 234   4 100  30 200   4 210  21  66  32
```

This program is similar to the example above, except for the added second line. 'I' puts the loop index on the stack, and 'BASE @ DUP *' duplicates the value of the number base currently in effect and multiplies, leaving the square of the current base value on the stack, with the loop index value below it. ('BASE', like 'F', is an INTEGER type variable defined by the system, and used by the '.' output routine.) Then MOD replaces those two numbers by their modulus, which is zero if the loop index (which is the next address to be printed from) is evenly divisible by the square of the current number base; otherwise it is some non-zero value.

This is done because in Forth, structured conditional instructions take zero as truth-value FALSE; any non-zero value is TRUE. '0=' tests the top of the stack for zero, replacing it by '1' if it was zero, by zero otherwise; i.e., '0=' reverses the truth value of the number on top of the stack (some Forth systems call this operation 'NOT').

Now 'IF' tests the top of the stack; if it is true (non-zero), code following the 'IF' will be executed. (If false, code after an ELSE would be executed, but the 'ELSE' clause is optional and not used here.) In either case, execution continues after 'THEN'.

In this example, if the address of the next byte to be printed is evenly divisible by the square of the base, the modulus is zero, which 'IF' would interpret as false. '0=' reverses the truth value, and the code 'CR CR I . CR' is executed in that case. This code does two carriage returns, prints the address (the value of 'I'), and does another carriage return.

This program in object form takes 64 bytes; it can be left as a standard part of the system or, if memory is very tight, it could be loaded when needed. It can be used from the terminal as a command, or as part of another program. 'BYTE-DUMP' doesn't care how its arguments get on the stack, so they don't have to be literals; the full power of Forth is available to compute them.

This program is written in the Caltech version of Forth, which is distributed by DECUS for PDP-11. Other Forth versions use different names for a few of the operations, e.g. the equal sign may be equality test, with an exclamation mark used for the STORE operation, where these examples use qual sign. Usually software is fairly easily transportable even when the Forth versions have not been designed to be compatible, using redefinitions such as:

```
: = ! ;
```

But Forth versions on different machines can easily be identical, so that software written on one machine can be transported to others without change. Currently the Forth Interest Group (787 Old County Rd., San Carlos, Ca 94070) is developing identical implementations for several different micros and minis, and the code will be in the public domain.

This paper hasn't mentioned other features of Forth, e.g. nesting of loops and conditional, recursive routines, easy implementation of multitasking, tree-structured vocabularies, or modular debugging. Forth is especially suited to real-time control applications, to large projects, and to small machines. Software development times are often a fraction of what would otherwise be required. Today, access to Forth can be hard to find (e.g. to my knowledge, the system used for these examples is the only one currently running in Berkeley). We expect increasing use of this language as it becomes more available.

NOTE: For more information, see the newsletter *Forth Dimensions*, published by the Forth Interest Group, 787 Old County Rd., San Carlos, CA 94070. Price is $5 subscription, $1 single copy.

STRUBAL VS FORTH

Gentlepeople: *Received 78 July 5*

This letter was inspired by John James' article on FORTH in your latest magazine. (No. 25, page 21) I was very impressed with the ingenuity of this system. It seemed very cleverly designed to get so much capability in such a small space. Unfortunately, I think that FORTH fails one of the primary tests of a good programming language: readability. I feel that a good language is designed to make programs easy to read and understand. Easily read programs are easily debugged programs. I was hard pressed to figure out that:

```
: POLY DUP DUP * 5 * SWAP 7 * -10 -;
```

was the evaluation of the polynomial $5X^2-7X-10$. I think that a language should express such a statement more like the following form:

```
PRINT 5*X*X-(7*X)-10
```

which is much easier to relate to the task at hand. To understand the FORTH construction I have to think about stacks, etc. and the internal representation of data. High-level languages should insulate the programmer from such considerations as much as possible.

The illustration on page 24 is another example of the lack of readability of FORTH programs. I doubt that any programmer could quickly determine what this program does. I have seen programs for the LIFE game in many languages, but the FORTH example was by far the most obscure. No doubt it is easier to use than assembly, but there has to be something more readable. I think a well commented assembler program would make more sense to the average reader that the FORTH example!

I enclose an example of how I feel the example program might be written. The language of my example is STRUBAL, a language that J. Hemenway and I devised. A STRUBAL eample appeared in an earlier *Dr. Dobbs'* (no. 23). STRUBAL exists as a full-compiler for 6800 systems. It runs in a minimum of 16K bytes of RAM and requires mass storage. The example compiled into 1888 bytes of object code, which is considerably bigger than the FORTH code of 475 bytes. However, the run-time package required by the STRUBAL code was 3.9K bytes as opposed to the FORTH interpreter size of 5 to 6K bytes. The STRUBAL assembly code file could be readily optimized by a programmer knowledgeable in 6800 assembly language to remove 200 to 300 bytes. I feel that program size (within reason) is not so valuable a measure of language quality as readability, but STRUBAL and FORTH seem to produce roughly equivalent load modules when all memory-resident parts are accounted for. Since the STRUBAL code is directly executed, it will be much faster than the FORTH code, which must undergo some interpretation. True, STRUBAL requires the overhead of multiple passes through the source program, and it needs intermediate files, but the end result is a program that both works and can be read by someone else. The example program took me about 20 minutes to write and key in, about 8 minutes to compile and run, and it ran the first time.

The example maintains two "universes", each an array with 25 rows and 81 columns. A "generation" is defined as the conversion from the present universe to a new universe. Lines 6-12 set up a random first universe. On average, half the elements of the universe (called "cells) will have a star ('*') in them (called "alive"). Lines 13 through 18 copy the new unvierse back into the old and print out the universe as the latest generation. Line 16 uses the STRUBAL string capability to copy an entire character string at once. Lines 20-46 create the new generation. Lines 23-28 count the neighbors of a given cell. As in the FORTH example, the cell itself is counted. The STRUBAL keywod "XTRACT"removes a number of characters from a given string. Lines 30-38 determine the state of a cell in the new universe. Note that a cell counts itself as a neighbor.

The STRUBAL example is rather "self-explanatory" . . . and I feel that this is major strength. The STRUBAL compiler is available from:

Hemenway Associates
151 Tremont Street
Boston, Mass. 02111

on Smoke-Signals, ICOM, SWTP Flex, and PERCOM disks. A source-book which gives the entire source listing and comprehensive algorithm descriptions of the compiler will be available soon. Hemenway Associates markets an extensive line of 6800 software, including a macroassembler, editor, linking loader, and cross-assemblers for the 6502, 8080 and F-8.

I feel that software writers should strive for readability in their work. What good is a clever algorithm design if no one can find it among the verbiage?

Sincerely,
Robert D. Grappel

148 Wood Street
Lexington, MA 02173

```
0001.          * PROGRAM TO PLAY 'LIFE'
0002.          * INITIAL GENERATION RANDOM
0003.          * 80 BY 24 MATRIX TO FIT CRT SCREEN
0004.          * 'INSPIRED' BY FORTH VERSION IN DR. DOBBS #25 PG. 24
0005.          *
0006.                DSTRING B1(81,25),B2(81,25)
0007.                FOR I=1 TO 25
0008.                FOR J=1 TO 81
0009.                STRING B2(J,I)=' '
0010.                IF .RAND(I) .LT. 0 STRING B2(J,I)='*'
0011.                NEXT J
0012.                NEXT I
0013.          *
0014.          LOOP  PRINT ///,[79],'NEW GENERATION'
0015.                FOR I=2 TO 24
0016.                STRING B1(1,I)=B2(1,I)
0017.                PRINT /,%B1(2,I)
0018.                NEXT I
0019.          *
0020.                FOR I1=2 TO 24
0021.                FOR J1=2 TO 80
0022.                NBRS=0
0023.                FOR I2=I1-1 TO I1+1
0024.                FOR J2=J1-1 TO J1+1
0025.                XTRACT TEST=1,B1(J2,I2)
0026.                STRING IF TEST .EQ. '*' LET NBRS=NBRS+1
0027.                NEXT J2
0028.                NEXT I2
0029.          *
0030.                XTRACT TEST=1,B1(J1,I1)
0031.          * CELL WITH 2 OR 3 NEIGHBORS SURVIVES
0032.          * CELL WITH <2 OR >3 NEIGHBORS DIES
0033.          * EMPTY WITH 3 NEIGHBORS CREATES NEW CELL
0034.                STRING IF TEST .EQ. ' ' THEN EMPTY
0035.                IF NBRS .LT. 2 THEN DIES
0036.                IF NBRS .GT. 4 THEN DIES ELSE SURVIV
0037.          *
0038.          EMPTY IF NBRS .NE. 3 THEN DIES ELSE SURVIV
0039.          *
0040.          DIES  STRING B2(J1,I1)=' '
0041.                GOTO ELOOP
0042.          *
0043.          SURVIV STRING B2(J1,I1)='*'
0044.          *
0045.          ELOOP NEXT J1
0046.                NEXT I1
0047.          *
0048.                INPUT /,'CONTINUE (Y OR N)?',%TEST
0049.                STRING IF TEST .EQ. 'Y' THEN LOOP
0050.                STOP
0051.                END
```

USE OF NOPCODES AS EXECUTABLE LABELS

Received: 78 April 7

Every microprocessor has one opcode defined as a "no-operation" (NOP), useful as a filler in program gaps or timing loops. In addition, some micros have opcodes (defined or undefined, the latter "illegal" and so not necessarily identical in chips from different mask sets) that are executed without causing any detectable change anywhere. These can be called *nopcodes*. E.g., the 8080 and Z80 have 7 instructions ($7F, $40, $49, $52, $5B, $64, and $6D) that move the content of a register into the *same* register, and so are *defined nopcodes*. Although they must now be the least-often used opcodes in the 8080 set, they *could* be used as *executable labels for the next instruction*. Such labels, although meaningless to the CPU and with no effect on operation (except timing), could be *assigned meanings* interpretable by debuggers, relocaters and other programs. The information content of the 7 "monops" (short for one-byte nopcodes) is limited, but could be increased by using sequences of 2 ("monop-duos") or 3 ("monop-trios"). For example, the 8080 could have 49 monop-duos and 343 trios. Since interpreting logic could easily recognize solos, duos, and trios, this would allow 399 different labels. If the 8080 NOP (00) were allowed as the second and/or third element, the total would become 511.

Although the 650X has no defined nopcodes, it now has a large supply of undefined ones. (This of course will change in the enhanced versions now in the design stage, but there are thousands of systems now using the original design.) It has 6 monops ($1A, $3A, $5A, $7A, $DA, and $FA). It also has 5 *two-byte* illegals ($80, $82, $89, $C2, and $E2) that require an "operand" byte but make no use of it. I think of these as "binopcodes" (nopcode plus one "noperand") or "binops". Since the noperand can be any one of the 256 possible 8-bit bytes, the information content of binops is very large (and they would be simpler to interpret than monop-duos or -trios). There are also "trinops" (e.g., the 8 illegals of type X_1C).

Before going on, I should reply to the letter by Paul Schick (*DDJ* #21), that is a strongly negative reaction to my article on 650X opcodes (*DDJ* #17). Neither in that article nor in my note in *BYTE* (*2* (12): 72, 1977) did I advocate the use of illegal opcodes *as instructions*. In fact, the whole point of my legality-testing subroutines OPLEGL (*DDJ* #17 and #19) and HYLEGL (*DDJ* #22) was to *screen out* illegals. I presume that the 650X designers know very well what their chip does with illegal opcodes (although they see no need to document it!). I respect their decision *not* to support even the illegal operations that might be somewhat useful. However, designers are not endowed with divine omniscience and infallibility. They can—knowingly—do something foolish, or—unkowingly—do something wise. The point I am trying to make in *this* note is that *some* nopcodes may be so useful as programming aids that their retention as defined elements in an instruction set merits serious consideration by designers. I am presently exploring the implementation of 650X binop-labels, but this will require time and thought, so I am publishing the *concept* in order to make it immediately available to other programmers (and to designers).

In thinking over possibilities for scanning-debugging (of which my program SIMBUG in *DDJ* #19 is a very primitive example), it became increasingly clear that the superiority of assembly langugae over machine language largely rests on its much higher information content, especially in labels. This makes assembly-language programs much longer, somewhat more intelligible to human beings, and quite unintelligible to the microprocessor. A fairly complex assembler program is needed to translate them into machine-intelligible form. The assembler also acts as a scanning-debugger and as a program locating and addressing controller. Labels play the major role in these useful activities with the crucial symbol-table at the core of everything.

In the KIM-1 program SWEETS by Dan Fylstra (*BYTE 3* (2): 62, 1978), there is an ingenious use of machine-language labels. However, a Fylstra-labeled program is not executable. Labels must be removed before execution, and if there are execution bugs they may need to be re-inserted for the program revision. This limitation could be removed by using binop-labels. In fact, some problems of machine-code relocation might be soluble by retaining at least some nop-labels in the taped or disked programs, which could then be address-corrected by a relatively simple program.

A special kind of labeling is also possible with *legal* opcodes. E.g., a STORE instruction addressing a location in ROM is in effect a NOP, so that both the ROM page and the location on that page can convey information. Also, *any* instruction that is *unreachable* in program operation could be used to store non-program information (a trick used in my subroutine BYTNUM, in *DDJ* #17, 1977. However, the detection of and information-retrieval from such labels would be less convenient than with monop- or binop-labels. A ROM-trinop ought to use a STORE opcode normally rarely used in working programs, to make detection easier (the *page* of every such opcode would have to be checked to distinguish between ROM and RAM), but would still involve more bytes and execution time.

The conservative view of a microcomputer system is that it should differ from large systems primarily in scale, in the way that a VW differs from a Mercedes, and that there ought not to be *qualitative* differences in hardware, software, or methods of operation. Innovations should arise in the giant systems, and then (to the extent that the micropocketbooks of micro users allow) be copied at the lower levels. I prefer another analogy. There are no elephants that burrow underground like moles or can fly like bats, and downsizing to the level of insects allows even greater diversification. Micro systems are going to be individualized and customized in ways that would horrify IBM, and some innovations at the micro level (most will of course be failures) may well migrate *upward*.

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

MAKING A NEW case FOR THE HEATH H9 TERMINAL

BY HOWARD L. NURSE
665 Maybell Avenue
Palo Alto, California 94306

Copyright, 1978, Howard L. Nurse.

If one reason you invested in a computer is to take advantage of word processing programs, a display restricted to upper case is as useful to you as a one octave piano is to a composer. Here is a simple method for converting the Health H-9 video terminal to display lower case alphabetic characters.

After studying the circuitry, I found it would be easy to modify the H-9 to include the missing characters. The keyboard on the H-9 would be difficult to convert, but Heath has provided a convenient parallel I/O connector on the rear panel of the terminal, through which an external keyboard having upper and lower case can communicate.

This article describes how you can add the integrated circuits and keyboard to an H-9 to make it a full upper/lower case terminal.

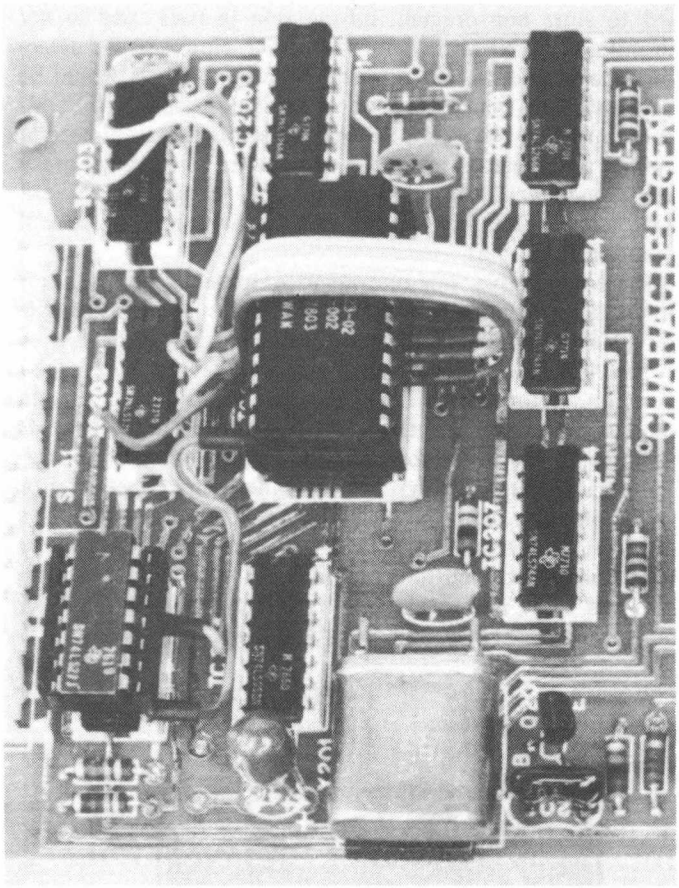


Fig. 1
Detail of H-9 Character Generator board showing lower case modification

Reprinted with permission from REMark Magazine, Journal of Heath User's Group.

Circuit Modifications

A circuit diagram and parts list for the changes to the H-9 Character Generator Circuit Board are shown in figure 2. Two integrated circuits, an RO-3-2513-/CGR-002 from General Instruments and an SN74LS27, must be added to generate the new characters.

The lower case memory, an RO-3-2513/CGR-002, has a surprise in store for the unwary. For some reason (probably perfectly logical) the addresses on the input of the CGR-002 chip are inverted with respect to its upper case twin. The H9 modification would have been simplified had the address formats of the two character generator chips corresponded. The General Instrument catalog lists a version of the 2513, the CGR-005, which has correct address levels, but this device is not readily available through mail order outlets. Fortunately, the required inverted addresses are available on the inverted Q outputs from IC202 and IC203. The outputs from the character generator chips are directly compatible.

The outputs from the character generator chips can be made active or high impedance, depending on the level of the enable inputs on pin 11 of each chip. When pin 11 is low, the outputs of the corresponding chip are active.

IC223, an SN74LS27 Triple NOR Gate, has been added to the character generator board to select which chip will be active. When bits 6 and 7 of the incoming ASCII character are high, IC223 recognizes that a lower case character is present and enables the lower case character generator chip. At all other times the upper case chip is enabled by the same IC.

Wiring the Two Integrated Circuits

The two new integrated circuits are piggy-backed onto host IC's on the Character Generator board. A photograph of a modified board is shown in figure 1. I found the easiest way to add the devices was to mount a socket on the original chip, and then to mount the new IC in the socket. A total of 7 external connections must be made to the character gene-

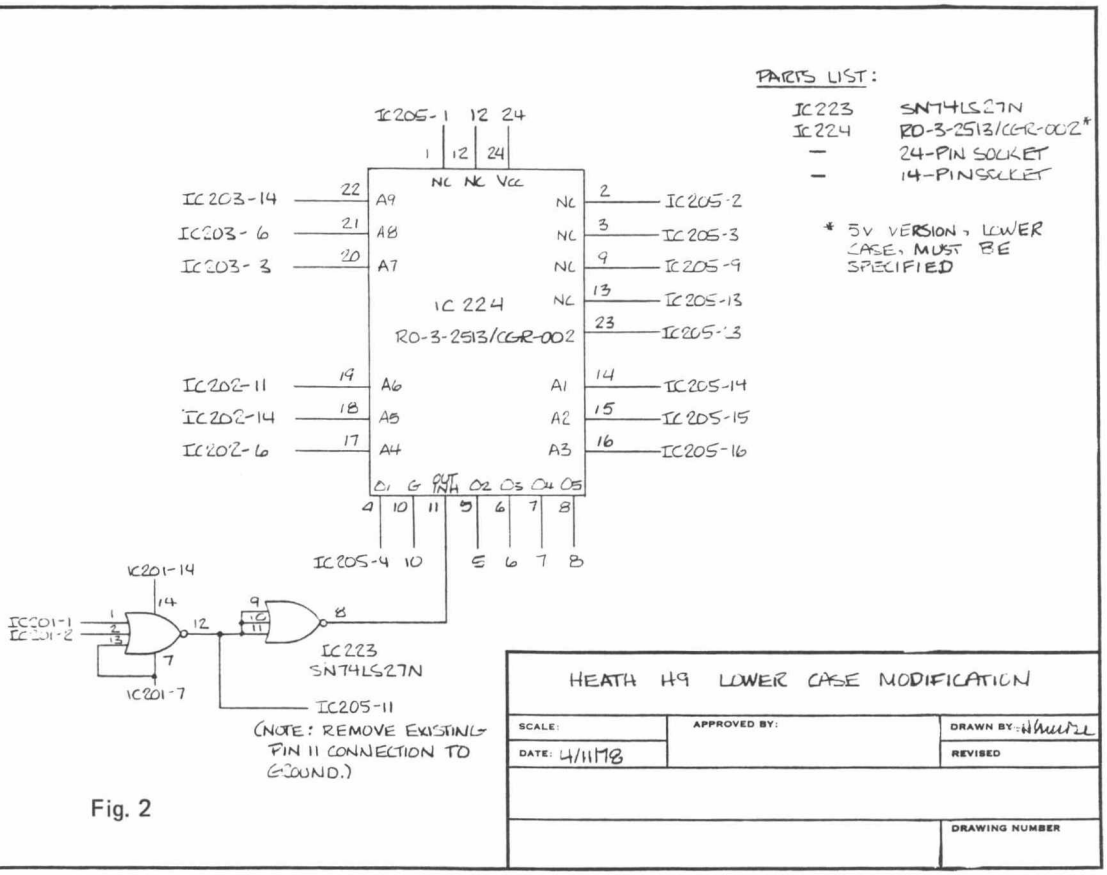

PARTS LIST:
IC223 SN74LS27N
IC224 RO-3-2513/CGR-002*
— 24-PIN SOCKET
— 14-PIN SOCKET
* 5V VERSION, LOWER CASE, MUST BE SPECIFIED
IC205-1 12 24
NC NC Vcc
IC203-14 22 A9
IC203-6 21 A8
IC203-3 20 A7
IC224
RO-3-2513/CGR-002
IC202-11 19 A6
IC202-14 18 A5
IC202-6 17 A4
NC 2 IC205-2
NC 3 IC205-3
NC 9 IC205-9
NC 13 IC205-13
23 IC205-23
A1 14 IC205-14
A2 15 IC205-15
A3 16 IC205-16
O1 G OUT PNA O2 O3 O4 O5
4 10 11 5 6 7 8
IC205-4 10 5 6 7 8
IC201-14
IC201-1
IC201-2
IC201-7
IC223
SN74LS27N
IC205-11
(NOTE: REMOVE EXISTING PIN 11 CONNECTION TO GROUND.)
HEATH H9 LOWER CASE MODIFICATION
SCALE:
APPROVED BY:
DRAWN BY:
DATE: 4/11/78
REVISED
DRAWING NUMBER

Fig. 2

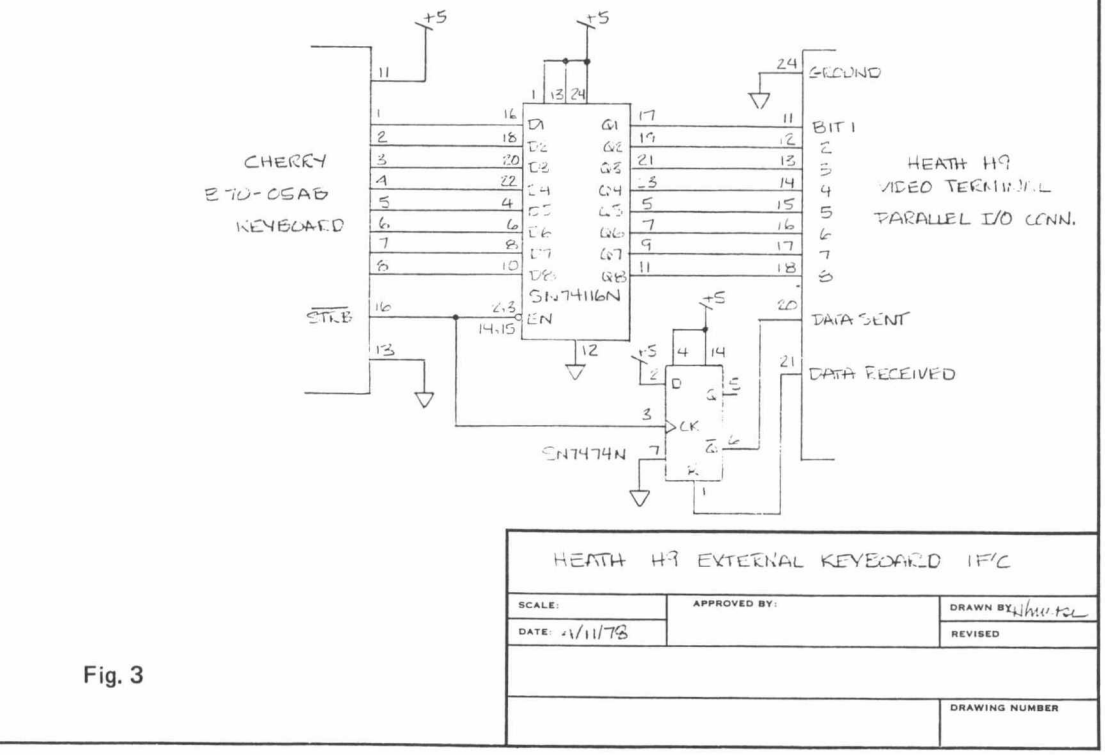

+5
+5
24 GROUND
CHERRY
E-70-0SAB
KEYBOARD
STRB
SN74116N
BIT 1
HEATH H9
VIDEO TERMINAL
PARALLEL I/O CONN.
DATA SENT
DATA RECEIVED
SN7474N
HEATH H9 EXTERNAL KEYBOARD I/F/C
SCALE:
APPROVED BY:
DRAWN BY:
DATE: 4/11/78
REVISED
DRAWING NUMBER

Fig. 3

rator chip through pins on the socket which have been bent out at right angles from the socket.

Once the socket pins have been bent, place the socket over the existing chip (which you should remove from the board while performing the surgery) and solder the remaining pins to the corresponding pins on the IC. Note that the chip enable, pin 11, is also bent out from the original upper case chip. This is the only pin on an IC which must be bent as part of the modification.

IC223 is attached to IC201 in the same manner. For this IC, however, only pins 1,2,7 and 14 are connected to the host chip. All other pins on the socket must be bent so they do not touch the host chip below.

The wiring of the two integrated circuits is completed by connecting the six address lines to IC202 and IC203, and the two chip enable inputs to IC223 as shown in the schematic.

Keyboard Interface

A variety of keyboards which provide lower case alphabetic characters is available at reasonable cost. Look for one which has a parallel output port and provisions for locking the alphabetic characters in upper case during the times when you don't want lower case. I chose the new Cherry "PRO" which offers much flexibility in addition to meeting the minimum requirements.

The Cherry "PRO" has five user-defined keys which you can wire to provide any output character you desire. I am using four of them to provide Control-I (TAB), Control-S (Stop), Control-Q (Go), and Control-C. By adding an SN74123N dual monostable multivabrator IC the keyboard has a "Typamatic" mode which repeats all keys if they are held down for more than a half second.

A schematic for the wiring between the keyboard and the terminal is shown in figure 3. The data transfer handshaking requires that the parallel data from the keyboard be present from the keyboard strobe until the terminal has sent back a signal saying that the data has been accepted. This handshaking takes two additional IC's on the output of the keyboard to latch the data.

Software

Heath software will support lower case characters. The Text Editor (Ted-8) and Extended BASIC are the two programs which can make the most use of the new capabilities of your terminal. Ted-8 is easily adapted to accept both upper and lower case characters by typing "L" while in the command mode, and then answering "Y" to the computer's question: LOWER CASE CHARACTERS (Y/N)?

Extended BASIC must be configured to accept lower case characters. Start with the distribution tape and add any required patches. Then follow the instructions in Appendix A of the BASIC section of your software manual. Type the prompt L, to which the computer will respond: LOWER CASE (Y/N)? A "Y" response will configure the software to accept both upper and lower case characters as String Data.

MONITORS FOR THE 6800

Dear Dr. Dobb's, *Received: 78 June 14*

The Motorola 6800 microprocessor is gaining in popularity against fantastic odds. It has been estimated that some 26 million lines of code have been written for the Intel 8080 so that all the 6800 owners are busy writing to catch up.

Southwest Technical Products Corp. has helped the 6800's popularity with selling systems at prices lower than most competitors. When they introduced their first systems, the operating system (monitor) provided was MIKBUG. This monitor has several problems and for the serious user its replacement is mandatory. The latest SWTPC systems now have SWATBUG, a monitor system developed at SWTPC. This system is also in a MC6830 ROM. For those users who own a PERCOM, SWTPC, or similar cassette interface, this system does not activate the RC line and the cassette interface must be operated manually. M6800 owners do not despair, help is on the way. First, Shifting Sands Microcomputer Products Corporation, Box 441, Fairborn, Ohio 45324, manufactures the RC-68A which is a PC board with sockets to allow user to plug in 2708 EPROM in place of the 6830. This allows one to use 2708 monitors such as the SMARTBUG monitor sold by Smokesignal Broadcasting Co. or to customize one's own system when access to programming facilities for 2708's are available.

The purchase of the PC board, SMARTBUG, and an MP-S board which is a must for I/O will certainly upgrade any old system. This is about the only way to allow use of the Percom CIS-30+ at 1200 baud, which is the rate used for local storage by this user, and this done without any errors for over a period of 10 months of extensive software development.

The owners of 6800 should look forward to a PASCAL and FORTRAN within the next year so that an update is probably in order to keep abreast of future advances. I also recommend for the user the Shifting Sands Dissassembler and Display software. For display make the following patches if one has a terminal such as the Lear Seigler ADM3A which allows 80 char/line.

ADDRESS	PATCH
$5E4E	$01
$5E4F	$01
$5E50	$01
$5E51	$01
$5E52	$01
$5E53	$01
$5E55	$01
$5E83	$01
$5E84	$01

I hope that this information will be of benefit to your readers if you see fit to publish same.

Sincerely,
Chuck Adams

Dept. of Computer Sciences
North Texas State University
Denton, Texas

A Programmable IC Tester for KIM

BY JOEL SWANK
4655 SW 142nd St. #186
Beaverton, OR 97005

With a few dollars in parts and a small program, KIM can be used to test most SSI integrated circuits and some MSI devices. The best thing about this tester is that it is automatically programmable. You don't sit over the data book trying to figure out what a device is supposed to do. You just plug in a known good device of the type under test, press the program key, and the tester is programmed to test that device.

This idea was published in the June '78 *Byte* by Mark Thorson, but his implementation is a black box with hard-wired logic and memory. I have moved all logic into software leaving some inverter/buffers, some wire jumpers, and the test socket as hardware.

The program communicates with the IC through a PIA (Peripheral Interface Adapter). The 8 bits of PIA port A are used to receive signals from the IC and the 8 bits of PIA port B are used to send signals to the IC. While being programmed the tester sends each of the 256 possible combinations of the 8 bits on port B to the IC and the response to each is recorded in memory. During testing, each combination is sent again in the same order, and the response is compared with that in memory. A result of not equal means a bad IC.

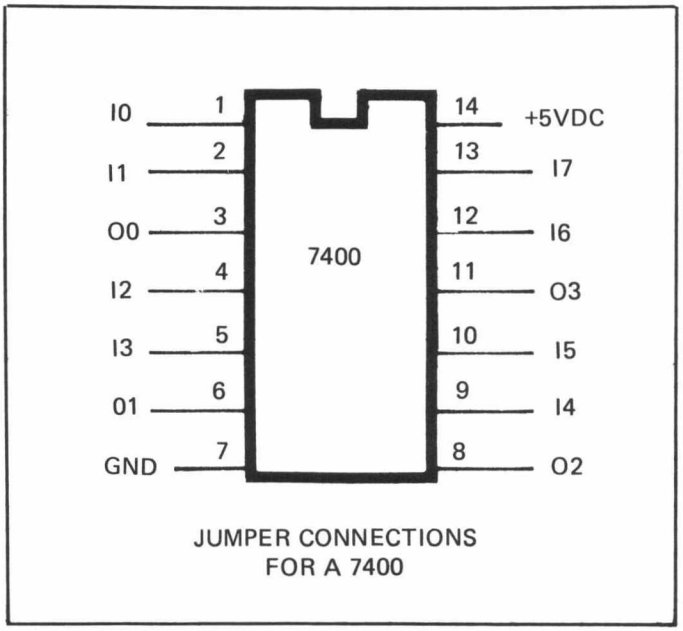
Since different IC's use different pins for input and output, it is also necessary to connect the PIA input and output lines to the proper pins of the test socket. This is most easily accomplished by jumper wires and some easily changeable type of connector. Unneeded inputs or outputs can be left unconnected. The power and ground must also be jumped to the proper pins.

The B port of the PIA is also used in automatic pulse mode to send a 1 microsecond pulse on its CB2 control line. This pulse occurs each time data is written to the B port. The CB2 line is inverted to become the CLK and inverted again to become the $\overline{\text{CLK}}$. These 2 signals are provided for devices that need a clock signal to strobe in data.

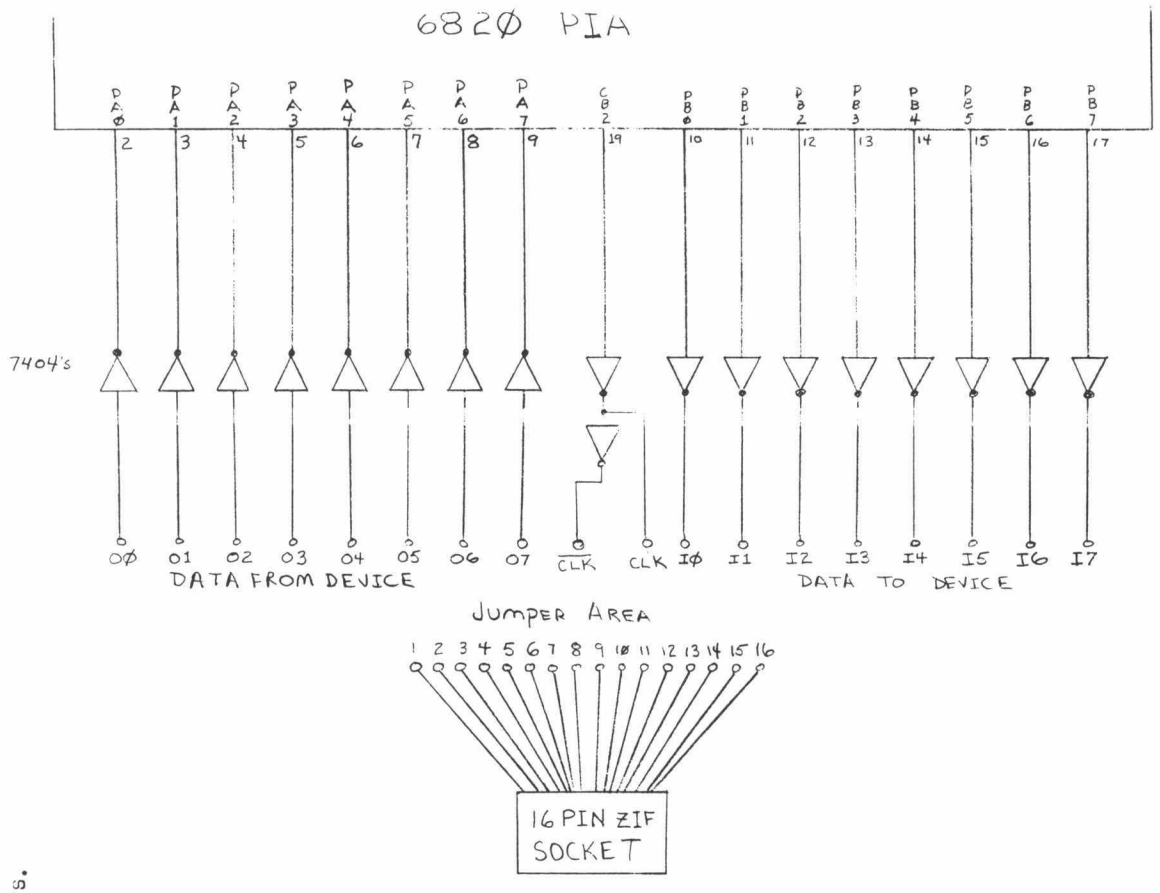
To use the tester, start execution at location 200. The KIM display will light with 0's. To program the tester, insert a good IC of the type to be tested in the test socket. Be sure that the jumpers are properly set. Press the PC key and the tester will program itself. This builds the pattern of responses in memory page 3. To test, plug the device to be tested into the socket and press the DA key. If the display lites with all 0's the device is good. If an error is encountered the display will contain information about the error. The left 2 digits contain the byte sent to the tester. The middle 2 digits contain the byte received from the tester. The rightmost 2 digits contain the byte from the pattern. The AD key returns control to KIM.

This tester is able to test most 14 and 16 pin TTL circuits. If 74LS TTL is used for the inverters, CMOS can also be tested. If care is taken with the jumper programming, some MSI devices can be tested, and by expanding the software more complicated devices could be checked.

The program of the tester is the 256 byte pattern in memory. This pattern completely and unambiguously defines the logic of the device being tested. Patterns for each device could be stored in a file for later use. Of course the jumpers would have to be programmed exactly the same to use the pattern again later. Are there other uses for these patterns? Could a program be used to generate the logic from these patterns? Maybe these or something similar could be used by the manufacturer to describe a device. The data could be used to program everyone's programmable tester.



JUMPER CONNECTIONS
FOR A 7400




```
:Z
1000            ;   PROGRAMMABLE IC TESTER
1010            ;
1020            ;   FUNCTION:
1030            ;       TO TEST SSI AND MSI INTEGRATED CIRCUITS.
1040            ;
1050            ;   ZERO PAGE STORAGE
1060            ;
1070            PATL .DL 00B0
1080            PATH .DL 00B1
1090            KESA .DL 00B2
1100            INH  .DL 00F9
1110            PNTL .DL 00FA
1120            PNTH .DL 00FB
1130            ;
1140            ;   PIA COMMUNICATIONS
1150            ;
1160            DTAA .DL DA00
1170            DIRA .DL DA00
1180            CTLA .DL DA01
1190            DTAB .DL DA02
1200            DIRB .DL DA02
1210            CTLB .DL DA03
1220            ;
1230            ;   MONITOR ADDRESSES
1240            ;
1250            SCND .DL 1F1F
1260            STRT .DL 1C4F
1270            GETK .DL 1F6A
1280            ;
1290                 .CR 0200
1300            ;
1310 0200 D8    INIT CLD                    INITIALIZE I/O PORTS
1320 0201 A9 38      LDA 38
1330 0203 8D 01 DA   STA CTLA
1340 0206 8D 03 DA   STA CTLB
1350 0209 A2 FF      LDX 0FF                PORT B IS OUTPUT
1360 020B 8E 02 DA   STX DIRB
1370 020E E8         INX                    PORT A IS INPUT
1380 020F 8E 00 DA   STX DIRA
1390 0212 A9 3C      LDA 3C
1400 0214 8D 01 DA   STA CTLA
1410 0217 A9 2C      LDA 2C
1420 0219 8D 03 DA   STA CTLB
1430 021C A9 00      CLR LDA 00             CLEAR DISPLAY
1440 021E 85 FB      STA *PNTH
1450 0220 85 FA      STA *PNTL
1460 0222 85 F9      STA *INH
1470            ;
1480 0224 20 1F 1F   ALUP JSR SCND          DISPLAY
1490 0227 20 6A 1F   JSR GETK               GET KEY
1500 022A C5 E2      CMP *KESA              SAME KEY AS LAST TIME?
1510 022C F0 F6      BEQ ALUP               YES , IGNORE
1520 022E 85 E2      STA *KESA
1530 0230 C9 15      CMP 15                 VALID KEY?
1540 0232 B0 F0      BCS ALUP               NOPE
1550 0234 C9 11      CMP 11                 DA KEY?
1560 0236 F0 1E      BEQ TEST               YES TEST
1570 0238 C9 14      CMP 14                 PC KEY?
1580 023A F0 07      BEQ PROG               YES PROGRAM
1590 023C C9 10      CMP 10                 AD KEY?
1600 023E D0 E4      BNE ALUP               NO IGNORE
1610 0240 4C 4F 1C   JMP STRT               YES BACK TO KIM
1620            ;
1630            ;   ROUTINE TO GENERATE TEST PATTERN FROM
1640            ;       A KNOWN GOOD DEVICE
```

```
1650                    ;
1660 0243 A9 03         PROG LDA #3          INIT PATTERN FIELD POINTER
1670 0245 85 B1              STA *PATH
1680 0247 A0 00              LDY #0
1690 0249 84 E0              STY *PATL
1700 024B 98            PLUP TYA
1710 024C 20 70 02           JSR PULS        SEND , RECEIVE
1720 024F 91 B2              STA (PATL),Y    SAVE RETURNED DATA
1730 0251 C8                 INY
1740 0252 D0 F7              BNE PLUP        256 TIMES
1750 0254 F0 C6              BEQ CLR         DONE
1760                    ;
1770                    ;    ROUTINE TO TEST UNKNOWN DEVICE USING
1780                    ;        PRE-GENERATED PATTERN
1790                    ;
1800 0256 A0 00         TEST LDY #0
1810 0258 98            TLUP TYA
1820 0259 20 70 02           JSR PULS        SEND , RECEIVE
1830 025C D1 B2              CMP (PATL),Y    CORRECT ANSWER?
1840 025E D0 05              BNE ERRO        NO
1850 0260 C8                 INY
1860 0261 D0 F5              BNE TLUP        256 TIMES
1870 0263 F0 B7              BEQ CLR         DONE
1880                    ;
1890 0265 84 FB         ERRO STY *PNTH       DATA SENT
1900 0267 85 FA              STA *PNTL       DATA RECEIVED
1910 0269 B1 B2              LDA (PATL),Y
1920 026B 85 F9              STA *INH        DATA EXPECTED
1930 026D 4C 24 02           JMP ALUP
1940                    ;
1950                    ;    PULS : SEND BYTE IN A , TOGGLE CLOCK ,
1960                    ;           AND RETRIEVE DATA RETURNED.
1970                    ;
1980 0270 8D 02 DA      PULS STA DTAB        SEND DATA
1990 0273 AD 00 CA           LDA DTAA        DATA RETURNED
2000 0276 60                 RTS
2010                    ;
2020                    ;    END$ .EN
:A

SYMBOL TABLE
ALUP   0224
CLR    021C
CTLA   DA01
CTLB   DA03
DIRA   DA00
DIRB   DA02
DTAA   DA00
DTAB   DA02
END$   0277
ERRO   0265
GETK   1F6A
INH    00F9
INIT   0200
KESA   00B2
PATH   00B1
PATL   00B0
PLUP   024B
PNTH   00FB
PNTL   00FA
PROG   0243
PULS   0270
SCND   1F1F
STRT   1C4F
TEST   0256
TLUP   0258
```

```
D
>D 300,3FF
0300  0F 0E 0E 0E  0D0C0C0C  0D0C0C3C  0D0C0C0C  [................]
0310  0E0A0A0A  0908080C  0908083C  0908080C  [................]
0320  0E0A0A0A  0908080C  0908083E  0908080C  [................]
0330  0E0A0A0A  0908080C  0908080C  0908080C  [................]
0340  0706060C  0504040C  0504040C  0504040C  [................]
0350  0302020C  0100000C  0100000C  0100000C  [................]
0360  0302020C  0100000C  0100000C  0100000C  [................]
0370  0302020C  0100000C  0100000C  0100000C  [................]
0380  0706060C  0504040C  0504040C  0504040C  [................]
0390  0302020C  0100000C  0100000C  0100000C  [................]
03A0  0302020C  0100000C  0100000C  0100000C  [................]
03B0  0302020C  0100000C  0100000C  0100000C  [................]
03C0  0706060C  0504040C  0504040C  0504040C  [................]
03D0  0302020C  0100000C  0100000C  0100000C  [................]
03E0  0302020C  0100000C  0100000C  0100000C  [................]
03F0  0302020C  0100000C  0100000C  0100000C  [................]
>
$A2
>       PATTERN FOR A 7402 QUAD NAND GATE
```


LIGHT PEN FOR COMMODORE PET

News Release *Received: 78 July 17*

A self-contained light pen which plugs directly into the Commodore PET 2001 user port has been announced by the 3G Company. This light pen makes it possible to bypass the PET's keyboard and interact directly with the information displayed on the CRT screen.

The light pen adds versatility to most graphics programs and is valuable as a teaching aid for young children. It also adds unique capabilities for application programs aimed at the non-computer oriented person. The light pen is complete and ready to plug into the PET. A sample program and programming instructions comes with the pen. The entire package sells for $24.95 and is available from 3G Company at Rt. 3 Box 28 A, Gaston, Oregon 97119; (503) 985-7176.

NEW COMPUTER LAW

Dear Dr. Dobbs:

Received: 78 July 17

Computer users should know about SB1766, the "Federal Computer Systems Protection Act of 1977", now working its way through Congress. The vast, open-ended scope of its wording makes any use of almost any computer "without authorization" punishable by 15 years imprisonment or fifty thousand dollars fine.

While directed mainly against financial crime, the wording clearly covers students who play games with computer time intended for class assignment use, or employees who use company computer time for unapproved research on personal projects. Another problem concerns university and other computer centers where typically a few files are explicitly confidential, some are explicitly public, but most development work is accessible but not explicity public, essentially a kind of proto-publication of "anyone can look at this but I don't assume responsiblity for correctness." It is common practice to browse this work for the sole purpose of keeping up with the field and improving programming technique. Under the new law, any use of this vast and irreplacable community library could be interpreted as a felony.

The law could deter public use of open-access community bulletin boards (which now exist in several cities), open computer conferences, and other public data bases. These free computer services are more than just experiments or toys, as someone setting up one can be repaid many times over by up-to-the-moment, specialized professional information coming in. We can expect hundreds of scholars or groups to start open-access computer conferences or data centers for specialized interest areas (not necessarily computer oriented) as terminals become more widespread. Although using these services would be entirely legal under the proposed law, it may be hard to communicate which ones are open, because typically no advance arrangements are made. Commonly you hear through the grapevine "call this number and hit carriage return", and often there is no way to even find out who is running it, short of attempting to log in. If the computer turned out to be restricted to members of a certain organization, the law would be broken, and usually a record of the call would exist.

Essentially this proposed law covers every possibility and leaves the actual definition of what is or is not legal to prosecutors and courts. This is poor legal draftsmanship, as overzealous prosecution does exist, especially in hot new areas of law enforcement, and defendants are stripped of legal protection.

This proposed law covers so many legitimate activities that it threatens the natural growth of decentralized computer use. It appears to have been drafted without proper input from industry. Congress must hear from industrial and professional groups, or it will pass a law with effects which could not possibly have been intended.

Sincerely,
John S. James

1090 Miller Ave.
Berkeley, CA 94708

95th CONGRESS
1st Session

S. 1766

IN THE SENATE OF THE UNITED STATES

June 27 (legislative day, May 18), 1977

Mr. Ribicoff (for himself, Mr. Domenici, Mr. Griffin, Mr. Heinz, Mr. Jackson, Mr. Kennedy, Mr. Metcalf, Mr. Percy, and Mr. Thurmond) introduced the following bill; which was read twice and referred to the Committee on the Judiciary

A BILL

To amend title 18, United States Code, to make a crime the use, for fraudulent or other illegal purposes, of any computer owned or operated by the United States, certain financial institutions, and entities affecting interstate commerce.

Be it enacted by the Senate and House of Representatives of the United States of America in Congress assembled,

That this act may be cited as the "Federal Computer Systems Protection Act of 1977".

SEC 2. The congress finds that –

(1) computer related crime is a growing problem in the Government and in the private sector;

(2) such crime occurs at great cost to the public since losses for each incident of computer crime tend to be far greater than the losses associated with each incident of other white collar crime;

(3) the opportunities for computer related crimes in Federal programs, in financial institutions, and in other entities which operate in interstate commerce through the introduction of fraudulent records into a computer system, unauthorized use of computer facilities, alteration or destruction of computerized information files, and stealing of financial instruments, data, or other assets, are great;

(4) computer related crime directed at institutions operating in interstate commerce has a direct effect on interstate commerce; and

(5) the prosecution of persons engaged in computer related crime is difficult under current Federal criminal statutes.

SEC. 3 (a) Chapter 47 of title 18, United States Code, is amended by adding at the end thereof the following new section:

" 1028. Computer fraud

"(a) Whoever directly or indirectly accesses or causes to be accessed any computer, computer system, computer network, or any part thereof which, in whole or in part, operates in interstate commerce or is owned by, under contract to, or operated for, on behalf of, or in conjunction with, any financial institution, the United States Government, or any branch, department, or agency thereof, or any entity operating in or affecting interstate commerce, for the purpose of (1) devising or executing any scheme or artifice to defraud, or (2)

obtaining money, property, or services by means of false or fraudulent pretenses, representations, or promises, shall be fined not more than $50,000, or imprisoned not more than fifteen years, or both.

"(b) Whoever, intentionally and without authorization, directly or indirectly accesses, alters, damages, or destroys any computer, computer system, or computer network described in subsection (a), or any computer software, program, or data contained in such computer, computer system, or computer network, shall be fined not more than $50,000, or imprisoned not more than fifteen years, or both.

"(c) For purposes of this section, the term –

"(1) 'access' means to approach, instruct, communicate with, store data in, retrieve data from, or otherwise make use of any resources of, a computer, computer system, or computer network;

"(2) 'computer' means an electronic device which performs logical, arithmetic, and memory functions by the manipulations of electronic or magnetic impulses, and includes all input, output, processing, storage, software, or communication facilities which are connected or related to such a device in a system or network;

"(3) 'computer system' means a set of related, connected or unconnected, computer equipment, devices, and software;

"(4) 'computer network' means the interconnection of communication lines with a computer through remote terminals, or a complex consisting of two or more interconnected computers;

"(5) 'property' includes, but is not limited to, financial instruments, information, including electronically produced data, and computer software and programs in either machine or human readable form, and any other tangible or intangible item of value;

"(6) 'services' includes, but is not limited to, computer time, data processing, and storage functions;

"(7) 'financial instrument' means any check, draft, money order, certificate of deposit, letter of credit, bill of exchange, credit card, or marketable security;

"(8) 'computer program' means a series of instructions or statements, in a form acceptable to a computer, which permits the functioning of a computer system in a manner designed to provide appropriate products from such computer system;

"(9) 'computer software' means a set of computer programs, procedures, and associated documentation concerned with the operation of a computer system;

"(10) 'financial institution' means –

"(A) a bank with deposits insured by the Federal Deposit Insurance Corporation;

"(B) a member of the Federal Reserve including any Federal Reserve bank;

"(C) an institution with accounts insured by the Federal Savings and Loan Insurance Corporation;

"(D) a credit union with accounts insured by the National Credit Union Administration;

"(E) a member of the Federal home loan bank systems and any home loan bank;

"(F) a member or business insured by the Securities Investor Protection Corporation; and

"(G) a broker-dealer registered with the Securities and Exchange Commission pursuant to section 15 of the Securities and Exchange Act of 1934.".

(c) The table of sections of chapter 47 of title 18, United States Code, is amended by adding at the end thereof the following:

"1028. Computer fraud".

COMPAS EDITOR AND ASSEMBLER FOR 6500 FAMILY

News Release *Received: 78 June 23*

Computer Applications Corporation announces the immediate availability of two software packages for the 6500 microprocessor family made by Rockwell, Synertek and MOS/Technology. These are a text editor (E/65) and two-pass assembler (A/65) designed to run on any 6500 based system. Both programs are similar to software written by COMPAS staff and offered in the Rockwell System 65. Each program is designed with transfer vectors to link the program to I/O routines in the user's system.

E/65 is primarily designed to edit assembler source code. Line oriented commands specify input/output of text and find specific lines to be edited. String oriented commands allow the user to search for and optionally change a text string. Character oriented commands allow cursor positioning and character deletion. Text may be loaded from or dumped to a bulk device other than the system terminal. E/65 also features a second entry point to edit text which is already in memory.

A/65 is a two-pass assembler which conforms to the specifications detailed in the Rockwell and MOS/Technology cross assembler manuals. A full range of run-time options are provided to control listing formats, printing of generated code for ASCII strings and generation of object code. Object code may be stored directly in memory or output to a file which may be the same as or different from the listing device. Thus an assembly may be made for listing only, object code only or both.

E/65 and A/65 are priced at $100 each (Prepaid) and are supplied in object form on paper tape or KIM compatible audio tape. Listings of the source code are available for $25 each. Programs are assembled to start at memory location 200 (hex) but may be ordered to start at any location if a listing is ordered. Full documentation on the installation and use of each package is provided. Contact COMPAS, 413 Kellogg, Ames IA 50010; (515) 232-8181.

RUMBLINGS ABOUT DIGITAL GROUP

Dated: 78 July 25

Persistent rumors have been circulating about Digital Group's financial troubles including the suggestion they were about to file Chapter 11. A call to Digital Group gleaned the following: Ms. Edgar, secretary to DG president Dick Bemis was rather candid about the fact that the company was having financial trouble but that they had had a meeting with their creditors who were working with them to solve the problem. She assured me that Digital Group was *not* filing for Chapter 11. – TRW

Dumping Northstar Disk Files

BY TITUS PURDIN
5901 JFK # 101
North Little Rock, AR 72116

Received: 78 Jun 20

Dear Sirs,

Enclosed is a small utility program which I have found very helpful in designing and debugging disk files. I found that, with my limited operating system, I could not look at the contents of such files in absolute form. I was wasting a lot of time writing utility programs in BASIC to read disk files in logical form as an aid in tracking down bugs. Problems such as end-of-record marks spilling over into subsequent records simply defied detection.

This program was designed to run in a Northstar environment and uses the disk drivers in Northstar DOS, but could be adapted to run with other operating systems. The program reads named disk files of any file type and prints the contents, 16 bytes per line, first in hex and then in ASCII. Each line is numbered to show the relative location of each byte in the file in hex. Non-printing ASCII characters are replaced by periods (.) in the print out. The print out can be interrupted with a line feed (OAH) and restarted with a carriage return (ODH). The print out can be aborted with a control-C. In the event of an abort, certain input errors, or normal program termination control is passed to Northstar DOS.

```
0001 ;  THIS PROGRAM PRINTS THE CONTENTS OF DISK FILES
0002 ;  WRITTEN IN NORTHSTAR FORMAT.  CONTENTS ARE PRINTE
0003 ;  16 BYTES PER LINE IN BOTH HEX AND ASCII.
0004 ;
0005           ORG   2A00H
0010 ;
0020 ;  PRINT MAIN HEADER
0030 ;
2A00 0E2F      0040 START:  MVI   C,47
2A02 21452C    0050         LXI   H,HDR1
2A05 CDEB2B    0060         CALL  PRINT
2A08 CDF52B    0070         CALL  CRLF
               0080 ;
               0090 ;  PRINT FILE NAME PROMPT
               0100 ;
2A0B 0E0C      0110 INPUT:  MVI   C,14
2A0D 21742C    0120         LXI   H,HDR2
2A10 CDEB2B    0130         CALL  PRINT
2A13 21F02C    0140         LXI   H,FILE
2A16 0E0A      0150         MVI   C,10
               0160 ;
               0170 ;  BLANK INPUT AREA
               0180 ;
2A18 3620      0190 ZFILE:  MVI   M,20H
2A1A 23        0200         INX   H
2A1B 0D        0210         DCR   C
2A1C C2182A    0220         JNZ   ZFILE
2A1F 21F02C    0230         LXI   H,FILE
2A22 0E09      0240         MVI   C,9
               0250 ;
               0260 ;  ACCEPT FILE NAME.  USES REG C AS COUNTER AND
               0261 ;  ACCEPTS UP TO 8 CHARACTERS.  TAKES CARE OF
               0262 ;  BLANKING CHAR IN RESPONSE TO BACKSPACE (5FH)
               0263 ;  RTN TERMINATES WHEN CR (0DH) IS ENCOUNTERED.
               0270 ;
2A24 CD3A2C    0280 INFIL:  CALL  CIN
2A27 3A512E    0290         LDA   HOLD
2A2A FE0D      0300         CPI   0DH
2A2C CA6B2A    0310         JZ    INUNT
2A2F FE5F      0320         CPI   5FH
2A31 C2502A    0330         JNZ   CONT1
2A34 3E09      0340         MVI   A,9
2A36 B9        0350         CMP   C
2A37 CA242A    0360         JZ    INFIL
2A3A 3A512E    0370         LDA   HOLD
2A3D 2B        0380         DCX   H
2A3E 0C        0390         INR   C
2A3F 47        0400         MOV   B,A
2A40 CD412C    0410         CALL  COUT
2A43 0620      0420         MVI   B,20H
2A45 CD412C    0430         CALL  COUT
2A48 065F      0440         MVI   B,5FH
2A4A CD412C    0450         CALL  COUT
2A4D C3242A    0455         JMP   INFIL
               0460 ;
               0461 ;  PLACES CHARACTERS FROM INFIL: SEQUENTIALLY
               0462 ;  INTO AREA CALLED 'FILE' TO BUILD INPUT
               0463 ;  FILE NAME.  PRINTS 'NAME TOO LONG' ERROR
               0464 ;  IF 8 CHARACTERS ARE EXCEEDED.
               0465 ;
2A50 77        0470 CONT1:  MOV   M,A
2A51 47        0480         MOV   B,A
2A52 CD412C    0490         CALL  COUT
2A55 23        0500         INX   H
2A56 0D        0510         DCR   C
2A57 C2242A    0520         JNZ   INFIL
2A5A CDF52B    0530         CALL  CRLF
2A5D 0E10      0540         MVI   C,16
2A5F 21822C    0550         LXI   H,HDR3
2A62 CDEB2B    0560         CALL  PRINT
2A65 CDF52B    0570         CALL  CRLF
2A68 C30B2A    0580         JMP   INPUT
               0581 ;
               0582 ;  ACCEPT DISK UNIT NUMBER.  CHECKS TO MAKE
               0583 ;  SURE INPUT IS ASCII 1,2, OR 3.
               0584 ;
2A6B 0E0C      0590 INUNT:  MVI   C,12
2A6D 21922C    0600         LXI   H,HDR4
2A70 CDF52B    0610         CALL  CRLF
2A73 CDEB2B    0620         CALL  PRINT
2A76 21FA2C    0630         LXI   H,UNIT
2A79 CD3A2C    0640         CALL  CIN
2A7C 47        0650         MOV   B,A
2A7D CD412C    0660         CALL  COUT
2A80 FE31      0670         CPI   31H
2A82 CAA02A    0680         JZ    GETCR
2A85 FE32      0690         CPI   32H
2A87 CAA02A    0700         JZ    GETCR
2A8A FE33      0710         CPI   33H
2A8C CAA02A    0720         JZ    GETCR
               0721 ;
```

```
                    0722 ;  PRINTS 'UNVALID UNIT' ERROR IF INPUT
                    0723 ;  FAILS EDITS IN PREVIOUS SECTION.
                    0724 ;
2A8F CDF52B         0730 ERUNT:  CALL   CRLF
2A92 0E0C           0740         MVI    C,12
2A94 219E2C         0750         LXI    H,HDR5
2A97 CDEB2B         0760         CALL   PRINT
2A9A CDF52B         0770         CALL   CRLF
2A9D C36B2A         0780         JMP    INUNT
                    0781 ;
                    0782 ;  TRIMS ASCII INPUT UNIT # TO BINARY
                    0783 ;  VALUE AND STORES IT IN AREA CALLED 'UNIT'.
                    0784 ;  ACCEPTS CR TO CONTINUE.  ANY OTHER INPUT
                    0785 ;  CAUSES 'INVALID UNIT' ERROR.  PUTS
                    0786 ;  ADDRESS OF FILE NAME AT 'FILE' IN REG H/L
                    0787 ;  AND VALUE IN 'UNIT' IN REG A.  CALLS DOS
                    0788 ;  DIRECTORY LOOKUP RTN AT 201CH.  IF CARRY
                    0789 ;  BIT IS SET ON RETURN, 'COULD NOT FIND'
                    0790 ;  ERROR IS PRINTED.
                    0791 ;
2AA0 E60F           0795 GETCR:  ANI    0FH
2AA2 77             0800         MOV    M,A
2AA3 CD3A2C         0810         CALL   CIN
2AA6 47             0820         MOV    B,A
2AA7 CD412C         0830         CALL   COUT
2AAA FE0D           0840         CPI    0DH
2AAC C28F2A         0850         JNZ    ERUNT
2AAF 060A           0860         MVI    B,0AH
2AB1 CD412C         0870         CALL   COUT
2AB4 3AFA2C         0880         LDA    UNIT
2AB7 21F02C         0890         LXI    H,FILE
2ABA CD1C20         0900         CALL   201CH
2ABD D2DE2A         0910         JNC    STDAT
2AC0 0E13           0920         MVI    C,19
2AC2 21AA2C         0930         LXI    H,HDR6
2AC5 CDEB2B         0940         CALL   PRINT
2AC8 0E08           0950         MVI    C,8
2ACA 21F02C         0960         LXI    H,FILE
2ACD CDEB2B         0970         CALL   PRINT
2AD0 0E0D           0980         MVI    C,13
2AD2 21BD2C         0990         LXI    H,HDR7
2AD5 CDEB2B         1000         CALL   PRINT
2AD8 CDF52B         1010         CALL   CRLF
2ADB C30B2A         1020         JMP    INPUT
                    1021 ;
                    1022 ;  STORES ADDRESS OF FILE ON DISK IN 'ADDR'
                    1023 ;
2ADE 0E08           1030 STDAT:  MVI    C,8
2AE0 11442E         1040         LXI    D,ADDR
2AE3 7E             1050 CONT3:  MOV    A,M
2AE4 12             1060         STAX   D
2AE5 23             1070         INX    H
2AE6 13             1080         INX    D
2AE7 0D             1090         DCR    C
2AE8 C2E32A         1100         JNZ    CONT3
2AEB 0E48           1110         MVI    C,72
2AED 21FC2C         1120         LXI    H,PLINE
                    1121 ;
                    1122 ;  MOVES SPACES TO 72 CHAR PRINT LINE
                    1123 ;  MADE UP OF 'PLINE', 'HEX', AND 'ASCII'.
                    1124 ;
2AF0 3620           1130 ZPLIN:  MVI    M,20H
2AF2 23             1140         INX    H
2AF3 0D             1150         DCR    C
2AF4 C2F02A         1160         JNZ    ZPLIN
2AF7 0E04           1170         MVI    C,4
2AF9 21FC2C         1180         LXI    H,PLINE
                    1181 ;
                    1182 ;  MOVES ZEROS TO FIRST FOUR POSITIONS OF
                    1183 ;  PRINT LINE.
                    1184 ;
2AFC 3630           1190 ZNUMB:  MVI    M,30H
2AFE 23             1200         INX    H
2AFF 0D             1210         DCR    C
2B00 C2FC2A         1220         JNZ    ZNUMB
                    1221 ;
                    1222 ;  PLACES UNIT # IN REG C, ADDRESS OF INPUT DATA
                    1123 ;  AREA IN REG D/E AND ADDRESS OF BLOCK TO BE READ
                    1124 ;  ON THE DISK IN REG H/L.  CALLS NORTHSTAR DISK
                    1125 ;  READ RTN AT 2022H.  PRINTS 'ARGUMENT ERROR' IF
                    1126 ;  CARRY BIT IS SET ON RETURN.
                    1127 ;
2B03 3E01           1230 RDBLK:  MVI    A,1
2B05 0601           1240         MVI    B,1
2B07 21FA2C         1250         LXI    H,UNIT
2B0A 4E             1260         MOV    C,M
2B0B 11442D         1270         LXI    D,DATA
2B0E 2A442E         1280         LHLD   ADDR
2B11 CD2220         1290         CALL   2022H
2B14 D2252B         1300         JNC    INIT
2B17 0E0E           1310         MVI    C,14
2B19 21CA2C         1320         LXI    H,HDR8
2B1C CDEB2B         1330         CALL   PRINT
2B1F CDF52B         1340         CALL   CRLF
2B22 C32820         1350         JMP    2028H
                    1351 ;
                    1352 ;  PUTS ADDRESS OF BEGINNING OF DATA AREA IN
                    1353 ;  REG D/E AND MOVES ZERO TO 'BLKCT' WHICH
                    1354 ;  WILL COUNT LINES FROM DATA AREA AS THEY
                    1355 ;  ARE PRINTED.
                    1356 ;
2B25 11442D         1360 INIT:   LXI    D,DATA
2B28 AF             1370         XRA    A
2B29 324E2E         1380         STA    BLKCT
                    1381 ;
                    1382 ;  MOVES ZERO TO 'HCTR' AND 'ACTR' WHICH
                    1383 ;  COUNT CHARACTERS AS EACH LINE IS BUILT.
                    1384 ;
2B2C AF             1390 NXTLN:  XRA    A
2B2D 324C2E         1400         STA    HCTR
2B30 324D2E         1410         STA    ACTR
                    1411 ;
                    1412 ;  BUILDS THE ADDRESS IN PRINT LINE WHERE
                    1413 ;  CHARACTER WILL BE PUT IN REG H/L.
                    1414 ;
2B33 0600           1420 PROC:   MVI    B,0
2B35 3A4C2E         1430         LDA    HCTR
2B38 4F             1440         MOV    C,A
2B39 21022D         1450         LXI    H,HEX
2B3C 09             1460         DAD    B
2B3D 1A             1470         LDAX   D
2B3E 0604           1480         MVI    B,4
                    1481 ;
                    1482 ;  GETS THE HIGH ORDER NIBBLE OF CURRENT
                    1483 ;  BYTE FROM DATA AREA.  CONVERTS IT TO ASCII
                    1484 ;  CHARACTER.  STORES IT IN PRINT LINE.  GETS
                    1485 ;  LOW ORDER NIBBLE.  CONVERTS IT AND STORES
                    1486 ;  ALSO.  'HCTR' COUNTS HEX CHAR AND USES REG C.
                    1487 ;  'ACTR' COUNTS ASCII CHARACTERS AND USES REG C.
                    1488 ;  A PERIOD (.) IS SUBSTITUTED FOR ANY NON
                    1489 ;  PRINTING ASCII VALUE.
                    1490 ;
2B40 0F             1495 SHIFT:  RRC
2B41 05             1500         DCR    B
2B42 C2402B         1510         JNZ    SHIFT
2B45 E60F           1520         ANI    0FH
2B47 CD002C         1530         CALL   CVNHT
2B4A 77             1540         MOV    M,A
2B4B 23             1550         INX    H
2B4C 1A             1560         LDAX   D
```

```
2B4D E60F      1570          ANI   0FH
2B4F CD002C    1580          CALL  CVNRT
2B52 77        1590          MOV   M,A
2B53 0C        1600          INR   C
2B54 0C        1610          INR   C
2B55 0C        1620          INR   C
2B56 79        1630          MOV   A,C
2B57 324C2E    1640          STA   HCTR
2B5A 0600      1650          MVI   B,0
2B5C 3A4D2E    1660          LDA   ACTR
2B5F 4F        1670          MOV   C,A
2B60 21342D    1680          LXI   H,ASCII
2B63 09        1690          DAD   B
2B64 1A        1700          LDAX  D
2B65 E67F      1710          ANI   7FH
2B67 0620      1720          MVI   B,20H
2B69 B8        1730          CMP   B
2B6A D26F2B    1740          JNC   PUTCH
2B6D 3E2E      1750          MVI   A,2EH
               1751 ;
               1752 ;  PLACES ASCII CHARACTER IN PRINT LINE.  CHECKS
               1753 ;  COUNTERS TO SEE IF PRINT LINE IS FULL.
               1754 ;
2B6F 77        1760 PUTCH:   MOV   M,A
2B70 0C        1770          INR   C
2B71 79        1780          MOV   A,C
2B72 324D2E    1790          STA   ACTR
2B75 13        1800          INX   D
2B76 0E10      1810          MVI   C,16
2B78 B9        1820          CMP   C
2B79 C2332B    1830          JNZ   PROC
               1840 ;
               1850 ;  PRINT DATA LINE RTN.  MONITORS KEYBOARD
               1851 ;  FOR LF (0AH) WHICH WILL TEMPORARILLY
               1852 ;  HALT PRINT.
               1860 ;
2B7C CDF52B    1870          CALL  CRLF
2B7F 0E48      1880          MVI   C,72
2B81 21FC2C    1890          LXI   H,PLINE
2B84 CDEB2B    1900          CALL  PRINT
2B87 CDF52B    1910          CALL  CRLF
2B8A AF        1920          XRA   A
2B8B CD182C    1930          CALL  HLTCK
2B8E FE0A      1940          CPI   0AH
2B90 C29B2B    1950          JNZ   CTRLC
               1951 ;
               1952 ;  THIS IS THE WAIT LOOP INITIATED BY A LF (0AH).
               1953 ;  IT LOOPS UNTIL A CR (0DH) IS DETECTED.
               1954 ;
2B93 CD182C    1960 WAIT:    CALL  HLTCK
2B96 FE0D      1970          CPI   0DH
2B98 C2932B    1980          JNZ   WAIT
               1981 ;
               1982 ;  THESTS FOR CTRL-C WHICH ABORTS THE PROGRAM
               1983 ;  AND RETURNS CONTROL TO DOS.
               1984 ;
2B9B FE03      1990 CTRLC:   CPI   03H
2B9D CADA2B    2000          JZ    ENDRN
2BA0 21FE2C    2010          LXI   H,PLINE+2
               2011 ;
               2012 ;  INCREMENTS THE HEX LINE NUMBER WHICH IS IN
               2013 ;  THE FIRST FOUR BYTES OF PRINT LINE.
               2014 ;
2BA3 34        2020 LNUMB:   INR   M
2BA4 3E3A      2030          MVI   A,3AH
2BA6 BE        2040          CMP   M
2BA7 C2AC2B    2050          JNZ   CHK47
2BAA 3641      2060          MVI   M,41H
               2061 ;
               2062 ;  TAKES CARE OF CARRY IN THE PREVIOUS
               2063 ;  LINE NUMBER INCREMENT RTN.
               2064 ;
2BAC 3E47      2070 CHK47:   MVI   A,47H
2BAE BE        2080          CMP   M
2BAF C2B92B    2090          JNZ   CKBLK
2BB2 3630      2100          MVI   M,30H
2BB4 2B        2110          DCX   H
2BB5 C3A32B    2120          JMP   LNUMB
2BB8 1A        2130          LDAX  D
               2131 ;
               2132 ;  CHECKS 'BLKCT' TO SEE IF IT HAS REACHED 16.
               2133 ;  IF IT HAS CONTROL GOES TO RTN TO SEE IF
               2134 ;  THERE ARE MORE BLOCKS IN THE FILE TO BE READ.
               2135 ;  IF NOT CONTROL GOES TO PROCESS THE NEXT LINE.
               2136 ;
2BB9 3A4E2E    2140 CKBLK:   LDA   BLKCT
2BBC 060F      2150          MVI   B,0FH
2BBE B8        2160          CMP   B
2BBF CAC92B    2170          JZ    MORBL
2BC2 3C        2180          INR   A
2BC3 324E2E    2190          STA   BLKCT
2BC6 C32C2B    2200          JMP   NXTLN
               2210 ;
               2220 ;  CHECK FOR MORE BLOCKS IN FILE
               2230 ;
2BC9 21442E    2240 MORBL:   LXI   H,ADDR
2BCC 34        2250          INR   M
2BCD 3A462E    2260          LDA   ADDR+2
2BD0 3D        2270          DCR   A
2BD1 CADA2B    2280          JZ    ENDRN
2BD4 32462E    2290          STA   ADDR+2
2BD7 C3032B    2300          JMP   RDBLK
               2310 ;
               2320 ;  END PROGRAM - ASK FOR 'MORE?'
               2330 ;
2BDA CDF52B    2340 ENDRN:   CALL  CRLF
2BDD 0E04      2350          MVI   C,4
2BDF 21D82C    2360          LXI   H,HDR9
2BE2 CDEB2B    2370          CALL  PRINT
2BE5 CDF52B    2380          CALL  CRLF
2BE8 C32820    2390          JMP   2028H
               2400 ;
               2410 ;  PRINT STRING AT ADDR IN REG H/L FOR
               2411 ;  NUMBER OF CHARACTERS IN REG C.
               2420 ;
2BEB 46        2430 PRINT:   MOV   B,M
2BEC CD412C    2440          CALL  COUT
2BEF 23        2450          INX   H
2BF0 0D        2460          DCR   C
2BF1 C2EB2B    2470          JNZ   PRINT
2BF4 C9        2480          RET
               2490 ;
               2500 ;  INSERT CR AND LF
               2510 ;
2BF5 060D      2520 CRLF:    MVI   B,0DH
2BF7 CD412C    2530          CALL  COUT
2BFA 060A      2540          MVI   B,0AH
2BFC CD412C    2550          CALL  COUT
2BFF C9        2560          RET
               2570 ;
               2580 ;  HEX TO ASCII CONVERSION RTN
               2590 ;
2C00 224F2E    2600 CVNRT:   SHLD  MARK
2C03 21E02C    2610          LXI   H,CONVT
2C06 0600      2620          MVI   B,0
2C08 B8        2630 SCAN:    CMP   B
2C09 CA112C    2640          JZ    HXCHR
2C0C 04        2650          INR   B
2C0D 23        2660          INX   H
2C0E C3082C    2670          JMP   SCAN
2C11 7E        2680 HXCHR:   MOV   A,M
2C12 0600      2690          MVI   B,0
2C14 2A4F2E    2700          LHLD  MARK
2C17 C9        2710          RET
```

```
                        2720 ;
                        2721 ;  CHECK FOR 0AH TO HALT PRINT TEMPORARILY
                        2722 ;  NOTE:  THIS MUST CONFORM TO YOUR INDIVIDUAL I/O
                        2723 ;  PORT CONFIGURATION
                        2724 ;
2C18 DB00               2725 HLTCK:  IN    0
2C1A E604               2730         ANI   04H
2C1C CA2B2C             2740         JZ    PORT2
2C1F DB00               2750         IN    0
2C21 E640               2760         ANI   40H
2C23 CA2B2C             2770         JZ    PORT2
2C26 DB01               2780         IN    1
2C28 E67F               2790         ANI   7FH
2C2A C9                 2800         RET
2C2B DB50               2810 PORT2:  IN    50H
2C2D E604               2820         ANI   04H
2C2F C8                 2830         RZ
2C30 DB50               2840         IN    50H
2C32 E640               2850         ANI   40H
2C34 C8                 2860         RZ
2C35 DB51               2870         IN    51H
2C37 E67F               2880         ANI   7FH
2C39 C9                 2890         RET
                        2900 ;
                        2910 ;  VECTOR FOR CHARACTER IN RTN
                        2920 ;  NOTE:  2900H SHOULD BE CHANGED TO THE
                        2921 ;         ADDRESS OF YOUR CHARACTER INPUT RTN
                        2922 ;         IN DOS I/O DRIVER AREA.
                        2923 ;
2C3A CD0029             2930 CIN:    CALL  2900H
2C3D 32512E             2940         STA   HOLD
2C40 C9                 2950         RET
                        2960 ;
                        2970 ;  VECTOR FOR CHARACTER OUT RTN
                        2980 ;  NOTE:  2903H SHOULD BE CHANGED TO THE
                        2981 ;         ADDRESS OF YOUR CHARACTER OUTPUT RTN
                        2982 ;         IN DOS I/O DRIVER AREA.
                        2983 ;
2C41 CD0329             2990 COUT:   CALL  2903H
2C44 C9                 3000         RET
                        3010 ;
                        3020 ; HEADERS
                        3030 ;
2C45 5468[illegible]    3040 HDR1:   DB    'THIS PROGRAM DUMPS NORTHSTAR FILES
2C68 494E2048455058     3050         DB    'IN HEX/ASCII'
2C74 4E414D452032060    3060 HDR2:   DB    'NAME OF FILE? '
2C82 4E414D452032070    3070 HDR3:   DB    'NAME IS TOO LONG'
2C92 4449534B20308080   3080 HDR4:   DB    'DISK UNIT #:'
2C9E 494E56414C3090     3090 HDR5:   DB    'INVALID UNIT'
2CAA 46494C452032100    3100 HDR6:   DB    'FILE WITH THE NAME '
2CBD 574153204E3110     3110 HDR7:   DB    'WAS NOT FOUND'
2CCA 415247554D3120     3120 HDR8:   DB    'ARGUMENT ERROR'
2CD8 444F4E453F3130     3130 HDR9:   DB    'DONE?'
2CDD 594553             3140 YES:    DB    'YES'
2CE0 30313233343150     3150 CONVT:  DB    '0123456789ABCDEF'
2CF0                    3160 FILE:   DS    10;HOLDS FILE NAME
2CFA                    3170 UNIT:   DS    2;HOLDS DISK UNIT #
2CFC                    3180 PLINE:  DS    6              ;LINE NUMBER FOR PRINT
2D02                    3190 HEX:    DS    50             ;HEX PORTION OF PRINT
2D34                    3200 ASCII:  DS    16             ;ASCII PORTION OF PRINT
2D44                    3210 DATA:   DS    256;BLOCK OF DATA FROM DISK
2E44                    3220 ADDR:   DS    8;ADDRESS OF BLOCK ON DISK
2E4C                    3230 HCTR:   DS    1;HEX CHARATER COUNTER
2E4D                    3240 ACTR:   DS    1;ASCII CHARATER COUNTER
2E4E                    3250 BLKCT:  DS    1              ;COUNT OF LINES IN BLOCK
2E4F                    3260 MARK:   DS    2              ;UTILITY AREA
2E51                    3270 HOLD:   DS    1              ;LAST CHAR FROM KEYBOARD
2E52                    3280 STOP:   END
```

```
?
*GO DUMP

THIS PROGRAM DUMPS NORTHSTAR FILES IN HEX/ASCII
NAME OF FILE? MAINT1
DISK UNIT #:2

0000  03 1C 50 41 49 4E 54 45 44 20 41 50 54 20 28 35   ..PAINTED APT (5
0010  20 47 41 4C 20 50 41 49 4E 54 29 50 41 43 10 30    GAL PAINT)PAC.0
0020  00 00 43 17 45 00 00 42 20 20 20 20 20 20 20 20   ..C.E..B
0030  20 20 20 20 50 41 43 10 30 00 00 43 17 45 00 00       PAC.0..C.E..
0040  42 20 00 00 00 41 03 1F 53 48 41 4D 50 4F 4F 45   B ...A..SHAMPOOE
0050  44 20 43 41 52 50 45 54 20 28 55 53 45 44 20 31   D CARPET (USED 1
0060  20 50 54 29 43 41 43 10 40 00 00 43 75 00 00 00    PT)CAC.@..CU...
0070  40 20 20 20 20 20 20 20 20 20 20 43 41 43 10 40 00  @          CAC.@.
0080  00 43 75 00 00 00 40 10 00 00 00 00 03 19 50 55   .CU...@.......PU
0090  54 20 4C 49 47 48 54 20 4F 56 45 52 20 4D 41 49   T LIGHT OVER MAI
00A0  4C 42 4F 58 45 4C 43 10 40 00 00 43 38 00 00 00   LBOXELC.@..C8...
00B0  40 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20   @
00C0  45 4C 43 10 40 00 00 43 38 00 00 00 40 14 50 00   ELC.@..C8...@.P.
00D0  00 43 03 21 57 4F 4D 45 4E 27 53 20 52 4F 4F 4D   .C.!WOMEN'S ROOM
00E0  20 54 4F 49 4C 45 54 20 53 54 4F 50 50 45 44 20    TOILET STOPPED
00F0  55 50 50 4C 43 10 40 00 00 43 10 00 00 00 00 20   UPPLC.@..C.....
0100  20 20 20 20 20 20 50 4C 43 10 40 00 00 43 10 00         PLC.@..C..
0110  00 00 00 77 00 00 00 42 03 21 55 53 45 44 20 4F   ...W...B.!USED O
0120  4C 44 20 50 41 52 54 20 46 49 58 45 44 20 54 4F   LD PART FIXED TO
0130  49 4C 2E 20 53 45 41 54 50 4C 43 10 40 00 00 43   IL. SEATPLC.@..C
0140  10 00 00 00 00 20 20 20 20 20 20 20 50 4C 43 10   .....        PLC.
0150  40 00 00 43 10 00 00 00 00 60 00 00 00 41 03 1C   @..C.....@...A..
0160  54 49 47 48 54 45 4E 20 54 55 42 20 46 41 55 43   TIGHTEN TUB FAUC

DONE
*
```

This is a sample run which dumps the contents of a type 3 (data) file.

VOCAL MEMORY DUMP

BY PHIL MORK
12 Woodland Rd.
Weston, MA 02193
(617) 894-7147

Dear Dr. Dobb, *Dated: 78 June 13*

Here's an offering for your journal, the Vocal Memory Dump. It was hand assembled, the listing is from my disassembler (hence no labels). The comments were added later. For $3 I can supply cassette tapes of VMD for those readers who don't want to type in the program or implement the input hardware. I'll include the resistor and capacitor for the output circuit. Specify Tarbell, Kansas City. or Digital Group format.

My hardware is a home made 8080-based CPU board on an S-100 bus. For memory I have 24K RAM plus another 256 bytes RAM and 1K PROM monitor on the CPU. I/O consists of six parallel input and output ports. I use a Digital group TVC video and cassette board, and a D to A which I use for music. My printer is a Kleinschmidt M-311. I have 256 x 208 dot video graphics implemented using a homebrew interface which uses one of my memory boards and gets timing from the video board. Software includes Digital Group Maxi-Basic, Software Technology's Music System, Wang's Palo Alto Tiny Basic and WSFN. These last two support my graphics hardware (PATB has added commands to ERASE the screen, PLOT a point, and DRAW a vector).

Keep up the good work! Best wishes to you and your "family".

Phil Mork

The vocal memory dump (VMD) program pronounces the octal value for each memory location in a block specified by the user. As such it is a useful tool for verifying memory contents (ever try to look at two listings at the same time?). It uses digitized speech waveforms in the form of a one bit sample every 155 sec., which means both hardware and memory requirements are minimal. It would seem that a one bit approximation (ON or OFF) would be completely unintelligable noise, but although it is indeed "lo-fi", it is quite intelligible and on par with many communications radios. The trick is to use a high pass filter before digitization and low pass filter on the "D to A" end.

The Software

The VMD requires slightly over 3K of memory, all of which may be ROM. It is assembled to start at location 000000 Octal, but may be loaded starting on any other page, then using the relocator program (listing 2) to fix jump addresses and memory references. The only modifications which might be needed to make the VMD run on any 8080 or Z-80 based computer would be the input subroutine (wait for keyboard, then input a character to the accumulator), the selection of the output port (now 013 octal) and a timing constant (now set for a 2 MHz clock).

The Hardware

To use the VMD program, you will need an 8080 or Z-80 or based computer with enough memory to hold the VMD and any other program you wish to "read". Also required is an ASCII keyboard interfaced to the computer, the most significant bit of an output port, an audio amp and speaker (I use a Hi-Fi), a resistor and capacitor. The amplifier should have a high impedance input (10kΩ or more) and be able to accept a signal of around 1 volt (the AUX is fine).

Doing It

Connect the most significant bit (bit 7) of an output port through a 1KΩ resistor and a 0.1 μF capacitor to the input of audio amp as shown in figure 1. Load the VMD program

(listing 1) anywhere in memory, as long as it starts at the beginning of a page. If other than page 0, load the relocator, set byte 000001 to the page number where the VMD now starts,and run the relocator. Make any needed changes in the keyboard subroutine (000230), and the output port number (000117). Turn on the amp, and run VMD. Type in two six digit split octal numbers: the starting and ending addresses of a block of memory (any non-octal characters will be considered octal, it's the 3 least significant bits that count). As you type, the digits should be echoed by a voice (my voice!). After 12 digits have been entered,the program takes off, pronouncing the octal values of memory locations until the end address is reached. The program is then ready to accept two more addresses. If the voice is too high or low (fast or slow), try playing with the timing constant at 000121. It should be OK for a 2MHz clock (no waits).

Your Own Thing

Octal output of the accumulator and digit output (3 least significant bits) are written as subroutines which can be called by external programs. Addresses are (000040) and (000063) respectively. The VMD is a simple-minded program which serves a useful function, but maybe more importantly, it demonstrates a simple way to give a computer vocal output. I chose octal for several reasons: I use octal and hex would require 8 more digits (another 3K of memory), and there might be some ambiguity between some of the digits (B,C,D, although I could use Bravo, Charlie, Delta, etc.). To digitize your own speech, use the circuit shown in figure 2. Input level should be around .5 volt, so you'll need a preamp for a microphone. What I did was tape the speech I wanted to digitize, then play the tape into the computer. You'll probably want to play with the input level some for best results. Use the programs in listing 3 to record and play back speech. I used the input program to record the digits for VMD and edited out the space between digits by hand. The sampling rate can be varied by changing byte 000071. As it stands it samples about 6500 bits per second (or 820 bytes/sec.). I can record about 30 seconds of speech in my 24K of memory. It would take around 24K to digitize the 64 ASCII characters. If you have a floppy, you might only need one word in memory at a time while the computer is pronouncing it and seeking the next word on the disk. I figure you could get over 600 words on an 8" diskette.

How about adding speech output to BASIC? Standard vocabulary might include the decimal digits, "point", "plus", "minus", "E", etc. to allow numeric output. Special messages for an application could be included in the BASIC program as an array of digitzed speech, or to avoid loading multi-K of speech, the program might ask you to say "shields weakening" or whatever during program initialization. This might also be fun for a conversation-type program.

Thanks are due to Jim Anderson for his talk and demonstration of speech digitization at the April '78 meeting of the New England Computer Society.

LISTING

```
VOCAL MEMORY DUMP (OCTAL)
000000 315 CALL 000200     get start addr
000003 353 XCHG
000004 315 CALL 000200     get end addr
000007 043 INX H
000010 353 XCHG
000011 176 MOV A,M         get byte
000012 315 CALL 000040     say it
000015 043 INX H           next byte
000016 174 MOV A,H         more?
000017 272 CMP D
000020 302 JNZ 000011      yes:loop
000023 175 MOV A,L
000024 273 CMP E
000025 302 JNZ 000011      yes;keep going
000030 303 JMP 000000      no: get new addr

Vocal Output of ACC (Octal)
000040 315 CALL 000160     Delay between bytes
000043 107 MOV B,A         save in B
000044 007 RLC             get m.s. digit
000045 007 RLC
000046 346 ANI 003
000050 315 CALL 000063     say it
000053 170 MOV A,B         get middle digit
000054 017 RRC
000055 017 RRC
000056 017 RRC
000057 315 CALL 000063     say it
000062 170 MOV A,B         l.s. digit
000063 346 ANI 007         SAY DIGIT
000065 207 ADD A           multiply by 4
000066 207 ADD A
000067 345 PUSH H          save everybody
000070 325 PUSH D
000071 365 PUSH PSW
000072 306 ADI 300         add in table base
000074 117 MOV C,A
000075 026 MVI D,000       table page to B
000077 012 LDAX B          now look up end addr of word
000100 127 MOV D,A         to DE
000101 003 INX B
000102 012 LDAX B
000103 137 MOV E,A
000104 003 INX B           beginning addr of word
000105 012 LDAX B          to HL
000106 147 MOV H,A
000107 003 INX B
000110 012 LDAX B
000111 157 MOV L,A
000112 176 MOV A,M         get word byte
000113 036 MVI C,010       8 bits
000115 017 RRC             shift in new bit
000116 323 OUT 013         out to speaker
000120 026 MVI D,002
000122 025 DCR D           delay
000123 302 JNZ 000122
000126 015 DCR C           more bits?
000127 302 JNZ 000115      loop for more
000132 043 INX H           next byte
000133 175 MOV A,L
000134 273 CMP E
```

```
000135 302 JNZ 000110          loop if not done
000140 174 MOV A,H
000141 272 CMP D
000142 302 JNZ 000110          more to say
000145 321 POP D               put 'em back
000146 321 POP D
000147 341 POP H
000150 311 RET                 go home


 DELAY 1/4 sec.

000160 001 LXI B,200000        set count
000163 015 DCR C
000164 302 JNZ 000163          inner loop
000167 005 DCR B
000170 302 JNZ 000163          keep going
000173 311 RET                 done


 KYBD TO HL (Octal)

000200 315 CALL 000204         keyboard to L
000203 145 MOV H,L             now it's H
000204 006 MVI B,003           3 digits
000206 056 MVI L,000           clean slate
000210 315 CALL 000230         get character
000213 346 ANI 007             throw away everything but the digit
000215 205 ADD L               add in old stuff
000216 157 MOV L,A             and save
000217 005 DCR B               more digits?
000220 310 RZ                  return when done
000221 207 ADD A               shift in new digit
000222 207 ADD A
000223 207 ADD A
000224 157 MOV L,A             save
000225 303 JMP 000214          get the next


   INPUT A CHARACTER TO ACC

000230 333 IN 000              get char
000232 267 ORA A               set flags
000233 362 JP 000230           loop if msb off
000236 365 PUSH PSW            save acc
000237 333 IN 000              wait for strobe bit to go away
000241 267 ORA A
000242 372 JM 000237
000245 315 CALL 000063         echo the digit (vocally, of course!)
000250 361 POP PSW             restore acc.
000251 311 RET                 done

  TABLE GIVES ADDRESSES OF WORDS IN MEMORY (END (HL),START (HL))

000300/002 216 001 000 003 256 002 217 004 344 003 257 006 023 004 345
000320/007 222 006 024 011 075 007 223 012 361 011 076 014 113 012 362

 DIGITIZED SPEECH
 "zero"
001000/374 001 000 200 001 340 377 000 220 067 000 000 360 077 000 300
001020/037 020 000 376 037 000 364 007 000 000 330 031 000 252 137 100
001040/000 370 177 000 310 137 001 003 300 376 001 040 152 005 070 000
001060/156 114 214 334 053 000 003 140 357 215 072 053 160 070 000 345
001100/276 000 107 153 312 003 340 356 016 210 267 250 230 020 377 057
001120/300 315 143 140 142 354 277 210 275 373 276 215 371 377 003 313
001140/273 023 063 362 377 003 376 077 010 033 361 177 201 377 015 354
001160/107 376 017 340 077 062 137 310 377 000 374 107 156 033 360 037
001200/200 237 374 151 043 376 023 373 263 077 154 304 177 300 037 374
001220/205 233 361 023 376 003 177 340 104 374 031 073 363 177 160 143
001240/376 210 073 343 317 170 102 276 021 177 370 077 340 214 377 206
001260/077 340 077 300 007 374 007 174 300 177 300 075 302 177 200 007
001300/374 017 060 007 370 017 360 200 377 001 300 003 376 007 174 300
001320/077 360 343 030 377 061 037 360 037 370 161 214 377 230 017 374
001340/307 374 340 030 377 071 037 360 207 071 003 107 376 347 076 340
001360/077 360 001 070 342 023 377 061 307 034 107 034 007 176 342 033
002000/377 340 307 034 301 201 037 170 300 035 372 231 107 346 300 207
002020/074 342 034 343 017 143 314 341 017 177 310 161 334 163 214 201
002040/207 077 376 061 306 201 217 207 017 076 370 301 017 143 170 340
002060/203 307 074 074 374 341 017 040 014 170 370 360 301 017 377 370
002100/007 160 007 037 170 000 307 203 077 234 003 170 300 303 001 007
002120/367 340 037 306 021 030 036 036 034 300 017 377 341 000 000 300
002140/207 001 000 216 203 017 002 000 360 301 001 300 340 000 074 030
002160/000 300 001 000 070 202 207 003 360 160 140 200 200 001 016 074
002200/200 203 003 000 170 040 000 000 000 016 016 003 000 000 014


 "One"
002217/360 007 000 000 177 100 377 377 300 374 017 300 017 370 377 303
002237/200 377 001 374 003 376 377 300 360 177 200 177 300 377 077 000
002257/377 017 340 017 370 177 000 360 377 001 374 001 377 001 300 376
002277/077 200 037 360 017 000 200 376 003 370 001 177 000 076 360 077
002317/360 037 370 301 363 201 077 360 177 300 007 317 007 077 362 303
002337/003 037 037 037 174 370 307 017 074 074 074 370 370 037 170 360
002357/360 370 360 174 074 340 301 303 343 340 174 074 016 017 017 307
002377/363 360 170 034 170 070 074 207 303 343 077 016 207 303 071 376
003017/376 363 170 370 160 270 307 203 076 017 017 034 034 347 170 300
003037/347 217 003 036 216 170 334 341 237 343 001 037 376 170 314 361
003057/317 037 017 170 374 361 303 003 177 076 174 314 361 343 217 077
003077/374 371 343 206 103 217 075 060 344 314 071 161 034 170 347 305
003117/021 177 356 314 370 200 173 157 004 161 343 206 000 160 300 374
003137/177 000 300 057 006 000 174 000 377 157 001 300 316 000 000 066
003157/200 374 023 300 200 300 000 003 176 000 370 003 300 000 077 000
003177/000 174 000 370 007 000 000 370 001 070 340 017 340 077 000 000
003217/340 007 000 340 007 300 377 001 300 301 017 000 200 017 001 245
003237/000 000 300 077 000 000 054 000 274 000 001 000 044 003 000 000


 "two"
003257/130 375 245 376 152 365 105 075 120 325 154 143 146 176 062 272
003277/032 236 256 343 133 131 265 270 364 275 255 106 050 067 331 315
003317/246 042 172 223 377 123 073 023 104 355 264 056 067 231 005 314
003337/250 052 052 073 233 210 015 262 021 260 213 031 111 156 047 273
003357/001 311 314 314 374 000 122 062 063 000 370 000 377 003 370 037
003377/360 173 001 373 013 370 037 170 001 371 037 370 037 270 001 371
004017/037 370 017 170 003 371 027 360 037 370 006 340 057 342 177 340
004037/033 210 377 200 377 001 337 100 374 017 374 017 340 016 304 377
004057/200 377 001 274 001 370 037 360 077 200 377 003 377 003 370 037
004077/370 177 000 377 003 360 077 340 277 040 376 017 310 077 040 257
004117/025 361 177 200 372 063 143 270 042 376 017 350 377 205 015 274
004137/000 377 007 370 077 342 012 170 000 376 037 340 377 001 160 342
004157/001 360 077 200 377 003 374 001 036 000 377 003 360 077 200 077
004177/200 007 300 377 001 376 003 370 017 300 007 360 377 000 377 007
004217/374 001 360 007 360 377 000 377 003 000 000 300 007 360 377 000
004237/076 000 360 002 300 037 300 377 001 004 000 140 000 002 177 000
004257/377 000 074 000 100 000 000 167 000 370 003 002 002 140 000 002
004277/107 300 077 000 303 103 000 300 010 002 100 003 016 060 203 140
004317/012 010 200 003 000 014 000 300 000 000 042 000 010 020 000 300
004337/001 000 002 000 000 042


 "three"
004345/300 200 335 003 004 000 002 276 000 314 241 140 361 005 101 016
004365/302 374 303 177 300 171 332 203 354 037 340 003 174 177 020 174
005005/170 076 056 010 360 303 042 355 062 013 046 260 014 000 210 017
005025/200 377 162 034 276 200 377 301 314 237 361 107 314 377 071 300
005045/377 200 147 234 377 001 360 277 360 314 363 177 002 374 017 074
005065/302 374 237 000 377 003 236 241 376 207 200 377 000 317 010 377
005105/001 340 077 200 043 304 177 303 370 077 330 003 340 077 001 374
005125/017 360 033 361 357 304 374 033 330 033 371 177 040 177 047 357
005145/233 310 037 100 376 103 237 035 354 177 140 077 271 117 231 344
```

```
005165/177 230 377 145 033 233 304 177 201 375 046 333 066 201 377 003
005205/335 067 331 234 000 376 017 264 377 046 301 026 370 177 201 353
005225/273 011 054 200 374 007 154 237 155 301 002 310 377 201 362 155
005245/027 260 002 372 177 200 374 377 200 334 003 373 277 040 371 027
005265/000 261 002 371 277 000 332 247 001 120 013 364 377 021 354 315
005305/322 002 150 201 376 077 304 077 001 175 023 126 200 377 017 200
005325/055 100 133 300 254 001 372 277 000 225 000 201 300 254 000 370
005345/027 000 051 000 012 000 120 134 000 374 003 007 000 004 000 002
005365/110 012 242 001 100 013 000 312 000 020 000 200 002 004 000 003
006005/000 022 000 000 000 001 000 000 000 001 000 001 010 000 200
```

"four"

```
006024/101 004 000 000 161 300 011 021 004 160 120 120 020 063 000 200
006044/000 000 000 104 024 121 100 100 012 001 102 230 010 140 101 120
006064/061 101 014 063 300 200 222 043 042 000 052 014 205 000 002 004
006104/000 150 000 071 063 002 004 000 010 061 002 060 003 010 000 142
006124/310 001 022 041 011 210 134 231 070 234 031 200 031 053 001 116
006144/134 074 161 024 116 060 246 234 060 061 006 047 110 000 116 000
006164/200 000 300 007 020 000 341 141 034 214 003 000 334 377 300 377
006204/001 006 376 340 377 340 037 374 017 077 370 077 370 017 376 303
006224/007 377 007 077 300 037 360 201 377 203 037 340 017 370 200 377
006244/201 037 370 017 370 300 337 201 037 374 037 370 200 237 203 037
006264/374 037 370 201 037 007 077 370 003 340 003 077 014 176 360 017
006304/300 017 370 141 360 203 077 000 176 300 017 203 037 374 001 370
006324/007 176 030 374 340 037 200 177 340 007 301 017 374 001 374 017
006344/374 160 370 201 077 000 374 003 177 030 374 340 017 340 347 201
006364/037 016 176 360 007 360 343 203 037 016 176 360 017 370 343 003
007004/037 036 176 360 037 364 207 007 076 074 176 340 077 370 017 037
007024/374 070 374 360 370 360 073 170 340 343 340 203 377 303 177 340
007044/017 174 014 076 340 017 343 203 341 203 317 341 007 370 170 076
007064/016 377 340 077 374 007 037 370 007 356 360 340 037 376 203 177
007104/370 207 377 230 207 377 370 217 361 201 037 370 016 347 301 037
007124/376 001 036 177 034 016 360 200 017 360 000 177 300 017 006 000
007144/300 000 000 024 000 140 160 304 161 207 063 300 001 200 141 000
007164/340 000 300 000 000 000 006 100 060 002 360 301 017 007 036 010
007204/014 030 100 030 140 140 201 203 001 007 014 000 014 040 030
```

"five"

```
007223/030 000 010 000 000 000 010 002 004 000 000 011 021 041 101 121
007243/040 024 120 100 010 000 001 100 204 020 360 020 004 130 305 006
007263/142 200 100 040 004 204 010 163 104 141 004 070 060 000 130 360
007303/007 204 000 002 004 000 200 007 017 000 360 007 376 340 017 360
007323/300 017 376 041 374 340 341 301 037 376 361 374 343 303 203 037
007343/174 340 360 303 207 007 037 074 300 341 203 007 037 036 360 360
007363/207 007 037 176 170 300 343 037 037 176 374 341 021 207 147 074
010003/374 361 207 307 034 236 361 370 303 034 036 163 070 306 343 037
010023/347 160 330 343 061 036 177 234 307 343 036 307 361 347 163 074
010043/036 363 074 016 036 217 321 341 260 307 261 360 170 040 016 307
010063/075 036 030 217 167 342 361 230 343 021 343 361 207 170 074 360
010103/070 342 170 376 060 036 017 074 036 170 034 037 017 016 007 036
010123/017 070 036 037 167 216 047 036 047 354 170 376 174 070 276 170
010143/334 060 307 235 347 230 343 217 347 017 174 374 171 014 360 370
010163/363 301 201 160 374 074 343 340 360 363 307 203 160 374 070 162
010203/340 361 343 317 203 361 361 363 340 300 207 217 077 037 374 234
010223/077 076 016 362 360 361 363 301 201 361 343 143 306 016 034 070
010243/170 360 303 306 211 201 035 036 374 360 203 000 202 007 017 174
010263/003 077 376 300 142 006 076 360 301 203 007 016 016 374 170 046
010303/040 350 071 360 203 037 203 077 072 160 340 017 366 003 176 100
010323/036 370 003 027 260 001 037 000 000 160 360 001 234 001 000 060
010343/041 162 344 101 300 077 000 003 300 003 020 004 000 000 140 000
010363/041 001 001 000 000 200 022 045 106 100 200 200 004 000 000 010
011003/100 100 030 000 000 000 024 204 000 000 100 002 240 022 023 002
011023/001 042 200 300 202 000 042 011 140 122 000 040 040 003 046 104
011043/110 001 047 011 024 065 022 231 240 300 244 040 103 052 022 000
011063/032 141 006 214 200 021 000 014 020 020 200
```

"six"

```
011076/200 000 004 002 000 020 000 010 031 200 010 040 042 000 002 000
011116/040 000 110 010 002 052 120 212 000 150 004 132 162 012 312 110
011136/006 246 040 001 120 201 105 200 051 351 252 214 051 042 104 056
011156/302 300 150 175 105 204 011 027 231 220 011 140 370 225 323 130
011176/331 305 001 115 004 136 156 316 147 122 166 157 065 107 220 264
011216/051 103 352 106 376 120 120 141 142 125 037 167 052 304 225 206
011236/136 234 223 234 353 115 300 252 352 015 173 171 214 102 372 125
011256/065 336 312 342 111 344 013 237 214 346 064 166 323 103 174 350
011276/272 124 065 120 073 164 232 132 351 052 144 141 263 370 251 336
011316/234 334 213 350 000 237 300 177 300 077 300 033 320 037 370 007
011336/376 260 221 375 300 157 360 207 015 354 007 176 201 037 334 304
011356/166 340 037 370 007 223 330 117 364 003 376 200 333 344 027 370
011376/007 377 001 066 331 077 360 117 340 337 364 337 044 377 023 371
012016/146 233 364 002 176 000 157 300 066 262 100 000 000 000 000 000
012036/000 000 000 000 000 000 000 000 000 000 040 000 070 000 002 140 000
012056/000 000 003 000 000 010 000 000 000 000 000 000 000 000 000 000 000
012076/000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000
012116/000 001 000 000 000 000 000 000 000 000 000 000 000 002 000 000 000
012136/000 000 000 000 000 000 000 000 000 000 213 061 342 361 067 076
012156/147 316 234 161 317 270 063 146 346 063 167 142 304 134 314 146
012176/321 116 073 163 314 314 101 011 221 031 153 134 226 356 156 213
012216/214 271 311 352 251 342 252 113 143 213 172 361 342 172 013 167
012236/364 075 007 145 264 346 050 117 002 354 312 125 254 040 024 246
012256/206 110 316 232 176 130 244 042 112 012 021 042 205 372 163 245
012276/212 025 150 046 326 207 251 234 205 200 025 007 022 210 314 026
012316/323 053 052 351 202 212 020 061 054 124 134 101 105 040 004 020
012336/051 045 006 100 214 260 163 001 001 042 000 000 030 000 000 000
012356/025 002 210 010
```

"seven"

```
012356/025 002 210 010 120 000 200 000 000 104 040 202 100 200 240 200
012376/024 015 005 241 302 060 012 044 124 042 327 212 304 126 065 002
013016/115 010 034 050 342 015 047 134 024 025 112 110 252 236 052 022
013036/260 005 145 122 161 103 254 057 202 264 377 225 005 001 100 242
013056/334 375 125 030 316 050 067 125 007 034 331 051 122 110 207 052
013076/030 112 342 022 072 047 342 124 332 005 123 020 130 137 254 124
013116/356 176 277 165 351 070 121 037 050 247 152 005 242 253 152 041
013136/360 007 276 070 074 240 003 372 001 040 176 340 117 300 037 374
013156/070 260 047 376 004 176 370 071 340 047 376 304 017 374 070 360
013176/047 157 300 017 374 070 360 143 157 304 017 376 160 340 307 236
013216/200 077 376 160 340 317 037 020 177 374 341 200 037 177 106 374
013236/370 203 003 176 374 071 361 303 017 034 360 343 316 200 007 177
013256/340 300 037 366 007 176 370 003 017 374 303 377 340 377 363 203
013276/007 376 177 330 340 377 023 035 016 370 377 102 200 163 216 210
013316/341 240 305 007 000 200 007 012 101 153 100 314 003 006 200 102
013336/040 200 023 167 376 104 007 045 002 200 304 041 117 005 061 016
013356/035 011 370 001 377 105 007 274 067 013 170 300 077 147 016 176
013376/171 040 070 360 317 077 014 376 370 140 034 370 343 037 160 360
014016/303 001 070 340 317 077 000 016 270 171 020 017 374 303 007 200
014036/037 360 107 014 074 360 357 034 000 076 000 000 072 000 340 002
014056/320 377 000 000 000 000 000 000 340 002 260 177 000 000 000 000
014076/000 000 040 000 000 177 000 000 000 000 150 000 275 001
```

RELOCATER FOR SPEECH PROGRAM
Load Vocal Memory Dump at desired location (low address must be zero), load relocater at ØØØØØØ. Change byte ØØØØØ1 to page number of speech program then run relocater.

```
000000 046 MVI H,030          set to page of VMD program
000002 021 LXI D,000021       fix table
000005 032 LDAX D             get fix addr
000006 157 MOV L,A
000007 176 MOV A,M            change memory contents
000010 204 ADD H
000011 167 MOV M,A
000012 023 INX D              next fix
```

```
000013 032 LDAX D
000014 267 ORA A
000015 302 JNZ 000006          loop for more
000020 166 HLT                 no more

000021/002 006 014 022 027 032 042 052 061 076 125 131 137 144 166 172
000041/202 212 227 235 244 247 300 302 304 306 310 312 314 316 320 322
000061/324 326 330 332 334 336 000
```

SPEECH INPUT PROGRAM

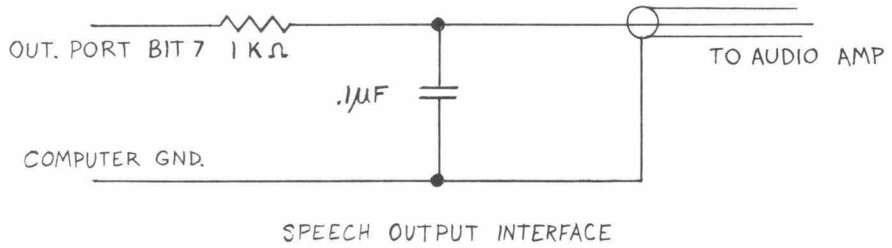
```
000000 041 LXI H,000100        start addr (after DELAY)
000003 016 MVI C,010           8 bits
000005 333 IN 010              get a bit
000007 207 ADD A               now it's in carry
000010 170 MOV A,B             get old stuff
000011 037 RAR                 shift in the new
000012 107 MOV B,A             and save
000013 315 CALL 000070         delay
000016 015 DCR C               more bits?
000017 302 JNZ 000005          loop
000022 160 MOV M,B             save
000023 043 INX H               next location
000024 174 MOV A,H             done? (24K)
000025 376 CPI 140             no, continue
000027 302 JNZ 000003          done.
000032 166 HLT
```

SPEECH OUTPUT PROGRAM

```
000040 041 LXI H,000100        start addr
000043 016 MVI C,010           8 bits
000045 176 MOV A,M             get a byte
000046 017 RRC                 shift in a bit
000047 323 OUT 013             send it out
000051 315 CALL 000070         delay
000054 015 DCR C               count bit
000055 302 JNZ 000046          loop
000060 043 INX H               next location
000061 174 MOV A,H             or are we done?
000062 376 CPI 140             back for more
000064 302 JNZ 000043          that's all
000067 166 HLT
```

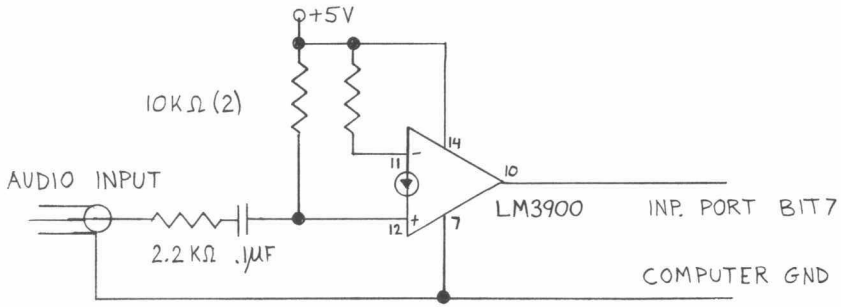
DELAY SUBROUTINE

```
000070 026 MVI D,020           time constant
000072 025 DCR D
000073 302 JNZ 000072          keep looping
000076 311 RET                 go home
```



SPEECH OUTPUT INTERFACE

FIG. 1



SPEECH INPUT INTERFACE

FIG. 2

COSMAC Elf

Experience and Comparisons

Dear Doctor Dobb: *Received: 78 March 24*

Your magazine is always a welcome sight whenever it does arrive. Keep it up.

From time to time you and some others have published bits about the 1802 micro and the 'Elf' computer. Although quite underdeveloped, this seems to be the lowest cost way for a youngster or beginner to start.

Two days a week I have been taking my Altair to a private school to expose the youngsters to micros. One student became so excited that he wanted to build a micro from plans in a very old issue of *Popular Electronics.* I explained to him that the later issues of said magazine have a series of articles on a much easier, lower cost micro-computer: the 'Cosmac Elf'. He came to the conclusion that it is really cheaper to buy a complete kit rather than one piece at a time.

Presently there are, as far as I know, just two good low-cost versions of the 'Elf'. The 'Elf II' from Netronics R&D in New Milford, CT, or the 'Super Elf' from Quest Electronics in California.

Which is better? Well, Timothy bought an 'Elf II' for a little over $100 and I got the 'Super Elf' for just about the same price. They are both good. Almost equal, but not quite.

First, the common points: Both are about the lowest cost for a good micro-training tool. Both use a single P.C. board with the 1802 processor, 256 bytes RAM and a nice hexadecimal keypad and led display.

The power supply is a single 5-volt regulator on a heat sink fed by four rectifiers and a 1000 Mfd capacitor. The step-down transformer is external so that no high-voltage is present on the circuit board. You must order the transformer separately for the 'Elf II'. Don't be foolish and try to run it from a 6 volt transformer or a bell transformer. The power supply is very weak and you must have the right transformer for good operation. About 8 to 10 volts AC is required. 12 volts AC is too much and will overheat the already hard-working regulator. 6 volts is not enough.

The reason is the hex display. It uses as much current as the rest of the computer. I have modified my 'Super Elf' so that the average display current is reduced by giving the led display pulsating D.C.

Both come with good assembly instructions, but some prior knowledge of electronics is helpful. Although Timothy has not ever built anything like this before, he did quite well. The main problem was trying to make modifications before finishing the kit. He now has it up and running and inside a very nice formica cabinet his father made.

Both dealers promise additional devices for their 'Elf' kits. Adding an I/O and 4K of memory will amost double your micro-computer investment. Still, it is about the only way you can get going for less than two hundred dollars. Now about the differences: The 'Elf II' has a larger board and has lots of room to expand, while 'Super Elf' is very crowded on a smaller board. One pad is provided for a standard 44 pin connector. If you wish to expand you must directly wire into the 50 terminals on the upper edge of the P.C. board.

The power supply transformer is included in the price of the 'Super Elf', as is a simple monitor on ROM. However, it is very limited. You might like to have an address display. It is available as an option for the 'Super'. No provision on the 'Elf II'. The hex key pads are almost identical on both, but Timothy and I agree that the arrangements on the 'Super Elf' is more logical. So he rewired his keypad so that '0' is in the upper left and 'F' in the lower right. 'Super' includes an audio amplifier and speaker for the 'Q' output. First thing I did was to try a music program which was not very loud. Timothy has built his own 'Q' amp and used a larger speaker for much greater volume and better bass.

My 'Super' came with large '500' type seven segment displays. 'Elf II' came with similar but smaller displays. The 'Elf II' can not single step a program. The 'Super' has provision for either single step or running a program slowly as an aid in program debugging.

To sum it up: 'Super Elf' appears to be the winner, although 'Elf II' may be easier to expand. Both are a good buy. What is needed for both kits is a modification to use a 1702A PROM.

What the 'Elf' needs is a good prom program to read data from a Kansas City cassette and write into the 'Elf" memory. Tim uses the cassette program published in *Dr. Dobb's.* He first keys in the Read Cassette Program, then loads in his favorite programs via the cassette. I knew that it could be done, but the first time I saw him do it I was spellbound. His entire system cost less than the cassette interface for my Altair. This is a good example of the cost-effectiveness of micro-processors. The 1802 doesn't need a UART or fancy interface to load data into memory. Just a few simple parts and a short program!

The 1802 needs more software. I am going to try my hand at writing a Tiny Basic for the 1802. If it turns out OK then I will make it available to other hobbyists on cassette tape. I am using the Altair to cross-assemble to the 1802.

Maybe it sounds like I am down on Netronics. No, not really. Be aware that what you get is not exactly what you see in the photo. The 86 pin connectors are extra at $5.70 each. And the four buttons on the far right may be replaced by toggel switches. And not all IC's come with sockets. Both kits fail to use the required 22K pull-up resistors needed for TTL to CMOS. One of our units gave some trouble because of this.

Yours truly,
John B. Palmer

P.O. Box 27462
Tuscon, AZ 85726

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 29 | October 1978 | Volume 3, Issue 9

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

LEDIP: A KIM/6502 Line Editor *Kiumi Akingbehin* 4
An expandable program for creating text and source code

Quick and Dirty Routines for BASIC *Dennis Allison* 13
Subroutines to implement some useful functions

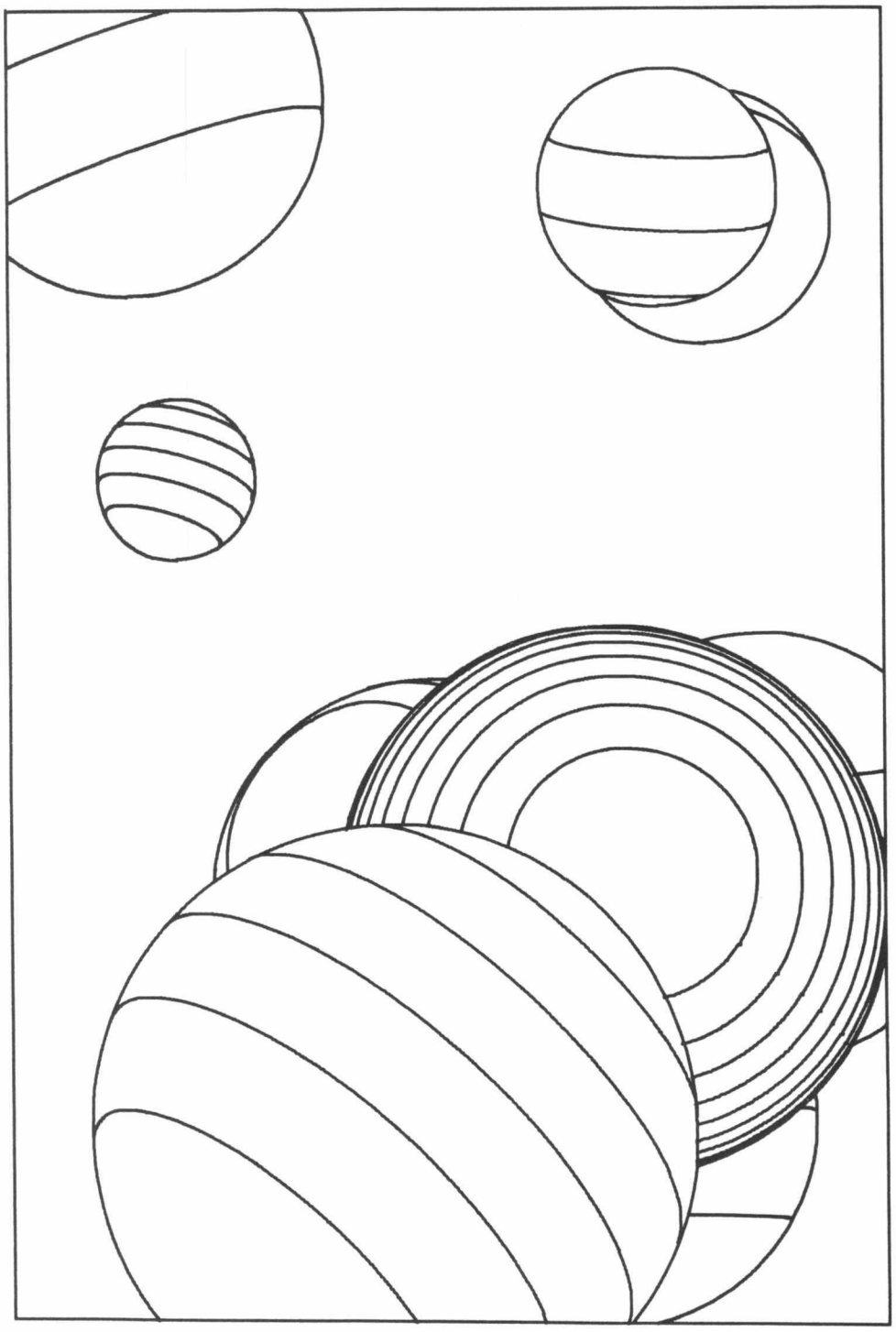
POW: An 8080 'Processor of Words' *Herman Watson* 18

Whatis: An 8080 Expression Evaluator *Mike Gabrielson* 34
A program for evaluating arbitrary arithmetic and logical expressions

HARDWARE

The Digicast™ Project *Jim Warren* 16
A broadcast-based information system for the public

Data-Boy™ Speech Processor *James Anderson* 30
Description of a new, low-cost speech interface



© David Dameron 1979

EDITING TINY BASIC

Richard Greenlaw *Received: 78 July 24*

Here is another extension for Lichen Wang's Palo Alto Tiny BASIC. I have modified the GETLN routine to allow editing of existing BASIC program lines.

To edit a line, type the line number as usual. After that each control-L (the load key on a Sol) will input successive characters from the original line in the program. You can backspace over them or blend them with new text as you like. The line can be terminated by carriage return from the keyboard or from the original line if you go that far with control-L.

If the line number doesn't exist a line delete is simulated.

I typed a source from the *Interface Age* version (12/76) so I added this by reassembly, not patching. You will note that I have changed some of the control codes, such as line delete, and have replaced the RST single byte calls with normal calls so I can run at 2A00H under Micropolis MDOS.

```
0010 GETLN  CALL  OUTC      ;OUTPUT FROM A
0020        XRA   A         ;ZERO
0030        STA   EDFLG     ;INDICATE EDIT POINTER NOT SET
0040        LXI   D,BUFFER
0050 GL1    CALL  CHKIO
0060        JZ    GL1
0070        CPI   0CH       ;CTL-L (LOAD KEY ON SOL)
0080        CZ    GLED      ;GET CHAR FROM ORIGINAL LINE
0090        CPI   7FH
0100        JZ    GL3
0110        CALL  OUTC
0120        CPI   0AH
0130        JZ    GL1
0140        ORA   A
0150        JZ    GL1
0160        CPI   18H       ;CTL-X IS LINE DELETE
0170        JZ    GL4
0180        STAX  D
0190        INX   D
0200        CPI   CR
0210        RZ
0220        MOV   A,E
0230        CPI   0FFH&BUFEND
0240        JNZ   GL1
0250 GL3    MOV   A,E
0260        CPI   0FFH&BUFEND
0270        JZ    GL4
0280        DCX   D
0290        MVI   A,5FH
0300        CALL  OUTC
0310        JMP   GL1
0320 GLED   PUSH  H         ;EDITOR
0330        PUSH  D
0340        PUSH  B
0350        STAX  D         ;DELIMIT PARTIAL INPUT
0360        LDA   EDFLG
0370        ORA   A
0380        JNZ   GLEDA     ;IF PTR INTO OLD LINE SET
0390 *FIND LINE WHOSE NBR BEGINS INPUT LINE
0400        LXI   D,BUFFER
0410        CALL  TSTNUM    ;HL=BINARY LINE # OR 0
0420        CALL  FNDLN     ;DE->LINE IF Z SET
0430        MVI   A,18H     ;CTL-X LINE DELETE
0440        JNZ   GLEDX     ;IF NOT FOUND CANCEL LINE
0450        INX   D         ;SKIP BINARY LINE NBR
0460        INX   D
0470        XCHG
0480        SHLD  EDPTR     ;INITIALIZE EDIT POINTER
0490        MVI   A,1
0500        STA   EDFLG     ;INDICATE EDIT PTR SET
0510 GLEDA  LHLD  EDPTR     ;EDIT POINTER
0520        MOV   A,M       ;FROM ORIGINAL LINE
0530        INX   H
0540        SHLD  EDPTR
0550 GLEDX  POP   B
0560        POP   D
0570        POP   H
0580        RET
0590 GL4    CALL  CRLF
0600        MVI   A,5EH
0610        JMP   GETLN
5450 LSTRO  EQU   $
5455 EDFLG  DS    1         ;EDIT ADDRESS SET FLAG
5460 EDPTR  DS    2         ;EDIT POINTER INTO ORIGINAL LINE
```

A NEAT TRICK

Dennis Allison *Received: 78 Aug 11*

This trick using the Decimal Adjust instruction generates the ASCII character corresponding to the hex digit in register A. I first saw it in the Intel MDS monitor. Ken Burgett tells me it was due to Bill Byerley. Both are with the Intel software group. Assuming a number between 0 and 15 is in the A register,

8080	*6800*
ACI 090H	ADD A, #$90
DAA	DAA
ACI 040H	ADC A, #$40
DAA	DAA

gives the characters 0 through 9 and A through F in ASCII.

Just how it works is left as an exercise for the reader.

16 BITS FROM CROMEMCO?

Reports are that Cromemco would be developing a new CPU around Zilog's new Z-8000 16 bit microprocessor which will be sampled this year. A call to Cromemco netted a sincere "no comment" with the additional remark that Cromemco was "looking to develop a family of compatible machines." We'll be watching. –TRW

PUSHING 8080 REGISTERS

Mike Gabrielson *Received: 78 July 27*

It recently became necessary for me to devise reentrant subroutines for pushing and pulling all registers on the 8080. The push routine turned out to be surprisingly difficult. Can your readers construct a more elegant version?

```
1             ;*************************************************
2             ; Pusher - reentrant routine to push all registers *
3             ; Mike Gabrielson   7/24/78                       *
4             ;*************************************************
5
6  0000 E3    Pusher: XTHL            ;swap HL for return address
7  0001 D5            PUSH    D      ;save D, E
8  0002 C5            PUSH    B      ;save B, C
9  0003 F5            PUSH    PSW    ;save A, PSW
10 0004 E5            PUSH    H      ;put return address on top of stack
11 0005 210800        LXI     H,8    ;get offset to stacked H, L
12 0008 39            DAD     SP     ;HL := address of stacked H, L
13 0009 D5            PUSH    D      ;now we need DE, so save an extra copy
14 000A 5E            MOV     E,M    ;E := stacked L
15 000B 23            INX     H      ;point to stacked H
16 000C 56            MOV     D,M    ;D := stacked H
17 000D EB            XCHG           ;restore HL
18 000E D1            POP     D      ;restore DE from extra copy
19 000F C9            RET
20
21            ;It's unlikely, but if the stack is situated so that it crosses
22            ;between 0 and FFFF, then the DAD instruction may destroy the
23            ;caller's carry bit.  In that case, the following longer but
24            ;foolproof version is necessary:
25
26            ;       XTHL
27            ;       PUSH    D
28            ;       PUSH    B
29            ;       PUSH    PSW
30            ;       PUSH    H
31            ;       PUSH    PSW     ;save carry before DAD
32            ;       LXI     H,10    ;now the offset's bigger, too
33            ;       DAD     SP
34            ;       POP     PSW     ;restore carry
35            ;       PUSH    D
36            ;       MOV     E,M
37            ;       INX     H
38            ;       MOV     D,M
39            ;       XCHG
40            ;       POP     D
41            ;       RET
42
43
44            ;*************************************************
45            ; Puller - reentrant routine to pop all registers *
46            ;         pushed with Pusher                      *
47            ; Mike Gabrielson   7/24/78                       *
48            ;*************************************************
49
50 0010 E3    Puller: XTHL            ;swap junk for return address
51 0011 F1            POP     PSW    ;ignore junk
52 0012 F1            POP     PSW    ;restore A, PSW
53 0013 C1            POP     B      ;restore B, C
54 0014 D1            POP     D      ;restore D, E
55 0015 E3            XTHL           ;restore H, L and stack return address
56 0016 C9            RET
```

LEDIP
A KIM/6502 Text Editor

BY KIUMI AKINGBEHIN
Department of Mathematics
Wayne State University
Detroit, MI 48202

LEDIP (an acronym for *Line EDItor Program*) is a general purpose line-oriented text editor program for 6502-based systems. LEDIP can be used for such purposes as writing letters, preparing texts, and generating source programs.

LEDIP is designed to be memory-efficient and easy to use. Residing in about 1K bytes of memory, LEDIP uses an efficient data structure to minimize the memory occupied by the user's text. LEDIP performs memory compressions and expansions as needed after each line of text is entered. Not a single byte of memory is wasted. In addition, LEDIP allows the user to select the location in memory where the text is stored. LEDIP's small memory requirements make it ideal for memory conscious users. With LEDIP, a reasonable amount of text can be edited in a system with as small as 2K bytes of memory.

Running LEDIP

LEDIP Version K4 (assembly listing shown), runs on KIM systems with at least 1.1K bytes of RAM starting at location 2000 hex and going upwards. Since LEDIP is a text editor and not a memory editor (compare EDITHA/SWEETS, *DDJ* Vol.3, Issue 5, May 78), and I/O device such as a teletype is also needed. Readers with such a configuration may directly key in the object code and enter LEDIP thru location 2000 hex using the G command. LEDIP should respond with the question, "STARTING ADDRESS?". This is the cold entry point; warm entry point is at location 203C hex. Version K4 with the changes indicated in parenthesis will also run on TIM/DEMON systems. Readers who don't feel like keying in a 1.1K object code can obtain paper tape or KIM cassette of LEDIP from the 6502 Program Exchange, 2920 Moana, Reno, NV 89509. Include a $2.50 duplication/distribution fee. Versions of LEDIP for other 6502-based systems including VIM (Synertek's new 6502-based SBC) are also available from The 6502 Program Exchange. JOLT users should note that the TEXT command supplies the rub-outs required by the JOLT resident assembler.

Using LEDIP

LEDIP starts by requesting a starting address for the text from you. Type a four-digit hexadecimal location. Your text will occupy this location and subsequent memory locations. Be sure to specify usable RAM. LEDIP uses 18 contiguous bytes near the top of page zero to store variables and constants pertaining to the text being edited. In addition, LEDIP resides in about 1K bytes of memory. These locations should be reserved for LEDIP's use and should not be used for any other purpose. The FILE command can be used to find out what these locations are. Once a valid starting address is given, LEDIP does some initialization and responds with the prompt character, a slash. A line number or command can now be typed.

A line number can be any four-digit decimal number between 0000 and 9999. Leading zeroes must be included. If a line number is typed after the prompt character, LEDIP automatically goes into the edit mode, types a space, and waits for a line of text to be entered. A line can be of any length between 1 and 252 characters. Any upper or lower case ASCII character can be entered. Control codes and other special codes can also be entered. All control codes, with the exception of the backspace (control H), are stored as received. A backspace deletes the last character entered. Carriage-returns are not allowed within a line. A carriage-return terminates a line. Text lines are modified, replaced, deleted, or inserted using line numbers in a manner similar to BASIC. Note that this technique makes edit-mode commands like DELETE, REPLACE, INSERT, etc. unnecessary.

To add a line of text, type a new line number and then type in the text. To insert a line of text between two existing lines of text, type a line number between the two current line numbers and then enter the text. For instance, to enter a line of text between lines 0022 and 0029, type 0024 and then type the new text. LEDIP will do the memory shifting and manipulations necessary, and will insert the new line between the two current lines. To delete a line, type the line number and a carriage-return. To replace or modify a line, type the line number and then type the new text. To create a blank line, type the line number, at least one space, and then type a carriage-return.

If a command is typed after the prompt, LEDIP automatically goes into the command mode. LEDIP recognizes the following five commands:

LIST – lists the entire text with line numbers
TEXT – lists the entire text without line numbers
FILE – states the block of memory currently occupied by the text
EXIT – returns control to the system monitor program (if present)
CLEAR – clears current workfile and requests location for new text

The FILE command states three blocks of memory: a block of 18 bytes used by LEDIP on zero page, a block of memory occupied by LEDIP, and a block occupied by the user's text. The LIST and TEXT commands can be terminated at any time by using the hardware interrupt or reset and re-entering LEDIP through the warm start. LEDIP should always be

entered through the warm start if the current text is to be preserved. The EXIT command leaves the monitor program counter pointing to the warm start; hence only a G need be typed in most cases to re-enter LEDIP. An accidental CLEAR initiation can be corrected by an interrupt and a jump to the warm start.

LEDIP texts can be saved on tape in two formats for future use: an ASCII format and a hexadecimal format. To save a text in ASCII format, type TEXT, start the paper tape punch or cassette recorder, and then type a carriage return. ASCII formatted type cannot be reloaded into LEDIP for future editing. If future editing is desired, the text should be saved in hexadecimal format. To save a text in hexadecimal format, type FILE. LEDIP will define three memory blocks (e.g. 00D1-00E2, 2000-249D, 0100-01C4). Now type EXIT to return to your system monitor program. The monitor can now be used to save and reload the data contained in the first and third memory blocks. When loading your text thus, LEDIP should be entered through the warm start.

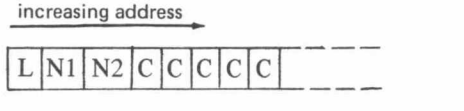
LEDIP checks the validity of commands, line numbers, line lengths, and continually performs read-after-write verifications. An error will result in one of the following error messages:

M! nonexistent memory or memory overflow
C! invalid command or line number
H! improper hex number
L! line too long

In the case of invalid four-letter commands, LEDIP defaults and executes the command whose first letter matches that of the invalid command.

A Brief Look Inside LEDIP

In keeping with the objective of a memory-efficient text editor program, LEDIP uses a sequential linear list (contiguous memory block) of variable length records to store the text. While a linked list or "table of line pointers" approach would have resulted in less code, it was decided that memory usage should be given priority over code reduction in the kind of environment in which LEDIP is likely to be used. The decision to use variable rather than fixed length records is based on the same consideration. Zero page locations STAD (starting address) always points to the top of the list and LOCC (location counter) always points to the bottom of the list. HEXBU (hexadecimal buffer) is invariably used to walk through the list. Each record (line of text) consists of three fields as shown in figure 1. LEDIP makes conservative use of the stack (page one) and only uses 18 bytes on page zero. These two pages are therefore largely available to the user.



L – length of line (one byte)
N1 – line number low order byte
N2 – line number high order byte
CCCC . . . – ASCII characters (variable length)

Figure 1: LEDIP data format.

Since the text list contains no absolute addresses or links, LEDIP is essentially text-relocatable. In fact, the block memory move subroutines in LEDIP can be used to move the text around in memory. Only STAD and LOCC need be changed whenever the text is relocated.

The other main consideration in writing LEDIP was to write an easy-to-use text editor. To achieve this goal, three decisions were made; viz. LEDIP shall be line oriented and not string oriented, line numbers shall be used for all edit-mode operations, non edit-mode commands and error messages shall be kept to one easily remembered minimum. The apparent simplicity with which line numbers are used to edit text lines obscures the actual processes which go on inside LEDIP during edit operations. The flowchart (figure 2) gives a clearer picture of these operations and the routines which are invoked by each. This flowchchart is roughly the second level in a four level top-down flowchart development of LEDIP.

LEDIP readily lends itself to modifications and extensions. Readers who wish to implement additional commands will find that the routines necessary for most additional commands (edit and non-edit) are already in the program. It should be noted that LEDIP does not use any command tables. Three NOP's have been included in the command handler (CMHD) to facilitate this. These NOP's will have to be replaced by an appropriate jump to the code extension. For instance, implementing a single line or line number range LIST only requires changing the contents of STAD and LOCC and then invoking the already existing LIST routines. LEDIP features several useful subroutines which are callable by other programs. These subroutines include block memory moves, ASCII conversion, hexadecimal and decimal character validation, save and restore register, and other routines. Zero page locations defined at the beginning of the program are used to pass parameters to and from these subroutines.

Since the CRLF, SPACE, and type-a-byte subroutines are as easily accessible as the standard read-a-character and type-a-character subroutines in most resident operating (monitor) systems, LEDIP directly calls all five I/O subroutines. All I/O calls flow thru a series of jumps near the end of LEDIP. Hence only ten locations need be changed to implement LEDIP on systems with different I/O configurations. LEDIP saves and restores all registers during I/O calls. Readers writing their I/O subroutines should remember to include proper delay for the CRLF as may be required by the console device. Readers who wish to add pagination to LEDIP listings should note that one inch top and bottom margins on the standard teletype requires 12 blank lines after every 54 text lines.

LEDIP does not feature a software BREAK test since the hardware interrupt or reset can be used to terminate LEDIP listings at any point. KIM users who wish to add a break text would have to poll the 6530 PIA data register at location 1740 hex. TIM users should poll location 6E02 hex. Since all I/O operations flow thru the restore register (RESR) routine, a good place to insert the break test is at the end of the RESR routine. Three NOP's have been included to facilitate this. In implementing a break test, care should be taken to restore the stack and to restore registers destroyed by the break routine. Since LEDIP preserves the syntax of the input text lines, readers who are interested in language translation will find LEDIP a useful basis for the development of an interactive compiling or interpreting language translator.

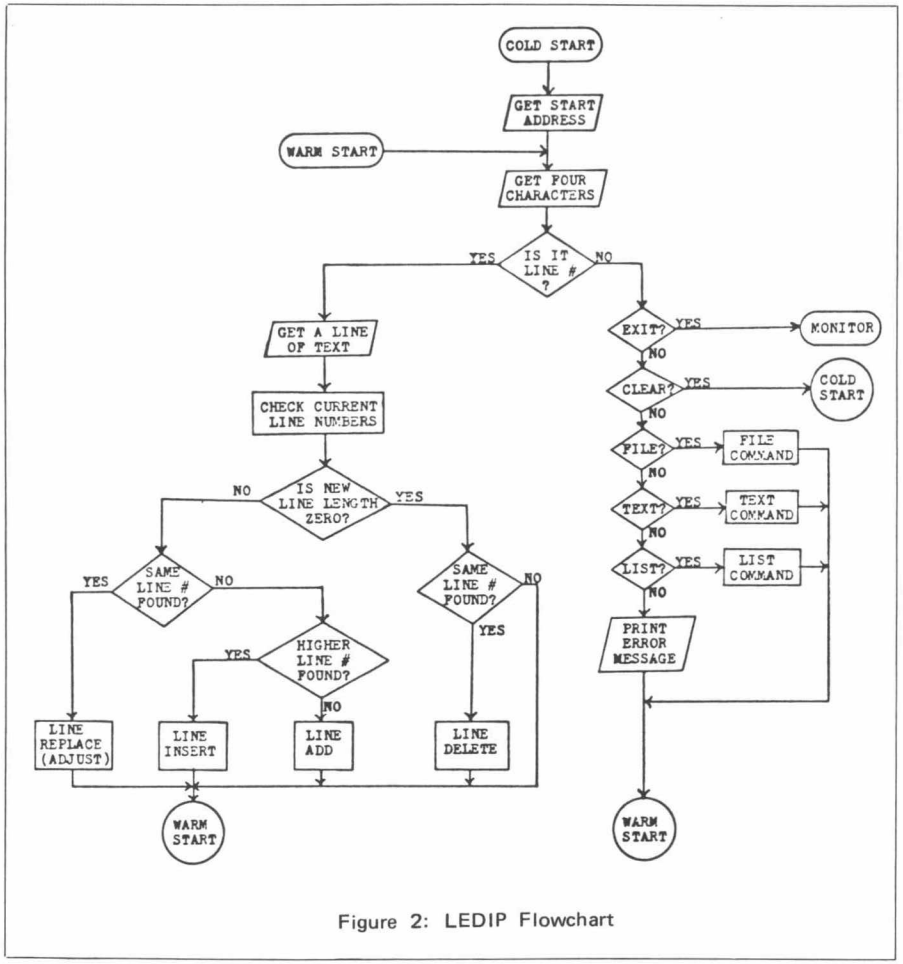


Figure 2: LEDIP Flowchart

```
CARD # LOC   CODE      CARD
    1                ;
    2                ;
    3                ;***********************************************;
    4                ;***********************************************;
    5                ;**                                           **;
    6                ;**     LEDIP (LINE EDITOR PROGAM)            **;
    7                ;**     VERSION K4                            **;
    8                ;**     COPYRIGHT 1978 BY KIUMI AKINGBEHIN    **;
    9                ;**                                           **;
   10                ;***********************************************;
   11                ;***********************************************;
   12                ;
   13                ;
   14                ; ZERO PAGE SCRATCHPAD MEMORY
   15                ;
   16                ;
   17                ASCBUM  = $E2            ASCII BUFFER
   18                ASCBU3  = $E1
   19                ASCBU2  = $E0
   20                ASCBUL  = $DF
   21                HEXBUH  = $DE            HEX BUFFER
   22                HEXBUL  = $DD
   23                STADH   = $DC            STARTING ADDRESS
   24                STADL   = $DB
   25                LOCCH   = $DA            LOCATION COUNTER
   26                LOCCL   = $D9
   27                TEMPR   = $D8            TEMPORARY REGISTER
   28                CHCC    = $D7            CHARACTER COUNTER
   29                MBEGH   = $D6            MEMORY BEGIN
   30                MBEGL   = $D5
   31                MENDH   = $D4            MEMORY END
   32                MENDL   = $D3
   33                MDESH   = $D2            MEMORY DESTINATION
   34                MDESL   = $D1
   35                ;
   36                        .OPT GENERATE    PRINT ASCII STRINGS
   37                        .OPT NOMEM
   38  0000                  *=$2000          ORG AT 2000 HEX
   39                ;
   40                ;
   41                ;***********************************************;
   42                ;***********************************************;
   43                ;**                                           **;
   44                ;**          MAIN ROUTINE FOLLOWS             **;
   45                ;**                                           **;
   46                ;***********************************************;
   47                ;***********************************************;
   48                ;
   49                ;
   50                ; LEDIP COLD ENTRY POINT (CSTAT)
   51                ;
   52                ; THIS ROUTINE TYPES A CRLF TWICE AND THE QUESTION
   53                ; "STARTING ADDRESS?" ON THE CONSOLE DEVICE.
   54                ;
   55  2000  D8      CSTAT   CLD              NO BCD ARITHMETIC
   56  2001  20 42 24        JSR CRLF         TYPE CR, LF
   57  2004  20 42 24        JSR CRLF
   58  2007  A0 EE           LDY #$EE         INITIALIZE FOR INDEXING
   59  2009  B9 88 23 CSTAT1 LDA STADO-$EE,Y
   60  200C  20 54 24        JSR OUTCH        TYPE A CHARACTER
   61  200F  C8              INY
   62  2010  D0 F7           BNE CSTAT1       NEXT CHARACTER
   63                ;
   64                ; READ STARTING ADDRESS (RSTAD)
   65                ;
   66                ; THIS ROUTINE RECEIVES FOUR ASCII CHARACTERS FROM
   67                ; THE CONSOLE DEVICE AND STORES THEM AS RECEIVED
   68                ; IN THE ASCII BUFFER. IT THEN CHECKS TO SEE IF
   69                ; ALL ARE VALID HEX CHARACTERS. ERROR MESSAGE "H"
   70                ; IS TYPED AND LEDIP RETURNED TO THE COLD START
   71                ; IF AN INVALID CHARACTER IS FOUND. CVAH IS CALLED
   72                ; AND STAD, LOCC INITIALIZED OTHERWISE.
   73                ;
   74  2012  20 30 24 RSTAD   JSR RDASC        READ FOUR CHARACTERS
   75  2015  20 07 23         JSR HCHK4        CHECK IF ALL ARE HEX
   76  2018  B0 10            BCS RSTADL
   77  201A  20 42 24         JSR CRLF
   78  201D  A9 48            LDA #'H
   79  201F  20 54 24         JSR OUTCH        TYPE H
   80  2022  A9 21            LDA #$21
   81  2024  20 54 24         JSR OUTCH        TYPE EXCLAMATION
   82  2027  4C 00 20         JMP CSTAT
   83  202A  20 C8 22 RSTADL  JSR CVAH         CONVERT ASCII TO HEX
   84  202D  20 42 24         JSR CRLF
   85  2030  A5 DD            LDA HEXBUL       INITIALIZE STAD AND LOCC
   86  2032  85 DB            STA STADL
   87  2034  85 D9            STA LOCCL
   88  2036  A5 DE            LDA HEXBUH
   89  2038  85 DC            STA STADH
   90  203A  85 DA            STA LOCCH
```

```
                        ;
                        ; LEDIP WARM ENTRY POINT (WSTAT)
                        ;
                        ; THIS ROUTINE ASSUMES THAT LOCC AND STAD ARE
                        ; ALREADY SET, TYPES A PROMPT CHARACTER, RECEIVES
                        ; FOUR CHARACTERS FROM THE CONSOLE DEVICE, AND IF
                        ; ALL ARE NUMERIC, CALLS CVAH (CONVERT ASCII TO
                        ; HEX). CONTROL IS OTHERWISE TRANSFERRED TO CMHD
                        ; (COMMAND HANDLER).
                        ;
2030  D8        WSTAT   CLD                     NO BCD ARITHMETIC
2031  A9 02             LDA #2
2033  85 D7             STA CHCC              INITIALIZE CHCC
2035  20 42 24          JSR CRLF              TYPE CR, LF
2038  A9 2F             LDA #$2F
203A  20 54 24          JSR OUTCH             TYPE PROMPT CHARACTER
203D  20 30 24          JSR RDASC             READ FOUR CHARACTERS
2040  20 0C 23          JSR DCHK4             CHECK IF LINE NUMBER
2043  B0 03             BCS AL8
2045  4C CE 21          JMP CMHD              ELSE CHECK IF COMMAND
2048  20 C8 22  AL8     JSR CVAH
204B  A5 DD             LDA HEXBUL            LINE # LOW ORDER BYTE
204D  A0 01             LDY #1
204F  91 D9             STA (LOCCL),Y
2051  D1 D9             CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
2053  F0 03             BEQ AL3
2055  4C 74 22  AL4     JMP INVM2
2058  A5 DE      AL3     LDA HEXBUH            LINE # HIGH ORDER BYTE
205A  C8                INY
205B  91 D9             STA (LOCCL),Y
205D  D1 D9             CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
205F  D0 F4             BNE AL4
2061  20 5D 24          JSR SPACE             TYPE SPACE IF OK
                        ;
                        ; RECEIVE ASCII (RVASC)
                        ;
                        ; THIS ROUTINE RECEIVES A LINE OF ASCII TEXT.
                        ; ERROR MESSAGE "L" IS TYPED AND CONTROL RETURNED
                        ; TO THE WARM START IF LENGTH OF LINE EXCEEDS
                        ; 252 CHARACTERS. CONTROL IS TRANSFERRED TO THE
                        ; APPROPRIATE ROUTINE OTHERWISE.
                        ;
2064  20 4B 24  RVASC   JSR GETCH             READ A CHARACTER
2067  C9 C8             CMP #$C8              IS IT BACKSPACE
2069  D0 0A             BNE RVASC1
206B  A5 D7             LDA CHCC
206D  C9 02             CMP #2
206F  F0 F3             BEQ RVASC
2071  C6 D7             DEC CHCC              DELETE LAST CHARACTER
2073  D0 EF             BNE RVASC
2075  E6 D7      RVASC1 INC CHCC              ACCEPT CHARACTER
2077  D0 10             BNE RVASC2
2079  20 42 24   INVL    JSR CRLF              LINE IS TOO LONG
207C  A9 4C             LDA #'L
207E  20 54 24          JSR OUTCH             TYPE L
2081  A9 21             LDA #$21
2083  20 54 24          JSR OUTCH             TYPE EXCLAMATION
2086  4C 3C 20          JMP WSTAT
2089  C9 0D      RVASC2 CMP #$0D              IS IT CARRIAGE-RETURN
208B  F0 0A             BEQ RVASC4
208D  A4 D7             LDY CHCC
208F  91 D9             STA (LOCCL),Y         STORE IF NOT
2091  D1 D9             CMP (LOCCL),Y
2093  D0 C0      AL2    BNE AL4
2095  F0 CD             BEQ RVASC             GET NEXT CHARACTER
2097  A5 D7      RVASC4 LDA CHCC              CHECK IF ZERO LENGTH
2099  C9 03             CMP #3
209B  D0 32             BNE LADJ              LINE ADJUST IF NOT
                        ;
                        ; LINE DELETE (LDEL)
                        ;
                        ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A
                        ; LINE DELETE OPERATION.
                        ;
209D  20 A4 22  LDEL    JSR HEXST             SET HEXBU TO STAD
20A0  20 AD 22  LDEL1   JSR CMPHL             COMPARE HEXBU WITH LOCC
20A3  F0 C5             BEQ AL1C
20A5  20 4F 23          JSR LNCHK             CHECK LINE NUMBER
20A8  B0 03             BCS AL7
20AA  4C 3C 20  AL10    JMP WSTAT             RETURN TO WARM START
20AD  F0 C6     AL7     BEQ LDEL2
20AF  20 2C 23          JSR INCHB             SET HEXBU TO NEXT LINE
20B2  4C A0 20          JMP LDEL1
20B5  A0 C0     LDEL2   LDY #0                GET LENGTH OF LINE
20B7  B1 DD             LDA (HEXBUL),Y
20B9  85 D8             STA TEMPR
20BB  20 ED 23          JSR MOV3              SET UP MOVMB PARAMETERS
20BE  20 60 23          JSR LSTLC             CHECK IF LAST LINE
20C1  F0 06             BEQ LDEL3             NO NEED FOR MOVE IF SO
20C3  20 10 24          JSR MOV4
20C6  20 77 23          JSR MOVMB             MOVE MEMORY BLOCK
20C9  20 E8 22  LDEL3   JSR DCLC
20CC  4C 3C 20          JMP WSTAT             RETURN TO WARM START
                        ;
                        ; LINE ADJUST (LADJ)
                        ;
                        ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A LINE
                        ; REPLACE OR MODIFY OPERATION.
                        ;
20CF  A5 D7     LADJ    LDA CHCC              GET CHARACTER COUNTER
20D1  A0 00             LDY #0
20D3  91 D9             STA (LOCCL),Y         STORE IT
20D5  D1 D9             CMP (LOCCL),Y         READ-AFTER-WRITE CHECK
20D7  D0 BA             BNE AL2
20D9  20 A4 22          JSR HEXST             SET HEXBU TO STAD
20DC  20 AD 22  LADJ1   JSR CMPHL             COMPARE HEXBU WITH LOCC
20DF  D0 06             BNE LADJ2
20E1  20 3C 23          JSR INCLC             ACCEPT LINE
20E4  4C 3C 20          JMP WSTAT
20E7  20 4F 23  LADJ2   JSR LNCHK             CHECK LINE NUMBER
20EA  B0 03             BCS AL9
20EC  4C A0 21          JMP LINS              LINE INSERT
20EF  F0 C6     AL9     BEQ CMPLL             LINE REPLACE
20F1  20 2C 23          JSR INCHB
20F4  4C E8 20          JMP LADJ1
20F7  A2 00     CMPLL   LDX #0                 COMPARE LINE LENGTHS
20F9  A1 D9             LDA (LOCCL,X)
20FB  C1 DD             CMP (HEXBUL,X)
20FD  F0 04             BEQ LADJ3             GO TO CORRECT ROUTINE
20FF  B0 0B             BCS LADJ4
2101  90 5E             BCC LADJ5
2103  20 C2 23  LADJ3   JSR MOV1              SAME LINE LENGTH
2106  20 77 23          JSR MOVMB
2109  4C 3C 20          JMP WSTAT             RETURN TO WARM START
                        ;
                        ; LINE ADJUST 4 (LADJ4)
                        ;
                        ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF
                        ; LENGTH OF CURRENT LINE (HEXBU) IS SHORTER THAN
                        ; LENGTH OF NEW LINE (LOCC).
                        ;
210C  A2 00     LADJ4   LDX #0                 FIND LENGTH DIFFERENCE
210E  A1 D9             LDA (LOCCL,X)
2110  E1 DD             SBC (HEXBUL,X)
2112  85 D8             STA TEMPR             SAVE IT
2114  20 19 24          JSR MOV5
2117  20 77 23          JSR MOVMB             MOVE MEMORY BLOCK
211A  20 ED 23          JSR MOV3
211D  20 60 23          JSR LSTLC             CHECK IF LAST LINE
2120  D0 06             BNE LADJ46
```

```
231  212E  20 3C 23          JSR INCLC          UPDATE LOCC
232  2131  4C 48 21          JMP LADJ47
233  2134  20 3C 23  LADJ46   JSR INCLC
234  2137  18                CLC
235  2138  A5 D4             LDA MENDH          SET UP MOVMB PARAMETERS
236  213A  85 D2             STA MDESH
237  213C  A5 D3             LDA MENDL
238  213E  65 D8             ADC TEMPR
239  2140  90 04             BCC LADJ43
240  2142  E6 D2             INC MDESH
241  2144  F0 24             BEQ AL6
242  2146  85 D1    LADJ43   STA MDESL
243  2148  20 9A 23          JSR MOVMR          MOVE MEMORY BLOCK
244  214B  20 C2 23  LADJ47   JSR MOV1
245  214E  20 77 23          JSR MOVMB
246  2151  A0 00             LDY #0
247  2153  38                SEC
248  2154  A5 D9             LDA LOCCL
249  2156  F1 D9             SBC (LOCCL),Y
250  2158  B0 02             BCS LADJ44
251  215A  C6 DA             DEC LOCCH
252  215C  18        LADJ44  CLC
253  215D  65 D8             ADC TEMPR
254  215F  85 D9             STA LOCCL
255  2161  90 04             BCC LADJ45
256  2163  E6 DA             INC LOCCH
257  2165  F0 03             BEQ AL6
258  2167  4C 3C 20  LADJ45  JMP WSTAT
259  216A  4C 74 22  AL6     JMP INVM2
260                  ;
261                  ; LINE ADJUST 5 (LADJ5)
262                  ;
263                  ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF LENGTH
264                  ; OF CURRENT LINE (HEXBU) IS LONGER THAN LENGTH OF
265                  ; NEW LINE (LOCC).
266                  ;
267  216D  38        LADJ5   SEC                FIND LENGTH DIFFERENCE
268  216E  A0 00             LDY #0
269  2170  B1 DD             LDA (HEXBUL),Y
270  2172  F1 D9             SBC (LOCCL),Y
271  2174  85 D8             STA TEMPR
272  2176  20 ED 23          JSR MOV3           SET PARAMETERS FOR MOVMB
273  2179  A5 DE             LDA HEXBUH
274  217B  85 D2             STA MDESH
275  217D  A5 DD             LDA HEXBUL
276  217F  18                CLC
277  2180  A2 00             LDX #0
278  2182  61 D9             ADC (LOCCL,X)
279  2184  90 04             BCC LADJ51
280  2186  E6 D2             INC MDESH
281  2188  F0 E0             BEQ AL6
282  218A  85 D1    LADJ51   STA MDESL
283  218C  20 60 23          JSR LSTLC          IS HEXBU AT LAST LINE
284  218F  F0 03             BEQ LADJ52
285  2191  20 77 23          JSR MOVMB
286  2194  20 C2 23  LADJ52  JSR MOV1
287  2197  20 77 23          JSR MOVMB
288  219A  20 E8 22          JSR DCLC           UPDATE LOCATION COUNTER
289  219D  4C 3C 20          JMP WSTAT
290                  ;
291                  ; LINE INSERT (LINS)
292                  ;
293                  ; CONTROL IS TRANSFERRED TO THIS ROUTINE FOR A
294                  ; LINE INSERTION.
295                  ;
296  21A0  20 19 24  LINS    JSR MOV5           INITIALIZE FOR MOVMB
297  21A3  20 77 23          JSR MOVMB
298  21A6  A5 DD             LDA HEXBUL         INITIALIZE AGAIN
299  21A8  85 D5             STA MBEGL
300  21AA  A5 DE             LDA HEXBUH
301  21AC  85 D6             STA MBEGH
302  21AE  20 00 24          JSR MOV32
303  21B1  20 1C 24          JSR MOV52
304  21B4  A5 D1             LDA MDESL
305  21B6  38                SEC
306  21B7  E9 01             SBC #1
307  21B9  B0 C2             BCS LINS1
308  21BB  C6 D2             DEC MDESH
309  21BD  85 D1    LINS1    STA MDESL
310  21BF  20 3C 23          JSR INCLC
311  21C2  20 9A 23          JSR MOVMR          MOVE MEMORY BLOCK
312  21C5  20 C2 23          JSR MOV1
313  21C8  20 77 23          JSR MOVMB          MOVE MEMORY BLOCK
314  21CB  4C 3C 20          JMP WSTAT
315                  ;
316                  ; COMMAND HANDLER (CMHD)
317                  ;
318                  ; CONTROL IS TRANSFERRED TO THIS ROUTINE IF A
319                  ; COMMAND IS TYPED ON THE CONSOLE DEVICE. A CHECK
320                  ; IS MADE TO SEE IF ASCBUM MATCHES THE FIRST
321                  ; LETTER OF ANY OF THE VALID LEDIP COMMANDS: VIZ
322                  ; LIST, TEXT, FILE, AND EXIT. IF A MATCH IS FOUND,
323                  ; LEDIP WAITS FOR A CARRIAGE-RETURN AND THEN
324                  ; TRANSFERS CONTROL TO THE APPROPRIATE ROUTINE.
325                  ; ERROR MESSAGE "C" IS TYPED AND CONTROL
326                  ; RETURNED TO THE WARM START OTHERWISE.
327                  ;
328  21CE  A5 E2     CMHD    LDA ASCBUM         FIRST LETTER OF COMMAND
329  21D0  29 DF             AND #%11011111     MASK, IF LOWER CASE
330  21D2  C9 45             CMP #'E            IS IT E
331  21D4  F0 16             BEQ EXIT           IF SO, EXIT
332  21D6  C9 46             CMP #'F            IS IT F
333  21D8  F0 28             BEQ FILE           IF SO, FILE
334  21DA  C9 54             CMP #'T            IS IT T
335  21DC  F0 51             BEQ TEXT           IF SO, TEXT
336  21DE  C9 4C             CMP #'L            IS IT L
337  21E0  F0 6C             BEQ LIST           IF SO, LIST
338  21E2  C9 43             CMP #'C            IS IT C
339  21E4  F0 13             BEQ CLEAR          IF SO, CLEAR
340  21E6  EA                NOP                ** COMMAND EXTENSION **
341  21E7  EA                NOP                ** COMMAND EXTENSION **
342  21E8  EA                NOP                ** COMMAND EXTENSION **
343  21E9  4C 84 22          JMP INVC
344  21EC  20 B8 22  EXIT    JSR CRSEN          WAIT FOR CR
345  21EF  A9 20             LDA #$20           WSTAT HIGH ORDER BYTE
346  21F1  48                PHA
347  21F2  A9 3C             LDA #$3C           LOW BYTE (TIM, $3D)
348  21F4  48                PHA
349  21F5  08                PHP
350  21F6  4C CC 1C          JMP $1CCC          (USE $7052 FOR TIM)
351  21F9  20 4B 24  CLEAR   JSR GETCH
352  21FC  20 B8 22          JSR CRSEN          WAIT FOR CR
353  21FF  4C 00 20          JMP CSTAT
354                  ;
355                  ; FILE COMMAND (FILE)
356                  ;
357                  ; THIS ROUTINE STATES THE BLOCKS OF MEMORY
358                  ; CURRENTLY BEING USED.
359                  ;
360  2202  20 B8 22  FILE    JSR CRSEN          WAIT FOR CR
361  2205  A0 EA             LDY #$EA           INIT X-REG FOR INDEXING
362  2207  B9 9E 23  FILE1   LDA FTAB-$EA,Y
363  220A  20 54 24          JSR OUTCH          TYPE A CHARACTER
364  220D  C8                INY
365  220E  D0 F7             BNE FILE1          NEXT CHARACTER
366  2210  20 42 24          JSR CRLF
367  2213  A5 0C             LDA STADH          START ADDRESS HIGH BYTE
368  2215  20 66 24          JSR OUTBYT
369  2218  A5 0B             LDA STADL          LOW BYTE
370  221A  20 66 24          JSR OUTBYT
```

```
371  221D  A9 2D        LDA #'-
372  221F  20 54 24     JSR OUTCH
373  2222  A5 DA        LDA LOCCH          LOCATION COUNTER
374  2224  20 66 24     JSR OUTBYT
375  2227  A5 D9        LDA LOCCL
376  2229  20 66 24     JSR OUTBYT
377  222C  4C 3C 20  ECMD  JMP WSTAT        RETURN TO WARM START
378                     ;
379                     ; TEXT COMMAND (TEXT)
380                     ;
381                     ; THIS ROUTINE LISTS THE CURRENT TEXT
382                     ; WITHOUT LINE NUMBERS.
383                     ;
384  222F  20 B8 22  TEXT  JSR CRSEN        WAIT FOR CR
385  2232  20 9E 22     JSR CLHS           TYPE CR, SET HEXBU
386  2235  20 AD 22  TEXT1 JSR CMPHL        CHECK IF LAST LINE
387  2238  D0 06        BNE TEXT2
388  223A  20 42 24     JSR CRLF
389  223D  4C 3C 20     JMP WSTAT
390  2240  20 F4 22  TEXT2 JSR DPASC        TYPE A LINE OF TEXT
391  2243  A9 7F        LDA #$7F           TYPE RUB-OUT
392  2245  20 54 24     JSR OUTCH
393  2248  20 2C 23     JSR INCHB          SET FOR NEXT LINE
394  224B  4C 35 22     JMP TEXT1
395                     ;
396                     ; LIST COMMAND (LIST)
397                     ;
398                     ; THIS ROUTINE LISTS THE CURRENT TEXT
399                     ; WITH LINE NUMBERS.
400                     ;
401  224E  20 B8 22  LIST  JSR CRSEN        WAIT FOR CR
402  2251  20 A4 22     JSR HEXST          SET HEXBU TO STAD
403  2254  20 AD 22  LIST1 JSR CMPHL
404  2257  F0 D3        BEQ ECMD
405  2259  A0 02        LDY #2
406  225B  B1 DD        LDA (HEXBUL),Y     TYPE LINE NUMBER
407  225D  20 66 24     JSR OUTBYT
408  2260  88          DEY
409  2261  B1 DD        LDA (HEXBUL),Y
410  2263  20 66 24     JSR OUTBYT
411  2266  20 5D 24     JSR SPACE          TYPE A SPACE
412  2269  20 F4 22     JSR DPASC          TYPE A LINE OF TEXT
413  226C  20 2C 23     JSR INCHB
414  226F  4C 54 22     JMP LIST1          NEXT LINE
415                     ;
416                     ; INVALID MEMORY (INVM1,INVM2)
417                     ;
418                     ; THIS MULTIPLE ENTRY ROUTINE PRINTS ERROR
419                     ; MESSAGE "M" ON THE CONSOLE DEVICE AND RETURNS
420                     ; CONTROL TO THE WARM START. INVM1 RESTORES THE
421                     ; STACK WHILE INVM2 DOES NOT.
422                     ;
423  2272  68        INVM1 PLA                RESTORE STACK
424  2273  68           PLA
425  2274  20 42 24  INVM2 JSR CRLF
426  2277  A9 4D        LDA #'M
427  2279  20 54 24     JSR OUTCH          TYPE M
428  227C  A9 21        LDA #$21
429  227E  20 54 24     JSR OUTCH          TYPE EXCLAMATION
430  2281  4C 3C 20     JMP WSTAT
431                     ;
432                     ; INVALID COMMAND (INVC)
433                     ;
434                     ; THIS ROUTINE TYPES ERROR MESSAGE "C" ON THE
435                     ; CONSOLE DEVICE AND TRANSFERS CONTROL TO THE WARM
436                     ; START.
437                     ;
438  2284  20 42 24  INVC  JSR CRLF
439  2287  A9 43        LDA #'C
440  2289  20 54 24     JSR OUTCH          TYPE C
441  228C  A9 21        LDA #$21
442  228E  20 54 24     JSR OUTCH          TYPE EXCLAMATION
443  2291  4C 3C 20     JMP WSTAT
444                     ;
445                     ;
446                     ;***************************************************;
447                     ;***************************************************;
448                     ;**                                               **;
449                     ;**  SUBROUTINES FOLLOW IN ALPHABETICAL ORDER     **;
450                     ;**                                               **;
451                     ;***************************************************;
452                     ;***************************************************;
453                     ;
454                     ;
455                     ; ASCII BIAS (ABIAS)
456                     ;
457                     ; THIS MULTIPLE EXIT SUBROUTINE ADDS THE
458                     ; APPROPRIATE BIAS TO THE ASCII CONTENT OF THE
459                     ; ACCUMULATOR TO MAKE IT A HEX CHARACTER.
460                     ; A DESTROYED, X AND Y PRESERVED.
461                     ;
462  2294  C9 3A     ABIAS  CMP #$3A          COMPARE WITH 9
463  2296  B0 03        BCS ABIAS1         MUST BE A-F IF GREATER
464  2298  E9 2F        SBC #$2F           BIAS FOR 0 THRU 9
465  229A  60           RTS
466  229B  E9 37     ABIAS1 SBC #$37          BIAS FOR A THRU F
467  229D  60           RTS
468                     ;
469                     ; CRLF, HEXBU SET TO STAD (CLHS)
470                     ; HEXBU SET TO STAD (HEXST)
471                     ;
472                     ; THIS MULTIPLE ENTRY SUBROUTINE TYPES A CRLF
473                     ; TWICE AND/OR SETS HEXBU EQUAL TO STAD. ENTRY
474                     ; POINT TO JUST SET HEXBU TO STAD IS HEXST.
475                     ; ENTRY POINT FOR BOTH IS CLHS.
476                     ; A DESTROYED, X AND Y PRESERVED.
477                     ;
478  229E  20 42 24  CLHS  JSR CRLF          TYPE CR, LF
479  22A1  20 42 24     JSR CRLF
480  22A4  A5 DB     HEXST  LDA STADL         SET LOW ORDER BYTE
481  22A6  85 DD        STA HEXBUL
482  22A8  A5 DC        LDA STADH          HIGH ORDER BYTE TOO
483  22AA  85 DE        STA HEXBUH
484  22AC  60           RTS
485                     ;
486                     ; COMPARE HEXBU AND LOCC (CMPHL)
487                     ;
488                     ; THIS SUBROUTINE COMPARES THE CONTENTS OF
489                     ; HEXBU AND LOCC. ZERO FLAG IS SET IF THEY
490                     ; ARE EQUAL. ZERO FLAG IS OTHERWISE CLEARED.
491                     ; A DESTROYED, X AND Y PRESERVED.
492                     ;
493  22AD  A5 DD     CMPHL  LDA HEXBUL         COMPARE LOW ORDER BYTES
494  22AF  C5 D9        CMP LOCCL
495  22B1  D0 04        BNE CMPHL1
496  22B3  A5 DE        LDA HEXBUH         HIGH ORDER BYTES TOO
497  22B5  C5 DA        CMP LOCCH
498  22B7  60        CMPHL1 RTS
499                     ;
500                     ; CARRIAGE-RETURN SENSE (CRSEN)
501                     ;
502                     ; THIS SUBROUTINE RECEIVES A CHARACTER FROM THE
503                     ; CONSOLE DEVICE AND CHECKS TO SEE IF IT IS A CR.
504                     ; IF SO, A CRLF IS ECHOED. IF NOT, ERROR MESSAGE
505                     ; C IS TYPED AND CONTROL RETURNED TO THE WARM START.
506                     ; A DESTROYED, X AND Y PRESERVED.
507                     ;
508  22B8  20 48 24  CRSEN  JSR GETCH         RECEIVE CHARACTER
509  22BB  C9 0D        CMP #$0D           IS IT CARRIAGE-RETURN
510  22BD  F0 C5        BEQ CRSEN1
```

```
511  228F  68           PLA              RESTORE STACK
512  2290  68           PLA
513  2291  4C 84 22     JMP INVC         TYPE ERROR IF NOT
514  2294  20 42 24  CRSEN1  JSR CRLF     ELSE ECHO WITH CRLF
515  2297  60           RTS
516                  ;
517                  ; CONVERT ASCII TO HEX (CVAH)
518                  ;
519                  ; THIS SUBROUTINE CONVERTS THE ASCII CONTENTS OF
520                  ; ASCBU (ASCII BUFFER) INTO THE HEX EQUIVALENT
521                  ; AND STORES THE ANSWER IN HEXBU (HEX BUFFER).
522                  ; ABIAS CALLED, A, X, TEMPR DESTROYED, Y CLEARED.
523                  ;
524  2298  A0 02  CVAH    LDY #2           SET INDEX
525  229A  A2 04          LDX #4           SET INDEX
526  229C  B5 DE  CVAH1   LDA ASCBUM-4,X   GET CHARACTER
527  229E  20 94 22       JSR ABIAS        APPLY BIAS AND SHIFT
528  22A1  0A             ASL A
529  22A2  0A             ASL A
530  22A3  0A             ASL A
531  22A4  0A             ASL A
532  22A5  85 D8          STA TEMPR
533  22A7  CA             DEX
534  22A8  B5 DE          LDA ASCBUM-4,X    NEXT CHARACTER
535  22AA  20 94 22       JSR ABIAS
536  22AD  18             CLC
537  22AE  65 D8          ADC TEMPR
538  22B0  99 DC 00       STA HEXBUH-2,Y    STORE IN HEXBU
539  22B3  CA             DEX
540  22B4  88             DEY
541  22B5  D0 E5          BNE CVAH1         NEXT CHARACTER
542  22B7  60             RTS
543                  ;
544                  ; DECREMENT LOCATION COUNTER (DCLC)
545                  ;
546                  ; THIS SUBROUTINE SUBTRACTS TEMPR (TEMPORARY
547                  ; REGISTER) FROM LOCC (LOCATION COUNTER) AND STORES
548                  ; THE ANSWER IN LOCC.
549                  ; A DESTROYED, X AND Y PRESERVED.
550                  ;
551  22B8  38     DCLC    SEC
552  22B9  A5 D9          LDA LOCCL         GET LOCC LOW ORDER BYTE
553  22BB  E5 D8          SBC TEMPR         SUBTRACT TEMPR
554  22BD  85 D9          STA LOCCL
555  22BF  B0 02          BCS DCLC1
556  22F1  C6 DA          DEC LOCCH         PROPAGATE BORROW
557  22F3  60     DCLC1   RTS
558                  ;
559                  ; DUMP ASCII (DPASC)
560                  ;
561                  ; THIS SUBROUTINE TYPES THE STRING OF  ASCII
562                  ; CHARACTERS CONTAINED IN THE DATA BLOCK POINTED
563                  ; TO BY HEXBU (HEX BUFFER).
564                  ; A AND Y DESTROYED, X CLEARED.
565                  ;
566  22F4  A0 03  DPASC   LDY #3            SET INDEX
567  22F6  B1 D0  DPASC1  LDA (HEXBUL),Y    GET CHARACTER
568  22F8  20 54 24       JSR OUTCH         TYPE IT
569  22FB  C8             INY               SET FOR NEXT CHARACTER
570  22FC  98             TYA
571  22FD  A2 00          LDX #0            CHECK IF LAST CHARACTER
572  22FF  C1 DD          CMP (HEXBUL,X)
573  2301  D0 F3          BNE DPASC1        NEXT CHARACTER
574  2303  20 42 24       JSR CRLF
575  2306  60             RTS
576                  ;
577                  ; HEXADECIMAL CHECK (HCHK4)
578                  ; DECIMAL CHECK (DCHK4)
579                  ;
580                  ; THIS MULTIPLE ENTRY SUBROUTINE CHECKS TO SEE IF
581                  ; ALL THE CONTENTS OF ASCBU (ASCII BUFFER) ARE
582                  ; VALID HEXADECIMAL OR DECIMAL CHARACTERS. ENTRY
583                  ; POINT FOR HEX CHECK IS HCHK4. ENTRY POINT FOR
584                  ; DECIMAL CHECK IS DCHK4. C-FLAG IS CLEARED IF AN
585                  ; INVALID CHARACTER IS FOUND. C-FLAG IS OTHERWISE
586                  ; SET.
587                  ; A, X, AND Y DESTROYED.
588                  ;
589  2307  AC FF  HCHK4   LDY #$FF          SET FOR HEX CHECK
590  2309  4C 0E 23       JMP CHEK1
591  230C  A0 00  DCHK4   LDY #0            SET FOR DECIMAL CHECK
592  230E  A2 04  CHEK1   LDX #4
593  2310  B5 DE  CHEK2   LDA ASCBUM-4,X    GET A BYTE
594  2312  C0 00          CPY #0
595  2314  F0 08          BEQ DCHK
596  2316  C9 47  HCHK    CMP #$47          CHECK IF HEX
597  2318  B0 0C          BCS ECHK1
598  231A  C9 41          CMP #$41
599  231C  B0 0A          BCS ECHK2
600  231E  C9 3A  DCHK    CMP #$3A          CHECK IF DECIMAL
601  2320  B0 04          BCS ECHK1
602  2322  C9 30          CMP #$30
603  2324  B0 02          BCS ECHK2
604  2326  18      ECHK1   CLC               INVALID CHAR, SET C=0
605  2327  60             RTS
606  2328  CA      ECHK2   DEX
607  2329  D0 E5          BNE CHEK2         NEXT CHARACTER
608  232B  60       ECHK    RTS
609                  ;
610                  ; INCREMENT HEXBU (INCHB)
611                  ;
612                  ; THIS SUBROUTINE ADDS THE CONTENTS OF THE
613                  ; MEMORY LOCATION POINTED TO BY HEXBU TO
614                  ; HEXBU AND STORES THE ANSWER IN HEXBU.
615                  ; A DESTROYED, X CLEARED, Y PRESERVED.
616                  ;
617  232C  A2 00  INCHB   LDX #0            CLEAR INDEX
618  232E  A1 DD          LDA (HEXBUL,X)    GET (HEXBU)
619  2330  18             CLC
620  2331  65 DD          ADC HEXBUL        ADD TO LOW ORDER BYTE
621  2333  85 DD          STA HEXBUL
622  2335  90 04          BCC INCHB1
623  2337  E6 DE          INC HEXBUH        PROPAGATE CARRY
624  2339  F0 10          BEQ INCLC2        ERROR IF OVER 65K
625  233B  60       INCHB1  RTS
626                  ;
627                  ; INCREMENT LOCATION COUNTER (INCLC)
628                  ;
629                  ; THIS SUBROUTINE ADDS THE CONTENTS OF THE
630                  ; MEMORY LOCATION POINTED TO BY LOCC TO THE
631                  ; THE VALUE OF LOCC AND STORES THE ANSWER IN LOCC.
632                  ; A DESTROYED, X CLEARED, Y PRESERVED.
633                  ;
634  233C  A2 00  INCLC   LDX #0            CLEAR INDEX
635  233E  A1 D9          LDA (LOCCL,X)     GET (LOCC)
636  2340  18             CLC
637  2341  65 D9          ADC LOCCL         ADD TO LOW ORDER BYTE
638  2343  85 D9          STA LOCCL
639  2345  90 07          BCC INCLC1
640  2347  E6 DA          INC LOCCH         PROPAGATE CARRY
641  2349  D0 03          BNE INCLC1
642  234B  4C 72 22  INCLC2  JMP INVM1      ERROR IF OVER 65K
643  234E  60       INCLC1  RTS
644                  ;
645                  ; LINE NUMBER CHECK (LNCHK)
646                  ;
```

```
                        ; THIS SUBROUTINE COMPARES THE LINE NUMBER IN THE
                        ; CURRENT LOCC (LOCATION COUNTER) TO THE LINE NUMBER
                        ; IN HEXBU (HEX BUFFER). ZERO FLAG IS SET IF AN
                        ; IDENTICAL LINE NUMBER IS FOUND IN HEXBU. CARRY
                        ; FLAG IS CLEARED IF A HIGHER LINE NUMBER IS FOUND
                        ; IN HEXBU. HEXBU IS LEFT AS IT IS.
                        ; A AND Y DESTROYED, X PRESERVED.
                        ;
234F  A0 02     LNCHK   LDY #2              SET INDEX
2351  B1 D9              LDA (LOCCL),Y
2353  D1 DD              CMP (HEXBUL),Y      COMPARE LOW ORDER BYTES
2355  90 02              BCC LNCHK1
2357  F0 01              BEQ LNCHK2
2359  60        LNCHK1  RTS
235A  88        LNCHK2  DEY
235B  B1 D9              LDA (LOCCL),Y
235D  D1 DD              CMP (HEXBUL),Y      HIGH ORDER BYTES TOO
235F  60                 RTS
                        ;
                        ; LAST LINE CHECK (LSTLC)
                        ;
                        ; THIS SUBROUTINE CHECKS FOR INVALID MEMORY
                        ; BLOCK MOVES WHICH MAY RESULT FROM AN ATTEMPT
                        ; TO ADJUST THE LAST LINE OF TEXT.
                        ; A DESTROYED, X AND Y PRESERVED.
                        ;
2360  A5 D5     LSTLC   LDA MBEGL           CHECK LOW ORDER BYTES
2362  C5 D9              CMP LOCCL
2364  F0 01              BEQ LSTLC1
2366  60                 RTS
2367  A5 D6     LSTLC1  LDA MBEGH           HIGH ORDER BYTES TOO
2369  C5 DA              CMP LOCCH
236B  60                 RTS
                        ;
                        ; MEMORY MOVE CHECK (MCHEK)
                        ;
                        ; THIS SUBROUTINE CHECKS TO SEE IF A BLOCK MEMORY
                        ; MOVE IS COMPLETE. ZERO FLAG IS SET IF SO.
                        ; A DESTROYED, X AND Y PRESERVED.
                        ;
236C  A5 D3     MCHEK   LDA MENDL           COMPARE LOW ORDER BYTES
236E  C5 D5             CMP MBEGL
2370  D0 04              BNE ENDMC
2372  A5 D4              LDA MENDH           HIGH ORDER BYTES TOO
2374  C5 D6              CMP MBEGH
2376  60        ENDMC   RTS
                        ;
                        ; MOVE MEMORY BLOCK (MOVMB)
                        ;
                        ; THIS SUBROUTINE MOVES THE CONTENTS OF MEMORY BLOCK
                        ; MBEG (MEMORY BEGIN) TO MEND (MEMORY END) INCLUSIVE
                        ; TO THE MEMORY BLOCK BEGINNING AT LOCATION MDES
                        ; (MEMORY DESTINATION). BYTES ARE MOVED IN LOW
                        ; ADDRESS TO HIGH ADDRESS SEQUENCE.
                        ; MBEG AND MDES DESTROYED, Y CLEARED, A DESTROYED,
                        ; X PRESERVED.
                        ;
2377  A0 00     MOVMB   LDY #0              CLEAR INDEX
2379  B1 D5     MOVM1   LDA (MBEGL),Y       MOVE A BYTE
237B  91 D1               STA (MDESL),Y
237D  D1 D1               CMP (MDESL),Y     READ-AFTER-WRITE CHECK
237F  F0 03               BEQ AL1
2381  4C 72 22 AL5       JMP INVM1
2384  20 6C 23 AL1       JSR MCHEK         IS BLOCK MOVE COMPLETE
2387  F0 10               BEQ ENDM
2389  E6 D5               INC MBEGL         SET FOR NEXT BYTE IF NOT
238B  D0 02               BNE MOVM3
238D  E6 D6               INC MBEGH
238F  E6 D1     MOVM3   INC MDESL
2391  D0 E6               BNE MOVM1
2393  E6 D2               INC MDESH         PROPAGATE CARRY
2395  D0 E2               BNE MOVM1         NEXT BYTE
2397  F0 E8               BEQ AL5
2399  60        ENDM    RTS
                        ;
                        ; MOVE MEMORY BLOCK REVERSE (MOVMR)
                        ;
                        ; THIS SUBROUTINE IS IDENTICAL TO MOVMB (MOVE MEMORY
                        ; BLOCK) WITH TWO EXCEPTIONS: VIZ. (1) BYTES ARE
                        ; MOVED IN HIGH ADDRESS TO LOW ADDRESS SEQUENCE,
                        ; (2) MDES INDICATES END (AND NOT BEGINNING) OF
                        ; DESTINATION BLOCK.
                        ; MEND AND MDES DESTROYED. Y CLEARED, A DESTROYED,
                        ; X PRESERVED.
                        ;
239A  A0 00     MOVMR   LDY #0              CLEAR INDEX
239C  B1 D3     MOVR1   LDA (MENDL),Y       MOVE A BYTE
239E  91 D1             STA (MDESL),Y
23A0  D1 D1             CMP (MDESL),Y       READ-AFTER-WRITE CHECK
23A2  D0 DD             BNE AL5
23A4  20 6C 23          JSR MCHEK           IS MOVE COMPLETE
23A7  F0 18             BEQ ENDMR
23A9  38                SEC                 SET FOR NEXT BYTE IF NOT
23AA  A5 D3             LDA MENDL
23AC  E9 01             SBC #1
23AE  85 D3             STA MENDL
23B0  B0 02             BCS MOVR2
23B2  C6 D4             DEC MENDH
23B4  38        MOVR2   SEC
23B5  A5 D1             LDA MDESL
23B7  E9 01             SBC #1
23B9  85 D1             STA MDESL
23BB  B0 DF             BCS MOVR1
23BD  C6 D2             DEC MDESH
23BF  90 D8             BCC MOVR1           NEXT BYTE
23C1  60        ENDMR   RTS
                        ;
                        ; MEMORY MOVE INITIALIZE (MOV1)
                        ;
                        ; MBEG = LOCC
                        ; MEND = LOCC + (LOCC) - 1
                        ; MDES = HEXBU
                        ;
23C2  20 C9 23  MOV1    JSR MOV2
23C5  20 10 24          JSR MOV4
23C8  60                RTS
                        ;
                        ; MEMORY MOVE INITIALIZE (MOV2)
                        ;
                        ; MBEG = LOCC
                        ; MEND = LOCC + (LOCC) - 1
                        ;
23C9  A5 DA     MOV2    LDA LOCCH           SET HIGH ORDER BYTES
23CB  85 D6             STA MBEGH
23CD  85 D4             STA MENDH
23CF  A5 D9             LDA LOCCL           LOW ORDER BYTES TOO
23D1  85 D5             STA MBEGL
23D3  18                CLC
23D4  A2 00             LDX #0
23D6  61 D9             ADC (LOCCL,X)       ADD CONTENTS OF LOCC
23D8  90 09             BCC MOV21
23DA  E6 D4             INC MENDH
23DC  D0 05             BNE MOV21
23DE  68                PLA                 RESTORE STACK
23DF  68                PLA
23E0  4C 72 22  AL11    JMP INVM1           TYPE ERROR
23E3  38        MOV21   SEC
23E4  E9 01             SBC #1
23E6  B0 02             BCS MOV22
23E8  C6 D4             DEC MENDH
```

```
 787  23EA  85 D3      MOV22   STA MENDL
 788  23EC  60                 RTS
 789                   ;
 790                   ; MEMORY MOVE INITIALIZE (MOV3, MOV33)
 791                   ;
 792                   ; MBEG = HEXBU + (HEXBU)
 793                   ; MEND = LOCC - 1
 794                   ;
 795  23ED  A5 DE      MOV3    LDA HEXBUH        SET TO HEXBU
 796  23EF  85 D6              STA MBEGH
 797  23F1  A5 DD              LDA HEXBUL
 798  23F3  18                 CLC
 799  23F4  A2 00              LDX #0
 800  23F6  61 DD              ADC (HEXBUL,X)    ADD CONTENTS OF HEXBU
 801  23F8  90 04              BCC MOV31
 802  23FA  E6 D6              INC MBEGH
 803  23FC  F0 E2              BEQ ALL1
 804  23FE  85 D5      MOV31   STA MBEGL
 805  2400  A5 DA      MOV33   LDA LOCCH         MEND = LOCC - 1
 806  2402  85 D4              STA MENDH
 807  2404  A5 D9              LDA LOCCL
 808  2406  38                 SEC
 809  2407  E9 01              SBC #1
 810  2409  B0 02              BCS MOV32
 811  240B  C6 D4              DEC MENDH
 812  240D  85 D3      MOV32   STA MENDL
 813  240F  60                 RTS
 814                   ;
 815                   ; MEMORY MOVE INITIALIZE (MOV4)
 816                   ;
 817                   ; MDES = HEXBU
 818                   ;
 819  2410  A5 DD      MOV4    LDA HEXBUL
 820  2412  85 D1              STA MDESL
 821  2414  A5 DE              LDA HEXBUH
 822  2416  85 D2              STA MDESH
 823  2418  60                 RTS
 824                   ;
 825                   ; MEMORY MOVE INITIALIZE (MOV5, MOV52)
 826                   ;
 827                   ; MBEG = LOCC
 828                   ; MEND = LOCC + (LOCC) - 1
 829                   ; MDES = LOCC + (LOCC)
 830                   ;
 831  2419  20 C9 23   MOV5    JSR MOV2
 832  241C  A5 DA      MOV52   LDA LOCCH         MDES = LOCC + (LOCC)
 833  241E  85 D2              STA MDESH
 834  2420  A2 00              LDX #0
 835  2422  18                 CLC
 836  2423  A5 D9              LDA LOCCL
 837  2425  61 D9              ADC (LOCCL,X)
 838  2427  90 04              BCC MOV51
 839  2429  E6 D2              INC MDESH
 840  242B  F0 B3              BEQ ALL1
 841  242D  85 D1      MOV51   STA MDESL
 842  242F  60                 RTS
 843                   ;
 844                   ; READ ASCII (RDASC)
 845                   ;
 846                   ; THIS SUBROUTINE READS FOUR ASCII CHARACTERS FROM
 847                   ; THE CONSOLE DEVICE AND STORES THEM AS RECEIVED
 848                   ; IN ASCBU (ASCII BUFFER). FIRST CHARACTER
 849                   ; RECEIVED IS STORED IN HIGHEST LOCATION (ASCBUM).
 850                   ; X CLEARED, A DESTROYED, Y PRESERVED.
 851                   ;
 852  2430  A2 04      RDASC   LDX #4            SET INDEX
 853  2432  20 4B 24   RDASC1  JSR GETCH         GET A CHARACTER
 854  2435  95 DE              STA ASCBUM-4,X    STORE IT
 855  2437  CA                 DEX
 856  2438  D0 F8              BNE RDASC1        NEXT CHARACTER
 857  243A  60                 RTS
 858                   ;
 859                   ; SAVE REGISTERS (SAVR)
 860                   ;
 861  243B  85 D5      SAVR    STA MBEGL
 862  243D  86 D3              STX MENDL
 863  243F  84 D1              STY MDESL
 864  2441  60                 RTS
 865                   ;
 866                   ; I/O JUMPS
 867                   ;
 868  2442  20 3B 24   CRLF    JSR SAVR          CARRIAGE-RET LINE-FEED
 869  2445  20 2F 1E           JSR $1E2F         (USE $723A FOR TIM)
 870  2448  4C 6C 24           JMP RESR
 871  244B  20 3B 24   GETCH   JSR SAVR          READ A CHARACTER
 872  244E  20 5A 1E           JSR $1E5A         (USE $72E9 FOR TIM)
 873  2451  4C 6E 24           JMP RESR1
 874  2454  20 3B 24   OUTCH   JSR SAVR          TYPE A CHARACTER
 875  2457  20 A0 1E           JSR $1EA0         (USE $72C6 FOR TIM)
 876  245A  4C 6C 24           JMP RESR
 877  245D  20 3B 24   SPACE   JSR SAVR          TYPE A SPACE
 878  2460  20 9E 1E           JSR $1E9E         (USE $7377 FOR TIM)
 879  2463  4C 6C 24           JMP RESR
 880  2466  20 3B 24   OUTBYT  JSR SAVR          TYPE A BYTE
 881  2469  20 3B 1E           JSR $1E3B         (USE $7281 FOR TIM)
 882  246C  A5 D5      RESR    LDA MBEGL         RESTORE REGISTERS
 883  246E  A6 D3      RESR1   LDX MENDL
 884  2470  A4 D1              LDY MDESL
 885  2472  EA                 NOP               ** FOR BREAK TEST **
 886  2473  EA                 NOP               ** FOR BREAK TEST **
 887  2474  EA                 NOP               ** FOR BREAK TEST **
 888  2475  60                 RTS
 889                   ;
 890                   ; ASCII TABLES
 891                   ;
 892  2476  53 54      STADO   .BYTE 'STARTING ADDRESS? '
 892  2478  41 52
 892  247A  54 49
 892  247C  4E 47
 892  247E  20 41
 892  2480  44 44
 892  2482  52 45
 892  2484  53 53
 892  2486  3F 20
 893  2488  30 30      FTAB    .BYTE '00D1-00E2'
 893  248A  44 31
 893  248C  2D 30
 893  248E  30 45 32
 894  2491  0D                 .BYTE $0D, $00, $0A, $0C
 894  2492  00
 894  2493  0A
 894  2494  0C
 895  2495  32 30              .BYTE '2000-2490'
 895  2497  30 30
 895  2499  2D 32
 895  249B  34 39 44
 896                           .END

END OF MOS/TECHNOLOGY 650X ASSEMBLY VERSION 4
NUMBER OF ERRORS =    0,  NUMBER OF WARNINGS =    0
```

QUICK AND DIRTY FUNCTIONS FOR BASIC

BY DENNIS ALLISON
169 Spruce Ave.
Menlo Park, CA 94025

Recently, Gene Wallis called my attention to the fact that a number of BASIC interpreters for the hobbyist are not provided with a full complement of intrinsic functions. He was working on an energy utilization program and needed sine/cosine, log, and power functions badly. These programs are quick and dirty implementations which I did to give him a tool to get his job done.

A word of warning though. These are *quick and dirty* in the sense that they have not been carefully tuned or tested. They may well contain errors. Certainly the choice of approximation may not be optimal. All the functions are computed by using various range reducing formulae and then applying a polynomial or rational approximation from Hart's *Computer Approximations* (Wiley, 1968), an encyclopedic compendium of approximations which is now out of print. The approximations given in the algorithm section are all accurate to better than seven digits, but the programs should be trusted to only six digits or so because of truncation of the coefficients, carelessness of implementation, and the vagaries of local floating point arithmetic.

The programs are also given in BASIC. They have been tested using the dialect of BASIC supported by the Unix timesharing system. The Unix BASIC differs substantially from most hobby BASICS, so the test programs were transliterated into something more generally acceptable. Beware—errors may have crept into the code.

Lastly, if you do use the programs, considerable efficiency can be gained by tuning the code to your particular BASIC. In writing the functions we chose the simplest possible code rather than the most efficient. A little rewriting may speed the functions enormously. All these functions are to be invoked via gosubs; they expect an argument in x and return a value in y. $x0, x1, x2, \ldots$ are used as temporaries.

Square Root—SQR(x)

1. $x < 0$ is an error
2. reduce range
   $t = 1$
   $\{x = x/100; t = t * 10\}$ until $x \leqslant 1$
   note: $x * t * t$ = original number
   $0 \leqslant x \leqslant 1$
3. make a guess at $\sqrt{x}$ via a polynomial (accuracy is .56 digits):
   $y_0 = .115442 + 1.15442 * x$
4. iterate until $|y_{n+1} - y_n| < 10^{-6}$ (or so—depends upon floating point accuracy) using:
   $$y_{n+1} = \tfrac{1}{2}\left(y_n + \frac{x}{y_n}\right) = y_n + \tfrac{1}{2}\left(\frac{x}{y_n} - y_n\right)$$
5. final answer is:
   $y_{n+1} * t$

You don't really have to do the test in 4. as the iteration will double the accuracy (approx) every time. So:

	iteration	accuracy (digits)
	0	0.5
	1	1
	2	2
	3	4
should be enough →	4	8
certainly enough →	5	16

```
500 rem y = sqrt(x)
510 if x<0 then goto <error>
520 x1 = 1
530 if x <=1 then goto 570
540 x = x/100
550 x1 = x1 * 10
560 goto 530
570 y = .115442 + 1.15442*x
580 x2 = 5
590 y = 0.5*(y+(x/y))
600 x2 = x2 - 1
610 if x2>0 then goto 590
620 y = y*x1
630 return
```

Log Base 10

1. $x \leqslant 0$ is an error
2. $x < 1$ then $\left\{ \text{sign} = -1 \quad x = \frac{1}{x} \right\}$ else sign = 1
3. reduce range. find n such that

   since $\log\left(\frac{1}{x}\right) = -\log(x)$

   $$\frac{1}{10} \leqslant \frac{x}{10^n} < 1$$

   and set $x = \frac{x}{10^n}$
4. $s = \sqrt{10}\, x$
5. $z = (s - 1) \,/\, (s + 1)$

   $$r = z\,\frac{P(z^2)}{Q(z^2)}$$
6. lgt = $(n + r - .5) * \text{sign}$

P_0	3.1878	22082	024
P_1	−2.6558	08794	66
P_2	.26686	32700	47
Q_0	3.6701	15625	115
Q_1	−4.2809	73292	83
Q_2	1.0		

```
500 rem y = log(x)  base 10
510 if x <= 0 then goto <error>
520 x1 = 1
530 if x < 1 then x1 = -1
540 if x < 1 then x = 1/x
550 x2 = 0
560 if x < 1 then goto 600
570 x = x/10
580 x2 = x2 + 1
590 goto 560
600 x = 3.162278*x
610 x = (x+1)/(x-1)
620 y = x*x
630 x3=3.187822+y*(-2.655809+y*0.2668633)
640 x4=3.670116+y*(-4.280973+y)
650 y = (x2 + x*x3/x4 -.5)*x1
660 return
```

Power 10^x

1. alarms if x too large positively
   return zero if large negatively
2. if $x < 0$ $\left\{ \text{inv} = \underline{\text{true}},\ x = |x| \right\}$
   else $\left\{ \text{inv} = \underline{\text{false}} \right\}$
   reduce range to $[0, \tfrac{1}{2}]$

   $n = \text{int}(x)$

   $x = x - n - .5$

   $$z = \frac{2x\,P(x^2)}{Q(x^2) - x\,P(x^2)}$$

   result = $\sqrt{10}\ 10^n\,(z + 1)$
   if (inv) $\Rightarrow$ result = 1 / result

P_0	69.5569663167251
P_1	8.7517567967368
Q_0	60.4164133787158
Q_1	34.2951432306918
Q_2	1

```
500 rem y = 10^x
510 x1 = 0
520 if x < 0 then x1 = 1
530 if x < 0 then x = -x
540 x2 = int(x)
550 x = x - n - .5
560 y = x * x
570 x3 = 69.55697+y*8.751758
580 x4 = 60.41641+y*(34.29514+y)
590 y = (2*y*x3)/(x4-y*x3)
600 if x2 = 0 then goto 640
610 y = y * 10
620 x2 = x2 - 1
630 goto 600
640 if x1 = 1 then y = 1/y
650 return
```

Inverse Trig

1. If $x < 0$ return $-\text{atan}(|x|)$
   since $\text{atan}(-x) = -\text{atan}(x)$

2. If $x > 1$ return $\frac{\pi}{2} - \text{atan}(\frac{1}{x})$

   since $\text{atan}(|x|) = \frac{\pi}{2} - \text{atan}(\frac{1}{|x|})$ and $x > 0$

3. compute $\text{atan}(x)$ in $[0, \tan\frac{\pi}{4} = 1]$
   by:

$$z = x * x$$

$$\text{atan}(x) = x \frac{P_0 + P_1 z + P_2 z^2}{Q_0 + Q_1 z + Q_2 z^2 + Q_3 z^3}$$

P_0 80.78998484078
P_1 72.5814378046
P_2 11.11774163116

Q_0 80.78998647615
Q_1 99.5112677199
Q_2 28.13285868067
Q_3 1.0

```
500 rem y = atan(x)
510 x1 = 1
520 if x < 0 then x1 = -1
530 if x < 0 then x = -x
540 x2 = 0
550 if x > 1 then x2 = 1
560 if x > 1 then x = 1/x
570 y = x*x
580 x3=80.78998+y*(72.58144+y*11.11774)
590 x4=80.78999+y*(99.51128+y*(28.13286+y))
600 y = y*x3/x4
610 if x2 = 1 then y = 1.5707963 - y
620 y = y*x1
630 return
```

Trig Functions

Sine: Compute as cosine using:

$$\sin(x) = \cos(\frac{\pi}{2} - x)$$

Cosine: 1. test arg too large
$|x| > 10^{d-3}$, d = # of digits in floating point
then error

2. $x = |x|$ since $\cos(-x) = \cos(x)$
3. $x = \text{mod}(x, 2\pi)$ since
   $\cos(2\pi + x) = \cos x$. Range is now
   $0 \leqslant x < 2\pi$
4. $0 \leqslant x < \frac{\pi}{2}$: $s = +1$ $\quad x = x$

   $\frac{\pi}{2} \leqslant x < \pi$: $s = -1$ $\quad x = |\pi - x|$

   $\pi \leqslant x < \frac{3\pi}{2}$: $s = -1$ $\quad x = |\pi - x|$

   $\frac{3\pi}{2} \leqslant x < 2\pi$: $s = +1$ $\quad x = |x - 2\pi|$

5. $z = x^2$
   $y = P_0 + P_1 z + P_2 z^2 + P_3 z^3 + P_4 z^4$
   $\cos = y * s$.

P_0 .999999953464
P_1 −.499999053455
P_2 .0416635846769
P_3 −.0013853704264
P_4 .00002315393167

```
500 rem y = sin(x)
510 x = 1.570796 - x

520 rem cos(x)
530 if x < 0 then x = -x
540 if x > 1000 then goto <error>
550 x = x - int(x/6.283185)*6.283185
560 x2 = int(x/1.5707963)
570 x2 = 1
580 if x1 = 1 then x2 = -1
590 if x1 = 2 then x2 = -1
600 if x1 = 1 then x = 3.141593 - x
610 if x1 = 2 then x = 3.141593 - x
620 if x1 = 3 then x = x - 6.283185
630 x1 = x*x
640 y=0.9999999+x1*(-.4999991+x1*(.04166359+x1*(
    -0.001385370+x1*(0.00002315393))))
650 y = y * x2
660 return
```

THE DIGICAST™ PROJECT:

AN IMMEDIATELY VIABLE BROADCAST-BASED INFORMATION UTILITY FOR THE GENERAL PUBLIC

BY JIM C. WARREN, JR.
345 Swett Road
Woodside, CA 94062

This is the abstract of a talk being given before the 2nd Rocky Mountain Symposium on Microcomputers in Pingree Park, Colorado, August 27–29, 1978.

Computers have been consumer products since 1975. Individuals involved in the consumer computer industry estimate that there are 50,000-200,000 general-purpose digital computers installed in peoples' homes. Thus far, these have been exciting, educational, challenging toys—to the extent that they have been used for non-commercial, non-tax-deductable purposes. Their primary use as a consumer product has been in non-arithmetic or minimal-computation applications. One difference between their being an exciting toy, and making them into an obviously useful consumer product, of value to the general public, is the ability to attach them to large data-bases of interest and use to the general public. The Digicast Project proposes to provide one of the first such "attachments" for personal computers.

The single database facility most widely used by the general public is unquestionably the newspaper and newscast. The newspaper provides a large, though certainly limited, database with updating. The newscast provides a much more limited, similar database, with much faster delivery. In both cases, the database is available to the public only in a human-readable form, requiring manual scanning. The newscast has the additional undesirable limitation that it must be monitored in its entirety by a human listener or viewer, and must be monitored at explicit times. The Digicast Project proposes to provide most of the utility of the newspaper; provide much more content than is economically feasible using newsprint; provide the speed of access of a newscast; and provide the data in machine-readable form so that the consumer may use an arbitrarily "intelligent" intellectual assistant—the home computer—in scanning of the incoming data. This will be accomplished at a cost that is acceptable in the consumer marketplace, using well-known, established, reliable technology. The Digicast Project hopes to have at least one facility operational by the end of 1978.

The Project proposes to transmit digitally-encoded alphanumeric information on FM subcarriers. By the laws of physics, every FM broadcast facility has at least two such subcarriers that are usable within the laws of the FCC—the Federal Communication Commission. It has already been proven in commercial broadcasting environments that such transmissions can be reliably conducted at 4800 baud, and some of the engineers involved have indicated that they are certain that trivial hardware modifications could double that transmission rate. SCI Systems, Inc., 8600 S. Memorial Pkwy., Huntsville, AL 35802, has a printer that includes a receiver for an FM digital broadcast using a proprietary encoding scheme developed by SCI engineers in cooperation with personnel from Digital Broadcasting Corp. of McLean, VA. The FCC has already adopted all of the necessary policies to allow such utilization of SCA's—Subcarrier Authorizations. Only straightforward administrative approval is required in order for any licensed FM station to begin transmission. The hardware modifications at the transmitter cost well under $10,000. The receivers—*exclusive* of the receiving computer—are obviously producible for $50-$100. Both transmitter and receiver software can range from quite simple to arbitrarily complex. The most simple software can make the system extremely useful.

The scenarios for utilization are also arbitrarily varied: International, national, and regional news—all of it—can be made as accessible to the general public as it now is only to newsroom editors and reporters. It is delivered to newspapers and broadcasters by wire services, already in machine-readable form. Little additional effort is necessary to inject that data into the transmitter computer for "Digicasting." The consumer, interested in specified topics, need only instruct the receving computer to monitor the local Digicast(s), and save all articles containing specified keywords. Note that the keywords need not appear in a "headline;" they may be anywhere in the article, and will still be noted and the article saved.

Perhaps one of the most useful services that a Digicast can

offer is in the area of "want ads." If one desires a Honda for under $3,000, then one instructs the Digicast receiver in the obvious manner. There is no need to waste time scanning uninteresting want ads. There is also no need to classify the ads—a boon to those selling a variety of unrelated items.

Stock and commodities transactions can be Digicast, requiring little or no rekeying upon their receipt from the wire service. Further, if they are in fixed format, then it would be a simple matter for the receiving computer to feed the data to a stock analysis program, again requiring no rekeying.

The system must be financially viable if it is to be a widespread and useful information utility. It is reasonable to believe that it is financially viable. Consider the income sources of a newspaper or commercial broadcast—commercial advertising, classified advertising, and subscriptions. Classified advertising, sans classification, can be conducted without change from the past as far as payment is concerned, and with vastly greater utility. The counterpart of commercial advertising can be continued in a manner quite analogous to its traditional form. It can be "forced" on the consumer by injecting it in the beginnings of articles. Subscriptions can be implemented through encryption of the data being transmitted and sale of monthly decoding keys that need only be typed in to the receiving computer. One subscriber paying for the key and giving it to several others is no more of a problem than one subscriber to a newspaper giving his copy to someone else.

This abstract provides only the most brief glimpse of digital broadcasting and its potential utility and impact for the society at large. Most of the details of the Digicast Project will be published in the *Intelligent Machine Journal*, a biweekly newspaper oriented to low-cost computing power, edited and published by the author. The *Journal* carries discussion of the design alternatives of the Digicast hardware, software, systems, and protocols, and publishes Digicast standards as they are adopted. Discussion, suggestions, and general dialogue concerning the design of digital broadcasting systems is solicited for publication and distribution in this periodical. Most of the information is thereby explicitly placed in the public domain. Only explicit hardware designs will be proprietary, protected by the traditional patent process, and the names—Digicast and Digicasting—are proprietary and protected by trade mark. Additionally, explicit transmitter system designs will undoubtedly be protected by their creators via trade secret methods.

There are a multitude of FM stations. There are a multitude of interesting and useful applications of the proposed systems design. There is room enough for all. Let us stand on each others' shoulders; not on each others' toes. *

---

* almost-direct quote of Dennis Allison in *Dr. Dobb's Journal*, January, 1976.

Note: Subscriptions to the *Intelligent Machine Journal* are $18/year to U.S. addresses, and may be directed to 345 Swett Rd., Woodside, CA 94062. The author of this paper, the Director of the Digicast Project, will publish all public-domain information regarding the Project in the *Journal*. Personal requests for explicit information will be answered with information about the *Journal*, due to limited time and funds. Correspondence offering suggestions may be published. Correspondence offering experience-based assistance or cooperation is, unfortunately, the only kind that may evoke a personal reply. Please include day and evening telephone numbers in this latter form of communication.

CUTTING STENCILS WITH A DIABLO PRINTER

Dear Dr. Dobb's, *Received: 78 July 7*

These routines allow you to use a Diablo printer to cut an ordinary office stencil, a problem with many small systems printers because the printer head has so little mass, as opposed to ordinary office typewriters. We used no film in front of the stencil, and inserted the typing plate behind the stencil. We then programmed the computer in the system to output the text to the printer as per the attached program routines.

The routine CUTSTEN.LIN just types each line 4 times. This is adequate to cut through the stencil for most letters. The routine CUTSTEN.CHR does better, though; it types heavy characters like H or M more times than light characters like i or l.

Most computers which allow programming of output routines should be similarly adaptable. Many word processing systems will likely require such modification, because the impact printers used (like the Diablo) commonly strike fast but not heavy. Our system is a Cromemco Z-2, and the printer is made by Data Terminals & Communications (San Jose), using a Diablo Hy-Term print mechanism. The routines are written in Sheppardson Basic, operating under Cromemco's CDOS.

Sincerely yours,
Richard French

495 3rd Ave.
San Francisco, CA 94118

```
100 @ : @"::::::::  CUTSTEN: TYPES OUT EDIT FILES ONTO STENCIL  ::::::::" : @
105 REM- VERSION CUTSTEN.LIN:  CUTS STENCIL LINE AT A TIME, USING 4 PASSES.
110 DIM F$(13),A$(79)
120 SET 0,-1 : REM- DISABLES PAGE-WIDTH CHECKING.
130 REM
140 @"PRINTER ON LINE & RESET?"
150 @"PITCH = 10, WHEEL = # 72, RIBBON OFF?"
160 @"STENCIL BACKED BY PLATE, IN POSITION?" : @
170 INPUT"TEXT FILE NAME ? ",F$
180 REM
190 OPEN\1\F$ : REM- OPEN ASCII DISC FILE ON CHANNEL 1.
200 OPEN\4\"ST5" : REM- OPEN DIABLO DRIVER ON CHANNEL 4.
210 OUT%0050%,%0084% : REM- SET BAUD=300 ON TUART (I/O BOARD).
220 REM
230 ON ESC GOTO 350
240 REM- IF USER WANTS ESCAPE GO CLOSE FILES.
250 REM
260 INPUT\1\A$
280 @\4\A$;CHR$(13);A$;CHR$(13);A$;CHR$(13);A$
330 GOTO 260
340 REM
350 CLOSE : @ : END
```

```
100 @ : @"::::::  CUTSTEN: TYPES EDITED ASCII FILE INTO STENCIL  ::::::" : @
110 REM- VERSION CUTSTEN.CHR: CUTS ONE CHARACTER AT A TIME, SEVERAL PASSES.
120 IMODE : REM- ALL VARIABLES ARE INTEGERS.
130 DIM F$(13),A$(79),C$(0)
140 INTEGER T(127)
150 SET 0,-1 : REM- DISABLES PAGE-WIDTH CHECKING.
160 REM
170 @"PRINTER ON LINE & RESET?"
180 @"PITCH = 10, PLOT = ON?"
190 @"WHEEL = #72, RIBBON OFF?"
200 @"STENCIL BACKED BY PLATE, IN POSITION?" : @
210 INPUT"TEXT FILE NAME ? ",F$
220 REM
230 OPEN\1\F$ : REM- OPEN ASCII DISC FILE ON CHANNEL 1.
240 OPEN\4\"ST5" : REM- OPEN DIABLO DRIVER ON CHANNEL 4.
250 OUT%0050%,%0084% : REM- SET BAUD=300 ON TUART (I/O BOARD).
260 @\4\CHR$(6) : REM- ENTER PLOT MODE.  (NO AUTO ADVANCE AFTER CHAR PRINT.)
270 REM
280 ON ESC GOTO 580
290 REM- IF USER WANTS ESCAPE GO CLOSE FILES.
300 REM
310 MAT T=1
320 DATA 3,3,8,9,5,5,2,3,3,4
330 DATA 3,3,2,2,2,2,8,4,5,4,9
340 DATA 5,6,4,9,9,5,5,5,2,3,2
350 DATA 4,6,8,9,5,8,8,7,7,10,10
360 DATA 4,6,10,5,12,10,7,7,10,8
370 DATA 8,7,8,7,11,9,5,5,3,3
380 DATA 3,1,2,3,5,7,5,9,5,6
390 DATA 9,7,4,5,8,4,9,5,5,10
400 DATA 8,3,6,4,5,5,8,6,6,4
410 DATA 3,3,3,3
420  FOR Z=33 TO 126
430  READ D : T(Z)=D : REM- LOAD TABLE WITH DATA.
440  NEXT Z
450 REM
460 INPUT\1\A$ : REM- GET ASCII LINE FROM DISC FILE.
470  FOR X=0 TO(LEN(A$)-1)
480  C$=A$(X,X) : REM- SELECT CHARACTER.
490  N=T(ASC(C$)) : REM- OBTAIN PARALLEL NUMBER FROM TABLE.
500   FOR Y=1 TO N
510   @\4\C$; : REM- PRINT CHARACTER N TIMES.
520   NEXT Y
530  @\4\CHR$(6);CHR$(32);CHR$(6); : REM- PLOT OFF, ONE SPACE, PLOT ON.
540  NEXT X
550 @\4\CHR$(6);CHR$(13);CHR$(10);CHR$(6) : REM- PLOT OFF, CR & LF, PLOT ON.
560 GOTO 460
570 REM
580 @\4\CHR$(7) : REM- TURN PLOT OFF (REGARDLESS OF PREVIOUS ON/OFF STATUS).
590 CLOSE : @ : END
```

POW: An 8080 'Processor of Words'

BY HERMAN WATSON
P.O. Box 341401
Coral Gables, FL 33134

Note: Normally we would have typeset the text of this article, but we reproduce it here as we received it as an example of what this program can do. A sample of the "Edit" or upper case Input File is given for comparison.—TRW

....POW....

POW is a program that is a "Processor Of Words" for the 8080 computer. POW most closely resembles a program called PRINTER written by Clyde Roby. POW has the ability to set margins, tabs, spacing, justification, and indention and in addition provides automatic centering, and automatic paging with titles and numbering.

This program works on an existing file that has been prepared with some type of editor and saved on a mass storage device. I am currently using a modified version of F. J. Greeb's editor that was published in DDJ June/July 1976, along with Peripheral Vision's floppy disc operating system. POW has in the past been used with cassette tape, and by substituting the cassette I/O routines for the FDOS I/O routines, it can easily be converted back.

Commands are freely inserted into the source text when it is prepared. POW will react to those commands when it encounters them while printing out the file. Commands all consist of the same format which is a colon immediately followed by a two letter command name, then a delimiter with optional data for the command, and/or a terminator. Delimiters are either a comma, or a space, and a terminator can be either a comma a space, a carriage return, or nothing. This allows maximum freedom from rigid command syntax. A sample of the command format is shown by the definition of margins:

```
:DM,10,70
```

In addition to the list of commands, POW also has the ability to allow the use of an upper and lower case printer with an upper case only input file. This is accomplished by using special characters representing a single capitalization, a Shift-Lock, and a Down-Shift. These special characters are an ASCII Shift-N and an ASCII Shift-L . One single Shift-N character is used for single capitalization, and two consecutive Shift-N characters are used for Shift-Lock. POW begins in the upper case mode and will remain in it until either a single shift character or a Down-Shift character is encountered. An input file containing both upper and lower case ASCII can be processed without modification in the default mode (Upper Case). Case modifications are only done automatically once the Upper Case mode has been exited.

```
THIS IS THE "EDIT" VERSION OF THE FILE USED FOR
PREPARING THE INSTRUCTIONS.  THIS IS THE TYPE OF FILE
THAT IS INPUT TO POW FOR PROCESSING.


:PT,10,5:PB,10,0
:TM,48,^^POW...-:-
:JT
:P6
:CM,^^....POW....\
:LF,3
:BP,^^POW\ IS A PROGRAM THAT IS A "^PROCESSOR ^OF ^WORDS"
FOR THE 8080 COMPUTER.
^^POW\ MOST CLOSELY
RESEMBLES A PROGRAM CALLED ^^PRINTER\ WRITTEN
BY ^CLYDE ^ROBY.  ^^POW\ HAS THE ABILITY
TO SET MARGINS, TABS, SPACING, JUSTIFICATION, AND
INDENTION AND IN ADDITION PROVIDES
AUTOMATIC CENTERING, AND AUTOMATIC PAGING WITH
TITLES AND NUMBERING.
:BP,^THIS PROGRAM WORKS ON AN EXISTING FILE THAT
HAS BEEN PREPARED WITH SOME TYPE OF EDITOR
AND SAVED ON A MASS STORAGE DEVICE.  ^I AM CURRENTLY
USING A MODIFIED VERSION OF ^F. ^J. ^GREEB'S
EDITOR THAT WAS PUBLISHED IN ^^DDJ J\UNE/^JULY
1976, ALONG WITH ^PERIPHERAL ^VISION'S FLOPPY DISC
OPERATING SYSTEM.
^^POW\ HAS IN THE PAST BEEN USED WITH CASSETTE
TAPE, AND BY SUBSTITUTING THE CASSETTE ^^I/O\
ROUTINES FOR THE ^^FDOS I/O\ ROUTINES, IT
CAN EASILY BE CONVERTED BACK.
:BP,^COMMANDS ARE FREELY INSERTED INTO THE SOURCE
TEXT WHEN IT IS PREPARED.  ^^POW\ WILL
REACT TO THOSE COMMANDS WHEN IT ENCOUNTERS THEM
WHILE PRINTING OUT THE FILE.
 ^COMMANDS ALL CONSIST OF THE SAME FORMAT WHICH IS
A COLON IMMEDIATELY FOLLOWED BY A TWO LETTER
COMMAND NAME, THEN A DELIMITER WITH OPTIONAL
DATA FOR THE COMMAND, AND/OR A TERMINATOR.
 ^DELIMITERS ARE EITHER A COMMA, OR A SPACE,
AND A TERMINATOR CAN BE EITHER A COMMA A SPACE, A
CARRIAGE RETURN, OR NOTHING.  ^THIS ALLOWS MAXIMUM
FREEDOM FROM RIGID COMMAND SYNTAX.  ^A SAMPLE
OF THE COMMAND FORMAT IS SHOWN BY THE DEFINITION OF
MARGINS::LF,1
:CM,:^^DM\,10,70
:BP,^IN ADDITION TO THE LIST OF COMMANDS, ^^POW\ ALSO
HAS THE ABILITY TO ALLOW THE USE OF AN UPPER AND
◆
```

The single shift command (Shift-N) will automatically capitalize only the character that immediately follows the command. For example, " pow" would be printed as "Pow". The Shift-Lock command locks the carriage up, until the Shift-Down command is given, so for example, " pow " would become "POW". Capitalization commands work in all areas of POW. Page title capitalization uses the existing capitalization mode, but will not change the present mode if capitalization is specified in the title. This means that when a new page begins with the case set in the lower case mode, even if a Shift-Lock command occurs in the title, once the printing of the title is over and the text is resumed, the lower case mode will be restored.

PROGRAM STRUCTURE

Pow starts with a greeting message containing instructions, and then goes to the "MAIN:" area to get a command from the console. The "L"-Load command reads all of the input file into memory and returns to the "MAIN" area. When the "P"-Print command is given, control is transferred to the principal loop of the program, starting at "PRIN:". "PRIN" then gets a character from memory and checks to see if it is the beginning of a command. If it is, control is passed to the area called "CMND:" and the command table "CTAB:" is searched for a match. If a match is found, then a subroutine call is done to the command routine with a return to "PRIN".

If the character is not the beginning of a command, it is then checked to see if it is a capitalization command. If it is, the routine "CAPR:" handles the case processing. The carry is set on return from "CAPR" to indicate whether the present character is to be ignored or not.

If the character is not a capitalization command, it is then checked to see if it is a Shift-K. This character is used to indicate that the character immediately following the Shift-K is to be output without any processing.

"PRIN" then checks to see if the justification is on or off. The routine "UFMT:" is used when there is no justification, and the routine "FMAT:" is used for both left and total justification.

The routine "UFMT" will output the text as specified in the input file, honoring all carriage returns as the end of the line. It will in addition force a carriage return if the text tries to go past the right margin.

The routine "FMAT" outputs text either left justified (against the left margin, with as many words on the line as possible) or total justified with spaces filled. Total justification begins as left justification, and then randomly begins filling with spaces until the line is completely filled between the left and right margins.

In either the left or total justification mode, all carriage returns are converted to a space, and the carriage returns that get printed are computed by POW.

It should be noted that POW ignores line feed characters, and will work with a text file that either contains them or not. If vertical spacing between lines is needed, it should be specified with the :LF command, and not with multiple line feeds or carriage returns.

There are three areas where the text is taken from memory and used in the program. The main one is at "PRIN" itself, and the other two are for the page titles and the centering commands. All output goes through "OBUF" except page titles. The printing of page titles is done directly to the printer without any use of the "OBUF" buffer. Title printing is suppressed until page two is reached. This allows the printout of the first page without a title.

Since I am using this program with my Selectric printer, I have included the output routines used for the printer. The conversion table is included, but might be different for other machines. The machine is a Datel 1030 terminal. POW itself is written to run strictly with ASCII code, and as I have demonstrated, conversion can be accomplished either on input or output to other codes if necessary.

POW has the provision for stopping the printer and waiting for the paper to be changed. Pressing the "Attention" button allows POW to continue with the next page. The area for that is at "DVDR:", and uses the code for Attn on my printer. This section can be changed to ring the bell and wait for a press of return on an ASCII terminal.

The usual proceedure for setting up a file that will be output with POW would be to first define the margins, then the top of the page length, with the line for the title, then the text length of the page, and the length of the bottom of the page. The title can be put on the bottom or the top of the page. The sum of the top of the page, the text length, and the bottom of the page length should be 65 for a Selectric with a Pitch of 10, or the same for a Teletype. Next, a page number of zero is specified, then the top or bottom message is defined. The justification mode is chosen, and then the tabs should be defined. The paragraph break can finally be defined, and the last command before the beginning of the text should probably be ":PG" which forces an immediate start of page number one. (page number goes from zero to one) The order that these commands are given is not critical, nor is it necessary to include them all. POW has all of the critical commands set to default values, and new specifications are necessary only if the default values are unsatisfactory. The ":PG" command is used, because it tells POW when it is starting the first page of the file.

With this brief description, the source listing should be pretty well self explanatory.

COMMAND SUMMARY AND DEFAULT VALUES

:DM,10,70 Define margins, left margin, right margin

:DT,15,22,30,45 Define tabs up to a maximum of 14 tabs. Tabs must be in ascending order and no error checking is done on them.

:PL,45 Define page length (the text portion) to be 45 lines long.

:JT Begin total justification mode with automatic space filling.

:JE Close present line and end all justification

:JL Begin left only justification mode.

:CT,Tab,Text Center the text about the tab that is specified (Note, the line is not printed after this command)

:CC Close Centering command MUST be issued after the last :CT command. This closes the output line.

:LF,Numb Line Feed the paper up "Numb" times (Index on Selectrics). This is really a CR, LF sequence to printer "Numb" times.

:BP Break for a new paragraph. Since carriage returns are converted to spaces during either total or left justification, it is necessary to be able to specify the beginning of a new paragraph while in these modes. This is done with this command. The printer closes the present line, then does the specified number of line feeds, and then indents to the specified tab. The indention is automatically set to the left margin when the margins are defined, and is also set with the next command.

:DB,1,15 Define the paragraph break to be a number of line feed's and an indention to a specific tab position (default is LF=1 and tab set to the left margin). The indention can be set at any place either to the right or left of the left margin.

:NP New Page forces the close of the present page, feeds it out of the printer, and then begins a new page.

:CM,Text Automatically centers the text between the left margin and right margin. This command does NOT need to be closed with a :CC command.

:PN,0 Set the page number to the specified value (max is 256)

:SP,1 Set the spacing. If single spacing is desired, the value would be 1, if triple spacing is desired, the value would be 3.

:PT,LF,Numb Define the top of the page to be LF (10) spaces long, with the title occurring on line "Numb". if "Numb" is zero, no message is printed. LF can also be zero for a page of complete text.

:PB,LF,Numb Define the bottom of the page.

:TM,Tab,Text Define what the top of page message will be. It will be printed "Tab" positions over, and "Text" will be the message. If in a colon is found while printing "Text", the present page number will be substituted for the colon. Example:
":TM,45,PAGE-:-" would cause the top message to be printed at tab 45 as "PAGE-4-" when page 4 is printed.

:BM,Tab,Text Define what the bottom of page message will be.

:PG Will immediately force the beginning of a new page (it will not finish the present page). This command should be given at the beginning of the printing of a new text file.

:OF This command will turn off the high quality printer but will continue processing of the text as if it were on.

:ON This command will turn back on the printer after it has been turned off with the above command. this allows sections of a long file to be skipped over after an error has been found and corrected, allowing printing to be resumed after the error is corrected.

FDOS ROUTINES USED

CI: Console input routine. When this routine returns, the accumulator contains the character typed on the console. If no key was struck, the CPU will wait in CI until one is struck.

CO: Console output routine. The ASCII character in the C register is output to the console printer. On return, both the A and C register contain the character.

CSTS: Console status check. If the console has had a key struck, the accumulator returns with the value 0FFH. If no key has been struck, the accumulator returns with the value 00H.

RESTRT: This is the address that restarts the FDOS monitor.

TXTYP: This routine types the text string pointed to by the HL register. The routine types all characters, and the last character typed will have the most significant bit set. This terminates the routine.

TXTIN: This routine is used to get the file name that is to be opened.

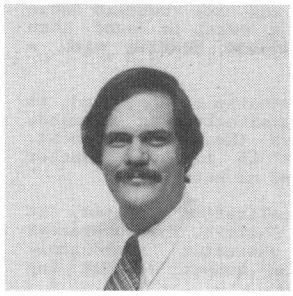
REDFIL: This is the address of the buffer pointer that will be used for input from an FDOS file.

FILNAM: This routine checks the syntax of the file name input with txtin.

LOKFIL: This routine looks in the FDOS directory to see if the specified file exists.

OPNIN: This is the routine that opens the FDOS file for sequential input.

FDREAD: This routine is a functional equivalent to CI: except it inputs the next character from the open FDOS file. If the end of the input file has been reached, the routine returns with the carry flag set.



Herman Watson received his Bachelor of Science in Zoology from the University of Florida at Gainesville, in 1969 and his Master of Science in Electrical Engineering from the University of Miami in 1972. He is a Biomedical Engineer with the Division of Pulmonary Diseases, Mount Sinai Medical Center, Miami Beach, Florida. Some of his research interests include modeling and analysis of the Pulmonary system, Ambulatory Monitoring, and EEG analysis.

```
                ..FOW...
                PROCESSOR OF WORDS FOR 8080
                DEC 16-30, 1977 :
                MODIFIED FOR FDOS JUN 3, 1978
                SELECTRIC MODS JUNE 15, 1978
                TOTAL JUST FIXED JULY 20, 1978

                BY HERMAN WATSON
                P.O. BOX 341401
                CORAL GABLES, FLA.
                33134

                THE FOLLOWING IS THE JAZZED UP VERSION FOR FDOS
                WITH INSTRUCTION PRINTOUT AT BEGINNING

                THIS PROGRAM ASSUMES STACK POINTER
                IS ALREADY SET (IN THIS CASE IT
                USES THE STACK AREA OF FDOS)

0000 21 0009    START:  LXI     H,STMSG
0003 CD 0073            CALL    TXTYP
0006 C3 0478            JMP     MAIN

0009 2E2E2E5052 STMSG:  .ASCII  /...PRO\
000E 4F4345535 \CESSO\
0013 4F52204F46 \R OF /
0018 20574F5244 \WORDS\
001D 532046F52  \ FOR /
0022 2038303830 \8080 \
0027 0DA544845  \THE  \
002C 20434F4D4D \COMM\
0031 414E445320 \AND A\
0036 415245241  \RE AS\
003B 532046F4C  \ FOLL\
0040 4C4F5753   \OWS  \
0044 0DA202022  \"P   \
0049 5022203D3E \" =>\
004E 20505249E  \PRINT\
0053 54         /       \
0054 0DA202022  \"L   \
0059 4C222D3E   \" =>\
005E 204C4F4144 \LOAD \
0063 0DA202022  \"O   \
0068 5122203D3E \" =>\
006D 20515549   \QUIT \
0072 0DA202028  \(C   \
0077 435452C2D  \TRL-X\
007C 582057494C \ WILL\
0081 4C2041424F \ ABOR\
0086 525420502  \T PRO\
008B 4F47524141 \GRAM)\
0090 29         /       \
0091 0DA504C45  \PLEA \
0096 4153452045 \SE EN\
009B 4E5445522  \TER Y\
00A0 594F555220 \OUR C\
00A5 434F4D4D41 \OMMAN\
00AA 4EC4       \D/
```

```
                        ; UTILITY ROUTINES
                        ; ADD A TO HL
00AC  85        ADAH:   ADD     L
00AD  6F                MOV     L,A
00AE  D0                RNC
00AF  24                INR     H
00B0  C9                RET
                        ; TEST DE .EQ. HL
                        ; RETURN ZERO IF SO
00B1  7A        TDHE:   MOV     A,D
00B2  BC                CMP     H
00B3  C0                RNZ
00B4  7B                MOV     A,E
00B5  BD                CMP     L
00B6  C9                RET
                        ;
                        ;
                        ;GENERATE RANDOM NUMBER 0-15
00B7  21 0707   RAND:   LXI     H,RNDV
00BA  7E        RND1:   MOV     A,M
00BB  07                RLC
00BC  3C                INR     A
00BD  07                RLC
00BE  07                RLC
00BF  AE                XRA     M
00C0  77                MOV     M,A
00C1  E60F              ANI     0FH
00C3  C9                RET
                        ;
                        ;
                        ; CONVERTS TABS TO CORRECT POSN
00C4  3A 072E   TBST:   LDA     LPOS
00C7  4F                MOV     C,A
00C8  11 070E           LXI     D,TTAB
                        ; SEARCH TTAB FOR NEXT GREATEST LOCATION
00CB  1A        TBLP:   LDAX    D
00CC  13                INX     D
00CD  B9                CMP     C
00CE  CA 00CB           JZ      TBLP
00D1  D2 00DC           JNC     GOTB
00D4  FE00              CPI     0
00D6  C2 00CB           JNZ     TBLP
                        ; NO MORE TABS IN TABLE
00D9  C3 02BD           JMP     CLOS
                        ; GOT GOOD TAB, UPDATE POINTERS
00DC  21 070D   GOTB:   LXI     H,RMAR
00DF  BE                CMP     M
00E0  D2 02BD           JNC     CLOS
00E3  32 072E   PUTB:   STA     LPOS
00E6  21 0757           LXI     H,OBUF
00E9  CD 00AC           CALL    ADAH
00EC  22 0731           SHLD    LADR
00EF  C9                RET
                        ;
                        ; SBR FOR ASCII=>BINARY
                        ; ADDR IN HL ON ENTRY, SAVED IN APNT ON EXIT
                        ; RETURN W/ VALUE IN HL
00F0  E5        ADEC:   PUSH    H
00F1  C1                POP     B
00F2  21 0000           LXI     H,0
00F5  0A        ADE1:   LDAX    B
00F6  CD 0113           CALL    NMCK
00F9  DA 010C           JC      ADE2
00FC  03                INX     B
00FD  54                MOV     D,H
00FE  5D                MOV     E,L
00FF  29                DAD     H
0100  29                DAD     H
0101  19                DAD     D
0102  29                DAD     H
0103  D630              SUI     48
0105  5F                MOV     E,A
0106  1600              MVI     D,0
0108  19                DAD     D
0109  C3 00F5           JMP     ADE1
010C  C5        ADE2:   PUSH    B
010D  E3                XTHL
010E  22 0735           SHLD    APNT
0111  E1                POP     H
0112  C9                RET
                        ; CHECK FOR DEC NMBR
0113  FE30      NMCK:   CPI     '0'
0115  D8                RC
0116  FE3A              CPI     '9'+1
0118  3F                CMC
0119  C9                RET
                        ;
                        ;
                        ; SBR INITIALIZES OBUF FOR
                        ; NEW FORMATTED LINE
011A  21 0757   NEWL:   LXI     H,OBUF  ;FILL WITH SPACES
011D  3A 0737           LDA     MAXL
0120  4F                MOV     C,A
0121  3E20              MVI     A,' '
0123  77        NEWA:   MOV     M,A
0124  23                INX     H
0125  0D                DCR     C
0126  C2 0123           JNZ     NEWA
0129  21 0757           LXI     H,OBUF  ;COMPUTE LEFT MAR ADDR
012C  3A 070C           LDA     LMAR
012F  CD 00AC           CALL    ADAH
0132  22 0731           SHLD    LADR
0135  3A 070C           LDA     LMAR    ;SET POSITION COUNTER
0138  32 072E           STA     LPOS
013B  21 0757           LXI     H,OBUF  ;COMPUTE RIGHT MAR ADDR
013E  3A 070D           LDA     RMAR
0141  CD 00AC           CALL    ADAH
0144  22 0733           SHLD    LEND
0147  360D              MVI     M,CR    ;EOL AT RIGHT
0149  C9                RET
                        ;
                        ;SBR FOR FORMATTED OUTPUT
                        ; CALL W/ LETTER IN A
                        ; HANDLES LEFT OR TOTAL JUST
014A  FE09      FMAT:   CPI     9       ;TEST TAB
014C  CA 00C4           JZ      TBST
014F  FE0D              CPI     CR      ;TEST CR
0151  C2 0156           JNZ     LFTS
0154  3E20              MVI     A,' '   ;REPLACE WITH SPACE
0156  FE0A      LFTS:   CPI     LF      ;IGNORE LF'S
0158  C8                RZ
0159  FE20              CPI     ' '
015B  C2 0168           JNZ     RPMT
                        ; HERE, EITHER LEFT OR TOTAL JUST, SO
                        ; ALLOW NO SPACES AT LEFT OF LINE
015E  21 070C           LXI     H,LMAR
0161  3A 072E           LDA     LPOS
0164  BE                CMP     M
0165  C8                RZ              ;AT START, SO STAY THERE
0166  3E20              MVI     A,' '   ;OK TO KEEP SPACE
0168  2A 0731   RPMT:   LHLD    LADR    ;NOW PLACE LETTER IN OBUF
016B  77                MOV     M,A
016C  23                INX     H
016D  22 0731           SHLD    LADR
0170  21 070D           LXI     H,RMAR  ;CHECK IF OBUF FULL
0173  3A 072E           LDA     LPOS
0176  3C                INR     A
0177  32 072E           STA     LPOS
017A  BE                CMP     M
017B  D8                RC
                        ; OBUF FULL, ASSUME LEFT JUST
                        ; BACK UP TO SPACE AND SAVE OVERFLOW
                        ; DE=TEMP ADDR
```

```
                        ; C = CHAR COUNT
                        ; HL=OBUF ADDR
017C  061E                      MVI     B,30    ;MAX AMOUNT TEML CAN HOLD
017E  0E00                      MVI     C,0     ;TEML CHAR COUNT
0180  11 0739                   LXI     D,TEML
0183  2A 0731                   LHLD    LADR
0186  360D                      MVI     M,CR    ;EOL IN CASE NOT POSSIBLE
                        ; LOOP TO BACK UP TO FIRST SPACE (WITHIN 30 LETTERS
                        ; AND WITHOUT HITTING LEFT MARGIN)
0188  2B                LJBU:  DCX     H
0189  7E                      MOV     A,M
018A  FE20                      CPI     ' '
018C  CA 0199                   JZ      LJFN
018F  12                LJRT:  STAX    D
0190  13                      INX     D
0191  0C                      INR     C
0192  05                      DCR     B
0193  CA 022F                   JZ      OUTL
0196  C3 0188                   JMP     LJBU
0199  2B                LJFN:  DCX     H
019A  7E                      MOV     A,M
019B  23                      INX     H
019C  FE20                      CPI     ' '
019E  CA 018F                   JZ      LJRT
01A1  79                      MOV     A,C
01A2  32 0738                   STA     TCNT    ;SAVE CHAR COUNT
01A5  3A 072E                   LDA     LPOS    ;BACK UP LPOS
01A8  91                        SUB     C
01A9  32 072E                   STA     LPOS
01AC  4F                        MOV     C,A     ;DON'T GO PAST NEW PARA TAB
01AD  3A 0722                   LDA     BRTB    ;FOR UNMODIFIED INDENTION
01B0  91                        SUB     C       ;TO THAT TAB POSITION
01B1  D2 022F                   JNC     OUTL    ;TERMS NOT MET
01B4  22 0731                   SHLD    LADR
01B7  360D                      MVI     M,CR    ;NEW EOL
                        ;
                        ; TEST HERE IF LEFT JUST OR TOTAL
                        ; 0=>NO JUST
                        ; 1=>LEFT ONLY
                        ; 2=>TOTAL JUST
                        ; BY THE WAY, AT THIS POINT LEFT JUST
                        ; IS COMPLETE
01B9  3A 0730                   LDA     JFLG
01BC  FE01                      CPI     1
01BE  CA 022F                   JZ      OUTL
01C1  FE02                      CPI     2
01C3  C2 022F                   JNZ     OUTL
                        ; TOTAL JUST AT THIS POINT
                        ; MOVE LINE TO RIGHT, AND PAD
                        ; IF LMAR REACHED, PUSH LINE LEFT AND
                        ; PAD AGAIN UNTIL DONE
                        ; (HL)=>(DE), SO DE IS RIGHT OF HL, AND WHEN
                        ; DE.EQ.HL, THE PADDING IS DONE
                        ;*****
                        ; BEGIN AT LAST CHAR AND SHOVE RIGHT
                        ; START PADDING AT THE SELF SPACE
                        ; (SELF IS RANDOM) AND CONTINUE
                        ; PADDING EACH OCCURRANCE OF A GROUP
                        ; OF SPACES UNTIL THE TWO POINTERS
                        ; ARE EQUAL TO EACH OTHER
                        ;
                        ;
01C6  CD 00B7           TOTL:  CALL    RAND    ;INIT SELF
01C9  32 0706                  STA     SELF
01CC  2A 0733                  LHLD    LEND
01CF  EB                       XCHG
01D0  2A 0731                  LHLD    LADR
01D3  3A 072E                  LDA     LPOS
01D6  4F                       MOV     C,A
01D7  0600                     MVI     B,0     ;MAKE SURE SP IS FOUND
                        ; DE.EQ.HL MEANS PAD DONE
01D9  CD 00B1           TDHE:  CALL    TDHE
01DC  CA 022F                  JZ      OUTL
                        ; RIGHT AND PAD
01DF  7E                RITE:  MOV     A,M     ;PICK UP FROM LEFT
01E0  12                       STAX    D       ;STORE AT RIGHT
01E1  FE20                     CPI     ' '     ;TEST PICKED UP CHAR
01E3  C2 0208                  JNZ     WORD
01E6  0601                     MVI     B,1     ;NOTE THAT SP WAS FOUND
01E8  3A 0706                  LDA     SELF    ;TST IF WE CAN INSERT YET
01EB  B7                       ORA     A       ;TEST FOR ZERO
01EC  CA 01F6                  JZ      PADD    ;CAN'T INSERT YET
01EF  3D                       DCR     A
01F0  32 0706                  STA     SELF
01F3  C3 0208                  JMP     WORD    ;NOT YET
01F6  3A 0708           PADD:  LDA     LCHR    ;CHECK IF GROUP OF SPACES
01F9  FE20                     CPI     ' '     ;DON'T ALLOW THIS COND.
01FB  CA 020C                  JZ      WRD1
01FE  3E20                     MVI     A,' '   ;PADDING DONE HERE
0200  1B                       DCX     D
0201  12                       STAX    D
0202  CD 00B1                 CALL    TDHE
0205  CA 022F                  JZ      OUTL
0208  7E                WORD:  MOV     A,M
0209  32 0708                  STA     LCHR
020C  1B                WRD1:  DCX     D       ;REST OF RIGHT AND PAD LOOP
020D  2B                       DCX     H
020E  0D                       DCR     C
020F  3A 0722                  LDA     BRTB    ;ALLOWS INDENTION
0212  B9                       CMP     C
0213  CA 021D                  JZ      LEFT    ;HIT INDENTION
0216  3A 070C                  LDA     LMAR    ;OR LEFT MARGIN
0219  B9                       CMP     C
021A  C2 01DF                  JNZ     RITE    ;CAN STILL PROCEED
                        ; PUSH LEFT AND TRY AGAIN
021D  13                LEFT:  INX     D
021E  23                       INX     H
021F  0C                       INR     C
0220  1A                       LDAX    D
0221  77                       MOV     M,A
0222  FE0D                     CPI     CR
0224  C2 021D                  JNZ     LEFT
0227  AF                       XRA     A       ;TEST IF ONE SP FOUND
0228  B0                       ORA     B
0229  CA 022F                  JZ      OUTL    ;NOPE, NOT ONE SP FOUND
022C  C3 01DF                  JMP     RITE
                        ; OUTPUT A COMPLETE FORMATTED LINE FROM OBUF
022F  21 0757           OUTL:  LXI     H,OBUF  ;OUT TO CR
0232  7E                OUTM:  MOV     A,M
0233  23                      INX     H
0234  FE0D                     CPI     CR
0236  CA 023F                  JZ      EOL
0239  CD 07E2                  CALL    OUTC
023C  C3 0232                  JMP     OUTM
023F  CD 011A           EOL:   CALL    NEWL    ;CLEAN OBUF
0242  3A 071F                  LDA     SPAS    ;PROCESS SPACING
0245  4F                       MOV     C,A
0246  CD 04CA           EOLP:  CALL    CRLF
0249  3A 072F                  LDA     PPOS    ;UPDATE TEXT PAGE POSITION
024C  3C                       INR     A
024D  32 072F                  STA     PPOS
0250  21 0720                  LXI     H,PLEN
0253  BE                       CMP     M
0254  D2 025E                  JNC     EOXP
0257  0D                       DCR     C
0258  C2 0246                  JNZ     EOLP
025B  C3 0261                  JMP     RSTR
025E  CD 02CE           EOXP:  CALL    NEXP    ;NEED A NEW PAGE!
                        ; RESTORE OVERFLOW FROM TEML
                        ; INTO OBUF STARTING AT LMAR
0261  11 0738           RSTR:  LXI     D,TCNT
0264  1A                       LDAX    D
```

```
0285  B7              ORA     A
0286  C8              RZ
0287  21 0739         LXI     H,TEML-1
028A  4F              MOV     C,A
028B  0D              DCR     C
028C  CD 0DAC         CALL    ADRH
028F  7E              MOV     A,M
0290  FE20            CPI     ' '         ;PREVENT FIRST BEING SPACE
0292  79              MOV     A,C
0293  12              STAX    D
0294  CA 02A1         JZ      RSTR
                ; FIRST NOT SPACE
0297  0C              INR     C
0298  3A 070C         LDA     LMAR
029B  81              ADD     C
029C  32 072E         STA     LPOS
029F  EB              XCHG
02A0  2A 0731         LHLD    LADR
02A3  1A        RSTL: LDAX    D
02A4  77              MOV     M,A
02A5  1B              DCX     D
02A6  23              INX     H
02A7  0D              DCR     C
02A8  C2 02A3         JNZ     RSTL
02AB  22 0731         SHLD    LADR
02AE  AF              XRA     A
02AF  32 0738         STA     TCNT
02B2  C9              RET

                ;
                ; UNFORMATTED OUTPUT ROUTINE
                ; EQUIV TO FMAT, BUT NO JUST
                ; IF TEXT EXCEEDS RMAR, THEN A CR IS FORCED AND A
                ; NEW LINE IS STARTED
02B3  FE09      UFMT:   CPI     9           ;TEST TAB
02B5  CA 00C4           JZ      TBST
02B8  FE0D              CPI     CR          ;TEST CR
02BA  2A 0731           LHLD    LADR
02BD  77                MOV     M,A         ;INSERT AS EOL
02BE  CA 02CF           JZ      OUTL
02C1  FE0A              CPI     LF          ;IGNORE LF'S
02C3  C8                RZ
02C4  23                INX     H
02C5  22 0731           SHLD    LADR
02C8  21 070D           LXI     H,RMAR      ;TEST IF FULL
02CB  3A 072E           LDA     LPOS
02CE  3C                INR     A
02CF  32 072E           STA     LPOS
02D2  3D                DCR     A
02D3  BE                CMP     M
02D4  D8                RC
                ; HERE, OBUF IS FULL SO FORCE CR AND CONTINUE
02D5  2A 0731           LHLD    LADR
02D8  360D              MVI     M,CR
02DA  C3 02CF           JMP     OUTL
                ; SBR TO OUTPUT OBUF IF ANYTHING IN IT
02DD  21 070C   CLOS:   LXI     H,LMAR
02E0  3A 072E           LDA     LPOS
02E3  BE                CMP     M
02E4  C8                RZ
02E5  D8                RC
02E6  2A 0731           LHLD    LADR
02E9  360D              MVI     M,CR
02EB  C3 02CF           JMP     OUTL
                ; SBR TO OUTPUT BOTTOM OF PAGE, THE DIVIDER, AND
                ; THE TOP OF THE NEXT PAGE
02EE  3A 072D   NEXP:   LDA     BLEN        ;GET BOTTOM LENGTH
02F1  B7                ORA     A
02F2  CA 02E9           JZ      DVDP        ;NO BOTTOM PLEASE
02F5  4F                MOV     C,A
02F6  3A 0726   NFBL:   LDA     BOTN        ;CHECK IF AT LINE FOR MSG OU
                TPUT
02F9  B9                CMP     C
02FA  CC 031A           CZ      BMSG        ;YES, OUTPUT IT THEN
02FD  CD 04CA           CALL    CRLF
0300  0D                DCR     C
0301  C2 02F6           JNZ     NFBL
                ; DETERMINE IF LAST PAGE IN FILE
0304  3A 0702           LDA     EOT
0307  B7                ORA     A
0308  C8                RZ                  ;ZERO SAYS EOF ENCOUNTERED
                ;START A NEW PAGE NOW
                ;OUTPUT ATTN TO SELECTRIC AND
                ;WAIT FOR ACK BEFORE MORE PRINT
0309  3E7C      DVDP:   MVI     A,7CH
030B  CD 08B1           CALL    SELO
030E  CD 08EA   INQP:   CALL    SELI
0311  FE7C              CPI     7CH
0313  C2 030E           JNZ     INQP            ;NOW READY YET
0316  3E34              MVI     A,34H
0318  CD 08B1           CALL    SELO
031B  21 0724           LXI     H,PAGN
031E  34                INR     M
031F  3A 0723   NEWP:   LDA     TLEN        ;GET TOP LEN
0322  B7                ORA     A
0323  CA 0335           JZ      NPXT        ;NO TOP PLEASE
0326  4F                MOV     C,A
0327  3A 0725   NPTL:   LDA     TOPN        ;EQUAL LINE FOR MSG OUTPUT?
032A  B9                CMP     C
032B  CC 035C           CZ      TMSG        ;YES, THEN PRINT IT
032E  CD 04CA           CALL    CRLF
0331  0D                DCR     C
0332  C2 0327           JNZ     NPTL
0335  AF        NPXT:  XRA     A
0336  32 072F           STA     PPOS
0339  C9                RET
033A  3A 0724   BMSG:   LDA     PAGN        ;NO MSG ON FIRST PAGE
033D  FE02              CPI     2
033F  D8                RC
0340  79                MOV     A,C
0341  32 0704           STA     TSTG        ;SAVE PRESENT LINE COUNT
0344  2A 0729           LHLD    BOTA        ;GET BOTM MSG ADDRESS
0347  3A 072C           LDA     BOTT        ;AND TAB POSITION
                ; THIS DIRECTLY OUTPUTS MESSAGE LINE TO PRINTER
                ; IT DOES NOT USE OR DESTROY OBUF AND ITS CONTENTS
034A  4F        TNTR:   MOV     C,A
034B  3A 0705           LDA     CASE
034E  F5                PUSH    PSW
034F  3E20      BMAL:   MVI     A,' '       ;SPACE OVER TO TAB
0351  CD 07E2           CALL    OUTC
0354  0D                DCR     C
0355  C2 034F           JNZ     BMAL
0358  7E        BMSL:   MOV     A,M         ;OUTPUT THE MESSAGE
0359  23                INX     H
035A  CD 03E1           CALL    CAPR        ;DO CASE PROCESSING
035D  DA 0358           JC      BMSL        ;IGNORE LAST CHAR
0360  FE3A              CPI     ':'         ;SUBSTITUTE THE PAGE NUMBER
0362  CC 036F           CZ      BIND        ;AT OCCURANCE OF COLON
0365  FE0D              CPI     CR
0367  CA 0370           JZ      BMXT
036A  CD 07E2           CALL    OUTC
036D  C3 0358           JMP     BMSL
0370  32 072F   BMXT:   STA     PPOS
0373  F1                POP     PSW
0374  32 0705           STA     CASE
0377  3A 0704           LDA     TSTG        ;RESTORE LINE COUNT
037A  4F                MOV     C,A
037B  C9                RET
                ; OUTPUT TOP OF PAGE MESSAGE (SEE BMSG)
037C  3A 0724   TMSG:   LDA     PAGN
037F  FE02              CPI     2
0381  D8                RC
0382  79                MOV     A,C
0383  32 0704           STA     TSTG
0386  2A 0727           LHLD    TOPA
```

```
0369  3A 072B           LDA     TOPI
036C  C3 032A           JMP     TNTP
                ; BINARY TO DECIMAL CONVERT CONTENTS OF A REG
                ; DIRECT OUTPUT TO PRINTER, COMPLETE WITH ZERO
                ; SUPRESSION
036F  1E00      BIND:   MVI     E,0
0371  3A 0724           LDA     PAGN
0374  0E64              MVI     C,100
0376  CD 0381           CALL    BIDA
0379  0E0A              MVI     C,10
037B  CD 0381           CALL    BIDA
037E  C630              ADI     '0'
0380  C9                RET
0381  062F      BIDA:   MVI     B,'0'-1
0383  04                INR     B
0384  91                SUB     C
0385  D2 0383           JNC     BIDA+2
0388  81                ADD     C
0389  57                MOV     D,A
038A  78                MOV     A,B
038B  FE30              CPI     '0'
038D  C2 0394           JNZ     BINZ
0390  7B                MOV     A,E
0391  B7                ORA     A
0392  7A                MOV     A,D
0393  C8                RZ
0394  1C        BINZ:   INR     E
0395  78                MOV     A,B
0396  CD 07E2           CALL    OUTC
0399  7A                MOV     A,D
039A  C9                RET
                ; COMMAND DECODER AND PRINT LOOP
                ; THIS IS THE MAIN TEXT AND WORD PROCESSING LOOP
                ; HERE, WE GET THE NEXT CHAR FROM TEXT AND SEE
                ; IF A COMMAND, OR PROCESSED TEXT
039B  2A 0735   PRIN:   LHLD    APNT    ;SOURCE TEXT POINTER
039E  7E        PRLP:   MOV     A,M
039F  23                INX     H
03A0  FE03              CPI     3
03A2  DA 03CF           JC      EOF
03A5  FE3A              CPI     ':'     ;COLON=BEGINNING OF COMMAND
03A7  CA 0428           JZ      CMND
03AA  CD 03E1           CALL    CAPR    ;DO CASE PROCESSING
03AD  DA 039E           JC      PRLP    ;LETTER WAS TO BE IGNORED
                ;
03B0  47                MOV     B,A
                ; TEST IF DIRECT OUTPUT OF NEXT CHAR
03B1  FE5B              CPI     '['     ;SHIFT-K?
03B3  C2 03B8           JNZ     PROC    ;NO
03B6  46                MOV     B,M     ;OUTPUT WITHOUT QUESTION
03B7  23                INX     H
03B8  3A 0730   PROC:   LDA     JFLG    ;PROCESS A LETTER
03BB  B7                ORA     A
03BC  78                MOV     A,B
03BD  22 0735           SHLD    APNT
03C0  CA 03C9           JZ      NOFM
03C3  CD 014A           CALL    FMAT    ;EITHER LEFT OR TOTAL JUST
03C6  C3 039B           JMP     PRIN
03C9  CD 0293   NOFM:   CALL    UFMT    ;NO JUSTIFICATION
03CC  C3 039B           JMP     PRIN
03CF  CD 02BD   EOF:    CALL    CLOS    ;CLOSE PENDING LINE
03D2  AF                XRA     A
03D3  32 0702           STA     EOT
03D6  CD 065D           CALL    NPAG
03D9  3EFF              MVI     A,0FFH
03DB  32 0702           STA     EOT
03DE  C3 0478           JMP     MAIN
                ; CASE PROCESSING SUBROUTINE
                ;  RETURN WITH CARRY SET IF CHAR IS TO BE
                ; IGNORED (I.E. WAS A SHIFT COMMAND)
                ;
                ; CHECK IF SHIFT OR UNSHIFT COMMAND

03E1  FE5E      CAPR:   CPI     5EH     ;SHIFT N
03E3  CA 0414           JZ      UCAS
03E6  FE5C              CPI     5CH     ;SHIFT L
03E8  CA 0410           JZ      LCAS
                ; CHECK AND PROCESS UPPER AND LOWER CASE
03EB  47                MOV     B,A
03EC  3A 0705           LDA     CASE
03EF  FE03              CPI     3
03F1  CA 040D           JZ      CASX    ;ZERO SAYS SHIFT LOCKED UP
                ; LAND HERE, EITHER SINGLE SHIFT OR LOWER CASE
                ; TEST IF SINGLE SHIFT
03F4  FE01              CPI     1       ;SINGLE SHIFT COND?
03F6  3E00              MVI     A,0     ;CLEAR SHIFT ANYWAY
03F8  32 0705           STA     CASE
03FB  CA 040D           JZ      CASX    ;ZERO SAYS SINGLE SHIFT
                ; LAND HERE, LOWER CASE COND
                ; TEST IF IT IS ALPHA
03FE  78                MOV     A,B
03FF  FE41              CPI     41H     ;"A"
0401  DA 040D           JC      CASX    ;NOT ALPHA
0404  FE5B              CPI     5BH     ;HIGHEST ALPHA
0406  D2 040D           JNC     CASX    ;NEITHER ALPHA
                ; LAND HERE, CONVERT TO LOWER CASE
0409  3E20              MVI     A,20H
040B  B0                ORA     B
040C  47                MOV     B,A
                ; EXIT WITH CLEAR CARRY
040D  B7        CASX:   ORA     A
040E  78                MOV     A,B
040F  C9                RET
                ;
                ;
                ; PROCESS UNSHIFT OR LOWER CASE MODE
0410  AF        LCAS:   XRA     A
0411  C3 0423           JMP     NOLCK
                ; PROCESS SHIFT (EITHER SINGLE OR SHIFT LOCK)
0414  3A 0705   UCAS:   LDA     CASE
0417  FE01              CPI     1
0419  CA 0421           JZ      LOCK
041C  3E01              MVI     A,1     ;SINGLE SHIFT
041E  C3 0423           JMP     NOLCK
0421  3E03      LOCK:   MVI     A,3
0423  32 0705   NOLCK:  STA     CASE
0426  37                STC             ;SET CARRY, IGNORE LETTER
0427  C9                RET
                ;
                ;
                ;
                ; COLON WAS ENCOUNTERED IN TEXT,
                ; THIS TESTS NEXT TWO CHARS
                ; AGAINST ALL COMMANDS TO
                ; FIND COMMAND AND CALL TO IT
0428  46        CMND:   MOV     B,M
0429  23                INX     H
042A  4E                MOV     C,M
042B  23                INX     H
042C  22 070A           SHLD    CEPT    ;POINTS TO DELIMETER
042F  21 04D7           LXI     H,CTAB
                ; LOOP TO FIND MATCH
0432  7E        CLOP:   MOV     A,M
0433  23                INX     H
0434  B7                ORA     A
0435  CA 045C           JZ      CRTN
0438  B8                CMP     B
0439  CA 0442           JZ      ONEM
                ; HERE, FIRST LETTER FAIL
043C  23                INX     H
043D  23                INX     H
043E  23                INX     H
043F  C3 0432           JMP     CLOP
                ; HERE, FIRST LETTER MATCH
0442  7E        ONEM:   MOV     A,M
```

```
0443  23              INX     H
0444  B9              CMP     C
0445  CA 044D         JZ      TWOM
                ; HERE, SECOND LETTER FAIL
0448  23              INX     H
0449  23              INX     H
044A  C3 0432         JMP     CLOP
                ; COMMAND MATCH
044D  5E        TWOM: MOV     E,M     ;LOAD ADDR
044E  23              INX     H
044F  56              MOV     D,M
0450  2A 070A         LHLD    CEPT
0453  22 0735         SHLD    APNT    ;POINTING AT DELIM
0456  21 039B         LXI     H,PRIN  ;SET UP RETURN
0459  E5              PUSH    H
045A  EB              XCHG
045B  E9              PCHL            ;EXECUTE COMMAND
                ;
                ;
                ; FAILED TO MATCH A COMMAND
                ; SO PRINT THE TEXT
045C  2A 0735   CRTN: LHLD    APNT
045F  46              MOV     B,M
0460  23              INX     H
0461  C3 03B8         JMP     PROC
                ;;
                ; TAPE INPUT ROUTINE
0464  CD 0805   FDIN: CALL    INIR
0467  21 0975         LXI     H,TEXT
046A  CD D04F   TPLP: CALL    GBYT
046D  77              MOV     M,A
046E  23              INX     H
046F  D2 046A         JNC     TPLP    ;NO EOF FOR FDOS
                ; FOR TAPE, I WOULD CHECK FOR VALUE LESS THAN 3
                ; I.E. BINARY 1=EOF FOR TAPE
                ; AND CARRY SET = EOF FOR FDOS
0472  2B              DCX     H
0473  3601           MVI     M,1
0475  C3 0478         JMP     MAIN
                ;
                ;
                ;
                ;
                ;
                ; THIS IS INTERACTIVE PORTION
                ; OTHER THAN DIALOG ADDED FOR FDOS
                ; ALLOW 'L' OR 'P' OR 'D' ONLY
                ; LOAD OR PRINT OR QUIT
0478  CD 04BF   MAIN: CALL    ECHO1
047B  0E2E            MVI     C,'.'
047D  0605            MVI     B,5
047F  CD D03D   MAIL: CALL    CO      ;OUTPUT PROMPT
0482  05              DCR     B
0483  C2 047F         JNZ     MAIL
0486  CD D037         CALL    CI      ;GET COMMAND
0489  4F              MOV     C,A
048A  FE4C            CPI     'L'
048C  C2 0495         JNZ     MAIP
                ; LOAD FROM DISC (OR TAPE)
048F  CD 04BC         CALL    ECHO
0492  C3 0464         JMP     FDIN
0495  FE50      MAIP: CPI      'P'
0497  C2 04AC         JNZ     MAIQ
                ; PROCESS TEXT AND EMBEDDED COMMANDS
049A  CD 04BC         CALL    ECHO
049D  21 0975         LXI     H,TEXT
04A0  22 0735         SHLD    APNT
04A3  CD 02BD         CALL    CLOS
04A6  CD 011A         CALL    NEWL
04A9  C3 039B         JMP     PRIN
04AC  FE51      MAIQ: CPI     'Q'
04AE  C2 0478         JNZ     MAIN
                ; RETURN TO MONITOR
04B1  CD 04BC         CALL    ECHO
04B4  3E1F            MVI     A,1FH   ;DOWNSHIFT JUST IN CASE
04B6  CD 08B1         CALL    SELD
04B9  C3 D049         JMP     RESTPT
                ;
                ; THIS ROUTINE ECHO'S ON CO AND OUTPUTS CRLF
04BC  CD D03D   ECHO: CALL    CO
04BF  0E0D      ECHO1: MVI    C,CR
04C1  CD D03D         CALL    CO
04C4  0E0A            MVI     C,LF
04C6  CD D03D         CALL    CO
04C9  C9              RET
                ;THIS ROUTINE OUTPUTS A CR AND LF TO
                ;HIGH QUALITY PRINTER AND
                ; RETURNS WITH CR IN A REG
04CA  3E0D      CRLF: MVI     A,CR
04CC  CD 07E7         CALL    OUTW
04CF  3E0A            MVI     A,LF
04D1  CD 07E7         CALL    OUTW
04D4  3E0D            MVI     A,CR
04D6  C9              RET
                ; THE COMMAND TABLE IN SYS-8 FORMAT
                ; I.E. THE  TEXT LETTERS HAVE A REVERSED ORDER
04D7  4D44      CTAB: .WORD   'MD'
04D9  055F            .WORD   DMAR
04DB  5444            .WORD   'TD'
04DD  0577            .WORD   DTAB
04DF  4C50            .WORD   'LP'
04E1  05FE            .WORD   DPAG
04E3  544A            .WORD   'TJ'
04E5  059A            .WORD   TDTJ
04E7  454A            .WORD   'EJ'
04E9  05AA            .WORD   ENDJ
04EB  4C4A            .WORD   'LJ'
04ED  05A2            .WORD   LEFJ
04EF  5443            .WORD   'TC'
04F1  05B4            .WORD   CENT
04F3  464C            .WORD   'FL'
04F5  0607            .WORD   LNFD
04F7  4244            .WORD   'BD'
04F9  0632            .WORD   DBRK
04FB  5042            .WORD   'PB'
04FD  0641            .WORD   BRKP
04FF  504E            .WORD   'PN'
0501  065D            .WORD   NPAG
0503  4D43            .WORD   'MC'
0505  0676            .WORD   MIDC
0507  4E50            .WORD   'NP'
0509  0686            .WORD   SETP
050B  5053            .WORD   'PS'
050D  068F            .WORD   SPAC
050F  5450            .WORD   'TP'
0511  0698            .WORD   PGTP
0513  4250            .WORD   'BP'
0515  06A7            .WORD   PGBT
0517  4D54            .WORD   'MT'
0519  06B6            .WORD   TMFS
051B  4D42            .WORD   'MB'
051D  06D0            .WORD   BMES
051F  4750            .WORD   'GP'
0521  0559            .WORD   PPAG
0523  4343            .WORD   'CC'
0525  0553            .WORD   CEND
0527  464F            .WORD   'FO'
0529  06E2            .WORD   OPOF
052B  4E4F            .WORD   'NO'
052D  06E9            .WORD   OPON
052F  00              .BYTE   0
                ; SPR TO TERMINATE A COMMAND
                ; (CR, COMMA, SP, OR NOTHING ARE ACCEPTABLE)
```

```
0530  2A 0735    CTRM:   LHLD    APNT
0533  7E                MOV     A,M
0534  23                INX     H
0535  22 0735           SHLD    APNT
0538  FE0D              CPI     CR
053A  C8                RZ
053B  FE20              CPI     ' '
053D  C8                RZ
053E  FE2C              CPI     ','
0540  C8                RZ
0541  2B                DCX     H
0542  22 0735           SHLD    APNT
0545  C9                RET
                  ; SBR TO FIND DELIMITER WITHIN A COMMAND
                  ; (SP OR COMMA ARE ACCEPTABLE)
0546  2A 0735    CDEL:   LHLD    APNT
0549  7E                MOV     A,M
054A  23                INX     H
054B  FE2C              CPI     ','
054D  C8                RZ
054E  FE20              CPI     ' '
0550  C8                RZ
0551  2B                DCX     H
0552  C9                RET
                  ; CLOSE OR END CENTER TAB COMMAND
0553  CD 022F    CEND:   CALL    OUTL
0556  C3 0530           JMP     CTRM
                  ; IMMEDIATELY FORCE A PAGE START
0559  CD 02E9    FPAG:   CALL    DVDR
055C  C3 0530           JMP     CTRM
                  ; DEFINE MARGINS, LEFT, RIGHT
055F  CD 06F1    DMAR:   CALL    GARG
0562  32 070C           STA     LMAR
0565  32 0722           STA     BRTB    ;SET THAT TOO
0568  CD 06F1           CALL    GARG
056B  32 070D           STA     RMAR
056E  CD 02BD           CALL    CLOS
0571  CD 011A           CALL    NEWL
0574  C3 0530           JMP     CTRM
                  ; DEFINE TABS .TAB1,TAB2,TAB3,ETC UP TO 14
0577  21 070E    DTAB:   LXI     H,TTAB
057A  22 071D           SHLD    TBAD
057D  CD 0546    DTBL:   CALL    CDEL
0580  C2 0592           JNZ     DTBX
0583  CD 00F0           CALL    ADEC
0586  7D                MOV     A,L
0587  2A 071D           LHLD    TBAD
058A  77                MOV     M,A
058B  23                INX     H
058C  22 071D           SHLD    TBAD
058F  C3 057D           JMP     DTBL
0592  2A 071D    DTBX:   LHLD    TBAD
0595  3600              MVI     M,0
0597  C3 0530           JMP     CTRM
                  ;
                  ; SET TOTAL JUSTIFICATION MODE
059A  3E02        TOTJ:   MVI     A,2
059C  32 0730           STA     JFLG
059F  C3 0530           JMP     CTRM
                  ;
                  ; SET LEFT JUSTIFICATION MODE
05A2  3E01        LEFJ:   MVI     A,1
05A4  32 0730           STA     JFLG
05A7  C3 0530           JMP     CTRM
                  ;
                  ; CLOSE PRESENT LINE AND SET NO JUSTIFICATION MODE
05AA  AF          ENDJ:   XRA     A
05AB  32 0730           STA     JFLG
05AE  CD 02BD           CALL    CLOS
05B1  C3 0530           JMP     CTRM
                  ; CENTER TAB,TAB,MESSAGE TO BE CENTERED
05B4  CD 06F1    CENT:   CALL    GARG    ;GET TAB
05B7  32 0709    CENA:   STA     CETM
05BA  CD 0546           CALL    CDEL
05BD  C2 0530           JNZ     CTRM
05C0  22 070A           SHLD    CEPT
05C3  CD 02BD           CALL    CLOS
05C6  0E00        CENP:   MVI     C,0
05C8  2A 070A           LHLD    CEPT
                  ; COUNT CHARS IN MESSAGE
05CB  7E          CECC:   MOV     A,M
05CC  23                INX     H
05CD  CD 03E1           CALL    CAPR    ;DO CASE PROCESSING
05D0  DA 05CB           JC      CECC    ;LETTER IS TO BE IGNORED
05D3  0C                INR     C
05D4  FE0D              CPI     CR
05D6  C2 05CB           JNZ     CECC
                  ; COMPUTE POSN OF FIRST LETTER OF MESSAGE
05D9  79                MOV     A,C
05DA  B7                ORA     A
05DB  1F                RAR
05DC  4F                MOV     C,A
05DD  3A 0709           LDA     CETM
05E0  91                SUB     C
05E1  22 0735           SHLD    APNT    ;POINTS PAST MESSAGE
05E4  21 0757    MOVL:   LXI     H,OBUF  ;COMPUTE LADR FOR MESSAGE
05E7  CD 00AC           CALL    ADAH
05EA  EB                XCHG
05EB  2A 070A           LHLD    CEPT    ;START OF MESSAGE ADDRESS
05EE  7E          CEMV:   MOV     A,M     ;MOVE IT TO OBUF
05EF  23                INX     H
05F0  CD 03E1           CALL    CAPR    ;DO CASE PROCESSING AGAIN
05F3  DA 05EE           JC      CEMV    ;LETER TO BE IGNORED
05F6  FE0D              CPI     CR
05F8  C8                RZ
05F9  12                STAX    D
05FA  13                INX     D
05FB  C3 05EE           JMP     CEMV
                  ;DEFINE PAGE LENGTH
05FE  CD 06F1    DPAG:   CALL    GARG
0601  32 0720           STA     PLEN
0604  C3 0530           JMP     CTRM
                  ; LINE FEED COMMAND (IGNORE ZERO LF'S)
0607  CD 06F1    LNFD:   CALL    GARG
060A  32 0709           STA     CETM
060D  CD 0613           CALL    LFDO
0610  C3 0530           JMP     CTRM
                  ; SBR THAT ACTUALLY DOES LF'S AND KEEPS TRACK OF
                  ; POSITION IN PAGE.  IF NEW PAGE, REST OF
                  ; LF COMMAND IS FORGOTTEN
0613  CD 02BD    LFDO:   CALL    CLOS
0616  3A 0709           LDA     CETM
0619  B7                ORA     A
061A  C8                RZ
061B  4F                MOV     C,A
061C  CD 040A    LFLP:   CALL    CRLF
061F  3A 072F           LDA     PPOS
0622  3C                INR     A
0623  32 072F           STA     PPOS
0626  21 0720           LXI     H,PLEN
0629  BE                CMP     M
062A  D2 02CE           JNC     NEXP
062D  0D                DCR     C
062E  C8                RZ
062F  C3 061C           JMP     LFLP
                  ; DEFINE A PARAGRAPH BREAK, #LF'S, TAB
0632  CD 06F1    DBRK:   CALL    GARG
0635  32 0721           STA     BPLF
0638  CD 06F1           CALL    GARG
063B  32 0722           STA     BRTB
063E  C3 0530           JMP     CTRM
                  ; BREAK FOR A NEW PARAGRAPH
```

```
0641  CD 02BD    BRKP:   CALL    CLOS    ;CLEAR LINE
0644  3A 072F            LDA     PPOS    ;GET PAGE POIN
0647  B7                ORA     A
0648  CA 0654           JZ      BRKT    ;NO LF AT TOP OF PAGE
064B  3A 0721           LDA     BRLF
064E  32 0709           STA     CETM
0651  CD 0613           CALL    LFDO
0654  3A 0722    BRKT:   LDA     BRTB
0657  CD 00E3           CALL    PUTB
065A  C3 0530           JMP     CTRM
                ; FORCE BOTTOM PRESENT PAGE AND START NEW ONE
065D  3A 0724    NPAG:   LDA     PAGN
0660  B7                ORA     A
0661  CA 02E9           JZ      DVDR
0664  CD 02BD           CALL    CLOS
0667  CD 011A           CALL    NEWL
066A  3A 0720           LDA     PLEN
066D  32 0709           STA     CETM
0670  CD 0613           CALL    LFDO
0673  C3 0530           JMP     CTRM
                ; CENTER MIDDLE (BETWEEN MARGINS), MESSAGE
0676  3A 070D    MIDC:   LDA     RMAR
0679  21 070C           LXI     H,LMAR
067C  96                SUB     M
067D  1F                RAR             ;DIV BY TWO
067E  86                ADD     M
067F  CI 05B7           CALL    LENA
0682  12                STAX    D
0683  C3 022F           JMP     OUTL
                ; PAGE NUMBER
0686  CD 06F1    SETP:   CALL    GARG
0689  32 0724           STA     PAGN
068C  C3 0530           JMP     CTRM
                ; SET SPACING
068F  CD 06F1    SPAC:   CALL    GARG
0692  32 071F           STA     SPAC
0695  C3 0530           JMP     CTRM
                ; DEFINE TOP OF PAGE LENGTH AND IT'S LINE LF OCCURAN
                CE
0698  CD 06F1    PGTP:   CALL    GARG
069B  32 0723           STA     TLEN
069E  CD 06F1           CALL    GARG
06A1  32 0725           STA     TOPN
06A4  C3 0530           JMP     CTRM
                ; DEFINE BOTTOM OF PAGE LENGTH AND IT'S LINE OF OCCU
                RANCE
06A7  CD 06F1    PBOT:   CALL    GARG
06AA  32 0B1D           STA     BLEN
06AD  CD 06F1           CALL    GARG
06B0  32 0726           STA     BOTN
06B3  C3 0530           JMP     CTRM
                ; SET TOP MESSAGE ADDRESS AND THE MESSAGE TAB
06B6  CD 06F1    TMES:   CALL    GARG
06B9  32 072B           STA     TOPT
06BC  CD 0546           CALL    CDEL
06BF  C2 0530           JNZ     CTRM
06C2  22 0727           SHLD    TOPA
06C5  7E        TMLP:   MOV     A,M
06C6  23                INX     H
06C7  FE0D              CPI     CR
06C9  C2 06C5           JNZ     TMLP
06CC  22 0735           SHLD    APNT
06CF  C9                RET
                ; SET BOTTOM MSG ADDRESS AND MESSAGE TAB
06D0  CD 06F1    BMES:   CALL    GARG
06D3  32 072C           STA     BOTT
06D6  CD 0546           CALL    CDEL
06D9  C2 0530           JNZ     CTRM
06DC  22 0729           SHLD    BOTA
06DF  C3 06C5           JMP     TMLP
                ;
```

```
                ; SBR TO TURN OFF PRINTER OUTPUT
06E2  AF        OPOF:   XRA     A       ;ZERO TURNS OUTPUT OFF
06E3  32 0703           STA     OPST    ;MARK AT OUTPUT STATUS
06E6  C3 0530           JMP     CTRM    ;THAT'S ALL
                ;
                ; SBR TO TURN ON PRINTER OUTPUT
06E9  3EFF      OPON:   MVI     A,0FFH  ;NON-ZERO SAYS ON
06EB  32 0703           STA     OPST    ;MARK AT STATUS
06EE  C3 0530           JMP     CTRM    ;THAT'S ALL
                ;
                ;
                ; SBR THAT GETS THE NEXT ARGUMENT
06F1  CD 0546    GARG:   CALL    CDEL
06F4  C2 06FE           JNZ     GARE
06F7  CD 00F0           CALL    ADEC
06FA  7C                MOV     A,H
06FB  B7                ORA     A
06FC  7D                MOV     A,L
06FD  C8                RZ
06FE  E1        GARE:   POP     H
06FF  C3 0530           JMP     CTRM
                ; DEFINE VARIABLES AND ASSIGN SOME DEFAULT VALUES
000D            CR      =       13
000A            LF      =       10
0702  FF        EOT:    .BYTE   0FFH    ;LAST PAGE PRINTED IF ZERO
0703  FF        OPST:   .BYTE   0FFH    ;OUTPUT ON OR OFF STATUS
0704  00        TSTG:   .BYTE   0       ;TEMPORARY STORAGE FOR BMSG
0705  03        CASE:   .BYTE   3       ;UPPER CASE LOCK INITIALLY
0706  00        SFLP:   .BYTE   0       ;FLOP FOR EVRY AND SPACE
0707  5A        RNDV:   .BYTE   5AH     ;SEED FOR RAND
0708  00        LCHR:   .BYTE   0       ;LAST CHAR FOR TOTAL JUST
0709            CETM:   .BLKB   1       ;CENTR TAB OR CHAR COUNT
070A            CEPT:   .BLKB   2       ;CENTR TEXT POINTR
070C  0A        LMAR:   .BYTE   10      ;LEFT MARGIN
070D  46        RMAR:   .BYTE   70      ;RIGHT MARGIN
070E  0F        TTAB:   .BYTE   15      ;TAB TABLE
070F  16                .BYTE   22
0710  1E                .BYTE   30
0711  2D                .BYTE   45
0712  00                .BYTE   0
0713                    .BLKB   10      ;UP TO 15 TABS
071D            TBAD:   .BLKB   2       ;TAB TABLE POINTER
071F  01        SPAC:   .BYTE   1       ;SPACING
0720  2D        PLEN:   .BYTE   45      ;PAGE LENGTH
0721  01        BRLF:   .BYTE   1       ;NEW PAGE LF'S
0722  0F        PRTB:   .BYTE   15      ;" " TAB
0723  0A        TLEN:   .BYTE   10      ;TOP LENGTH
0724  00        PAGN:   .BYTE   0       ;PAGE NO.
0725  00        TOPN:   .BYTE   0       ;MSG LINE NO.
0726  00        BOTN:   .BYTE   0       ;MSG LINE NO.
0727            TOPA:   .BLKB   2       ;MSG ADDR
0729            BOTA:   .BLKB   2       ;MSG ADDR
072B  0A        TOPT:   .BYTE   10      ;TOP TAB
072C  0A        BOTT:   .BYTE   10      ;BOT TAB
072D  0A        BLEN:   .BYTE   10      ;BOTTOM LENGTH
072E  01        LPOS:   .BYTE   1       ;LINE POSITION
072F  01        PPOS:   .BYTE   1       ;PAGE PO ITION
0730  00        JFLG:   .BYTE   0       ;NO JUST INITIALLY
0731            LPDA:   .BLKB   2       ;LPOS ADDR
0733            LEND:   .BLKB   2       ;RIGHT MARGIN ADDRESS
0735            APNT:   .BLKB   2       ;INPUT POINTER
0737  47        MAXL:   .BYTE   135     ;MAX LINE LENGTH
0738  00        TCNT:   .BYTE   0       ;OVERFLOW CHAR COUNT
0739            TEML:   .BLKB   30      ;OVERFLOW BUFFER
0757            OBUF:   .BLKB   136     ;THE OUTPUT BUFFER
                ;
                ; I-O ROUTINES
                ; ALL REGISTERS ARE SAVED IN THESE ROUTINES
                ;
                ;
                ; THIS IS THE COMMAND CONSOLE
```

```
                        ; INPUT DEVICE
07DF  C3 D037           INPC:   JMP     CI      ;USE FDOS INPUT .
                        ;
                        ;
                        ; THIS IS THE PRINTER OUTPUT ROUTINE
                        ; PO IS THE SELECTRIC OUTPUT ROUTINE
                        ; CAN BE CHANGED TO CO FOR CONSOLE DEVICE
07E2  FE0D              OUTC:   CPI     CR
07E4  CA 04CA                   JZ      CRLF
07E7  C5                OUTW:   PUSH    B
07E8  E67F                      ANI     7FH
07EA  4F                        MOV     C,A
07EB  3A 0703                   LDA     OPST    ;TEST IF OUTPUT ON OR OFF
07EE  B7                        ORA     A
07EF  C4 0841                   CNZ     PO      ;NONZERO SAYS OUTPUT ON
07F2  CD D046                   CALL    CSTS
07F5  B7                        ORA     A
07F6  C4 07FC                   CNZ     ABTST   ;KEY WAS STRUCK
07F9  79                        MOV     A,C
07FA  C1                        POP     B
07FB  C9                        RET
                        ;
                        ; TEST IF ABORT WANTED
07FC  CD D037           ABTST:  CALL    CI
07FF  FE18                      CPI     24      ;CTRL-X?
0801  C0                        RNZ             ;NO ABORT
0802  C3 D049                   JMP     RESTRT  ;RETURN TO FDOS MONITOR
                        ;
D037                    CI      =       0D037H  ;CONSOLE INPUT ROUTINE
D03D                    CO      =       0D03DH  ;CONSOLE OUTPUT ROUTINE
D046                    CSTS    =       0D046H  ;CONSOLE STATUS ROUTINE
                        ;
                        ;
                        ;FDOS CONSTANTS
D04F                    FDREAD  =       0D04FH  ;SEQUENTIAL BYTE READ
D049                    RESTRT  =       0D049H  ;FDOS REENTRY
D064                    OPNIN   =       0D064H  ;OPEN SEQUENTIAL READ
D070                    LOKFIL  =       0D070H  ;CHECK IF FILE IN DIR
D06D                    FILNAM  =       0D06DH  ;CHECK FILENAME SYNTAX
D073                    TXTYP   =       0D073H  ;MSG OUTPUT ROUTINE
D076                    TI      =       0D076H  ;SIMULATES CI
D079                    TXTIN   =       0D079H  ;LINE INPUT ROUTINE
D02A                    REDFIL  =       0D02AH  ;FDOS BUFFER POINTER
                        ;
                        ;
                        ;
                        ; SBR INITIALIZES SEQUENTIAL INPUT
                        ; FROM MASS STORAGE DEVICE
                        ; RIGHT NOW, I'M USING FDOS
0805  21 0824           INIR:   LXI     H,GREET ;ASK FILE NAME
0808  CD D073                   CALL    TXTYP
080B  CD D079                   CALL    TXTIN   ;GET FILE NAME
080E  2A D02A                   LHLD    REDFIL
0811  CD D06D                   CALL    FILNAM
0814  DA 0805                   JC      INIR
0817  CD D070                   CALL    LOKFIL
081A  DA 0805                   JC      INIR
081D  CD D064                   CALL    OPNIN   ;OPEN FILE
0820  DA 0805                   JC      INIR
0823  C9                        RET
                        ;
                        ;
0824  5748415420      GREET:  .ASCIS  "WHAT I\
0829  4953205448      \S THE\
082E  4520494E50      \ INPU\
0833  5554204649      \T FIL\
0838  4C45204E41      \E NAM\
083D  4D453FA0        \E? "
                        ;
                        ;
                        ; SBR GBYT PERFORMS SEQUENTIAL READ
                        ; OF NEXT BYTE FROM INPUT FILE
D04F                    GBYT    =       FDREAD
                        ;
                        ;
                        ;
                        ; THIS IS THE SELECTRIC OUTPUT ROUTINE
                        ; THIS ROUTINE KEEPS TRACK OF SELECTRIC
                        ; CASE STATUS, AND TRANSLATES ASCII
                        ; CODE TO SELECTRIC CODE
                        ;
                        ; ONLY CODES FOR ASCII 20H AND UP ARE ALLOWED
                        ;
                        ; THE FIRST ENTRY IN TABL CORRESPONDS
                        ; TO ASCII SPACE OR CODE OF 20H
                        ;
                        ;
0841  79                PO:     MOV     A,C
0842  FE0A                      CPI     LF
0844  C8                        RZ
0845  FE0D                      CPI     CR
0847  CA 0892                   JZ      CRPR    ;PROCESS CR DIFFERENTLY
084A  C5                        PUSH    B
084B  D5                POT:    PUSH    D       ;COMPUTE ADDR OF SEL CODE
084C  E5                        PUSH    H
084D  59                        MOV     E,C
084E  1600                      MVI     D,0
0850  21 08F5                   LXI     H,TABL-20H
0853  19                        DAD     D       ;GOT SEL CODE POINTER
0854  79                        MOV     A,C     ;SAVE FOR TESTS
0855  4E                        MOV     C,M             ;GET SEL CODE
                        ;TEST IF UC,LC, OR CHANGE CASE
                        ; ALSO TEST FOR SPECIAL CHARS
0856  FE20                      CPI     20H     ;SPACE?
0858  CA 0884                   JZ      NSHT
085B  FE2C                      CPI     2CH     ;COMMA?
085D  CA 0884                   JZ      NSHT
0860  FE2E                      CPI     2EH     ;PERIOD?
0862  CA 0884                   JZ      NSHT
0865  3A 08B0                   LDA     SCASE
0868  A9                       XRA     C
0869  E680                     ANI     80H
086B  CA 0884                   JZ      NSHT            ;NO SHIFT REQUIRED
                        ;SHIFT REQUIRED
086E  79                        MOV     A,C
086F  E680                      ANI     80H
0871  3E1C                      MVI     A,1CH           ;SHIFT UP
0873  C2 0878                   JNZ     USHT            ;UPSHIFT REQUIRED
0876  3E1F              LSHT:   MVI     A,1FH           ;LOW SHIFT COMMAND
0878  CD 08B1           USHT:   CALL    SELO
087B  CD 0898                   CALL    DLAY
087E  79                        MOV     A,C
087F  E680                      ANI     80H
0881  32 08B0                   STA     SCASE           ;UPDATE NEW CASE
0884  79                NSHT:   MOV     A,C
0885  CD 08B1                   CALL    SELO
0888  79                        MOV     A,C
0889  FE6D                      CPI     6DH             ;NEED CR DELAY?
088B  CC 08A6                   CZ      CRDLY
088E  E1                SELX:   POP     H
088F  D1                        POP     D
0890  C1                        POP     B
0891  C9                        RET
                        ;
0892  C5                CRPR:   PUSH    B
0893  0E7B                     MVI     C,7BH   ;TABLE ADDRESS FOR CR CODE
0895  C3 084B                   JMP     POT
                        ;
                        ; DELAY FOR SHIFT AND CRLF
0898  C5                DLAY:   PUSH    B
0899  01 0090                 LXI     B,90H
089C  05                X1:     DCR     B
```

```
089D  C2 089C           JNZ     $1
08A0  0D                DCR     C
08A1  C2 089C           JNZ     $1
08A4  C1                POP     B
08A5  C9                RET
                ;
08A6  CD 0898   CRDLY:  CALL    DLAY
08A9  C5                PUSH    B
08AA  01 0000           LXI     B,0
08AD  C3 089C           JMP     $1
                ;
08B0  00        SCASE:  .BYTE   0       ;MSB IS CASE STORAGE
                ;
                ;SELECTRIC OUTPUT ROUTINE
                ; THIS IS A SOFTWARE UART FOR MY
                ; DATEL 1030 TERMINAL
                ; SEE KILOBAUD #17 PAGE 44
                ; ARTICLE BY DAN STOGDILL
08B1  C5        SELO:   PUSH    B
08B2  57                MOV     D,A
08B3  1E07              MVI     E,7
08B5  AF                XRA     A
08B6  D300              OUT     0
08B8  01 015A           LXI     B,VALU
08BB  CD 08DF           CALL    TIMR
08BE  7A        OSEL:   MOV     A,D
08BF  D300              OUT     0
08C1  0F                RRC
08C2  57                MOV     D,A
08C3  01 015A           LXI     B,VALU
08C6  CD 08DF           CALL    TIMR
08C9  1D                DCR     E
08CA  C2 08BE           JNZ     OSEL
08CD  3E01              MVI     A,1
08CF  D300              OUT     0
08D1  01 015A           LXI     B,VALU
08D4  CD 08DF           CALL    TIMR
08D7  01 015A           LXI     B,VALU
08DA  CD 08DF           CALL    TIMR
08DD  C1                POP     B
08DE  C9                RET
                ;
08DF  0B        TIMR:   DCX     B
08E0  DB00              IN      0
08E2  E601              ANI     1
08E4  79                MOV     A,C
08E5  B0                ORA     B
08E6  C2 08DF           JNZ     TIMR
08E9  C9                RET
                ;
015A            VALU    =       15AH
                ;
                ;
                ;SELECTRIC INPUT ROUTINE
                ;  SOFTWARE INPUT UART
08EA  C5        SELI:   PUSH    B
08EB  D5                PUSH    D
08EC  E5                PUSH    H
08ED  11 0008           LXI     D,8
08F0  DB00      TSTR:   IN      0
08F2  E601              ANI     1
08F4  C2 08F0           JNZ     TSTR
08F7  01 015A           LXI     B,VALU
08FA  CD 08DF           CALL    TIMR
08FD  DB00      ISEL:   IN      0
08FF  E601              ANI     1
0901  B2                ORA     D
0902  0F                RRC
0903  57                MOV     D,A
0904  01 015A           LXI     B,VALU
0907  CD 08DF           CALL    TIMR
090A  1D                DCR     E
090B  C2 08FD           JNZ     ISEL
090E  7A                MOV     A,D
090F  E67F              ANI     7FH
0911  E1                POP     H
0912  D1                POP     D
0913  C1                POP     B
0914  C9                RET
                ;
                ;
                ; THIS IS A TABLE OF SELECTRIC CODES WHOSE
                ; POSITION ADDRESS CORRESPONDS TO THE
                ; ASCII CODE FOR THE LETTER
                ; THE MSB IS THE CASE OF THE SELECTRIC CODE
                ; I.E. MSB=0=>L.C.  MSB=>U.C.
                ;
0915  4001C9F084 TABL:  DB      40H,01,0C9H,0F0H,84H,88H
091A  88E849            ,0E8H,49H
                ; ABOVE ARE SP,!,",#,$,%,&,'
091D  B4E4B8933E        DB      0B4H,0E4H,0B8H,93H,3EH,37H
0922  375107            ,51H,7
                ; ABOVE ARE (,),*,+,,,-,.,/
0925  6420107004        DB      64H,20H,10H,70H,4,8
092A  085868            ,58H,68H
                ; ABOVE ARE 0,1,2,3,4,5,6,7
092D  3834EB6BB4        DB      38H,34H,0EBH,6BH,0B4H,13H
0932  13C587            ,0C5H,87H
                ; ABOVE ARE 8,9,:,;,<,=,>,?
0935  90F9F6FAAA        DB      90H,0F9H,0F6H,0FAH,0AAH,0C6H
093A  CAF3A3            ,0F3H,0A3H
                ; ABOVE ARE @,A,B,C,D,E,F,G (UC)
093D  A699C39AC6        DB      0A6H,99H,0C3H,9AH,0C6H,0E1H
0942  E1D2C5            ,0D2H,0C5H
                ; ABOVE ARE H,I,J,K,L,M,N,O (UC)
0945  8BDBA9A582        DB      8BH,0DBH,0A9H,0A5H,82H,0B2H
094A  B2B1F5            ,0B1H,0F5H
                ; AV ARE P,Q,R,S,T,U,V,W
094D  E2E7D4B440        DB      0E2H,0E7H,0D4H,0B4H,40H,0C5H
0952  C5A0B7            ,0A0H,0B7H
                ; ABOVE ARE X,Y,Z,[,\,],^,_
0955  4979767A2A        DB      49H,79H,76H,7AH,2AH,4AH
095A  4A7323            ,73H,23H
                ; ABOVE ARE `,A,B,C,D,E,F,G (LOWER CASE)
095D  2619431A46        DB      26H,19H,43H,1AH,46H,61H
0962  615245            ,52H,45H
                ; ABOVE ARE H,I,J,K,L,M,N,O (LC)
0965  0B5B292502        DB      0BH,5BH,29H,25H,2,32H
096A  323175            ,31H,75H
                ; ABOVE ARE P,Q,R,S,T,U,V,W (LC)
096D  626754            DB      62H,67H,54H     ;END OF ALL PRINTS
                BEI
                ; ABOVE ARE X,Y,Z  (LC)
                ; NOW FOR ADDITIONALS (TABLE IS 7FH
0970  6D                DB      6DH     ;CR
0971  7C                DB      7CH     ;ATTN
0972  000000            DB      0,0,0   ;TERMINATOR
                ;
                ;
                ;
                ;
0975            TEXT:   .END            ;THIS IS START OF TEXT
```

DATA-BOY™
SPEECH PROCESSOR

BY JAMES ANDERSON
Copyright 1978
Mimic Electronics
P. O. Box 921
Acton, MA 01720

The Data-Boy™ speech processor is essentially a speech "digitizer" which uses a proprietary signal processing technique to convert the human voice into a single bit data stream, and vice versa. "Digitized" speech is typically thought of as speech which has been sampled with an analog-to-digital (A/D) converter, stored in digital memory or transmitted in digital form, and then re-converted to analog form by a digital-to-analog (D/A) converter. By using certain characteristics of the speech waveform, especially the fact that the amplitude components tend to decrease with increasing frequency, the resolution of the A/D and D/A converters required can be decreased from, say, 8 bits down to a single bit while maintaining intelligibility. When this bit stream is sampled at a rate of 8000 samples per second, highly intelligible speech can be obtained. The speech quality is close to that which is given by a CB radio. At this low data rate (8000 bits per second), the sixteen hexadecimal digits, "zero" through "F", can be spoken from the data stored in only 8000 bytes of memory.

Speech digitization has several advantages over other methods of speech production. First of all, the hardware required is fairly simple and can easily be mass-produced, thus lowering the initial cost to the user. Secondly, the software cost is small. Simple programs for sampling, storing, and playing back the speech data can be written with little effort. Also, the cost of adding a new word to an existing vocabulary is minimal. Finally, the system gives "natural-sounding" speech (especially when whole phrases are recorded and played back) because the speech was originally produced by a human.

A one-bit speech representation has several advantages over other speech digitization methods, aside from the obvious reduction in the number of I/O lines. The speech can be stored in very slow memory (100 microsecond access time or less) and a few random bit failures in the memory will not significantly affect speech quality, as the speech data itself is highly redundant. Thus defective RAMs and ROMs can be used, and these are available at very low prices . . . so *don't base the cost of a speech storage memory on what you're used to paying for computer-grade chips*! Note that this use of defective memory is not recommended for use with, say, 8-bit A/D conversion speech digitizers as bit failures in the MSBs tend to give rather annoying "pops" in the speech playback.

Another interesting aspect of the one-bit representation of speech is that it is very easy to do digital signal processing with such data. The Discrete Fourier Transform (DFT) can run in less time than it takes the Fast Fourier Transform to run—and with a much simpler program. The reason for this is that when the time domain data is always a "zero" or a "one," all of the complex multiplications in the DFT turn into additions. Thus it may be possible to do automatic word recognition on a limited vocabulary in something approaching real-time operation.

The Data-Boy™ speech processor system provides a low-cost, easy to use, reliable method of speech production. In fact, this method of speech production is the lowest cost of any method available where limited vocabularies are concerned. New uses for computer speech output, such as video games, aids for the handicapped, and perhaps even talking appliances, may result.

Interfacing the Data-Boy™

The speech processor can be interfaced to just about any serial or parallel I/O port on a computer, as long as the port uses TTL-compatible signals. Any bit which can be changed rapidly under computer control can be used to produce speech! As a last resort, it is possible to interface the Data-Boy™ to a computer through the computer's A/D and D/A converters, as long as the TTL conventions for signal levels are observed (this is applicable to large computer installations where user access to I/O ports is not available).

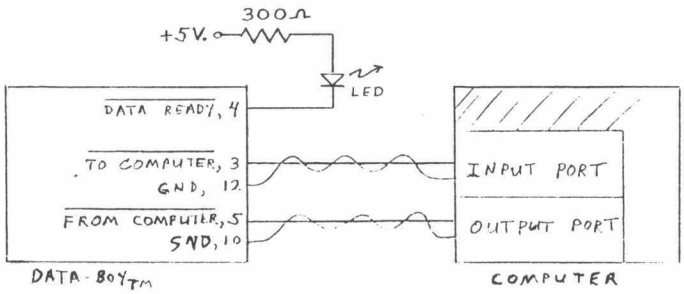


Figure 1

Simple Configuration for Recording and Playback

Figure 1 shows a simple configuration for the recording and playback of speech. The power supply, microphone, and speaker should be connected, but are not shown. An optional Light Emitting Diode (LED) is shown, which gives a visual indication to the user when the SQUELCH threshold has been exceeded and data is present on the TO COMPUTER line.

The recording process is initiated when the user notifies the computer that he or she is ready to record (via a RUN switch or keyboard input, for example). The user then speaks into the microphone while the computer samples the data on the TO COMPUTER line. The computer stores these bits (the speech data samples) in memory until a fixed length buffer is filled. At some future time, the computer puts the data bits onto the FROM COMPUTER line at the same rate at which they were originally sampled, and intelligible speech results at the speaker. Phonemes, words, or entire phrases can be stored and reproduced in this manner. *NOTE*: If phrases are stored (continuous speech), a certain amount of static (noise) may be heard between the individual words of the phrase. This can be eliminated by editing the data so that only the valid speech remains (set all the other bits to zero), or by simply recording the words one at a time and then outputting them in succession to form the phrase.

Sophisticated Configuration for Recording and Playback

Figure 2 shows a more sophisticated configuration for the recording and playback of speech. The power supply, microphone, and speaker should be connected, but are not shown.

In this configuration, a user speaks into the microphone, thereby automatically initiating the recording process. As long as the DATA READY signal is active (low), the computer samples the state of the TO COMPUTER line and stores these speech data bits in memory. The DATA READY signal line can be connected as an "interrupt" signal to the computer, if desired, or it can simply be sampled periodically to see if it is active. Speech output is handled in the same manner as before (see section E). A flowchart for a computer program which utilizes the configuration of Figure 2, and which makes efficient use of computer memory space, is presented in Fig. 3.

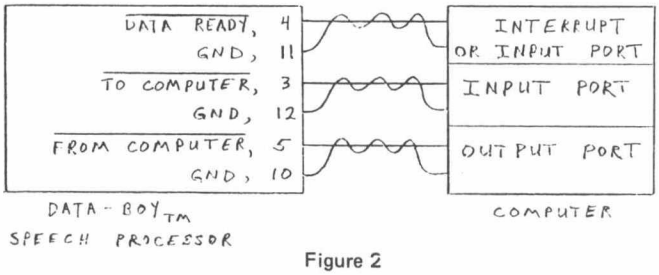
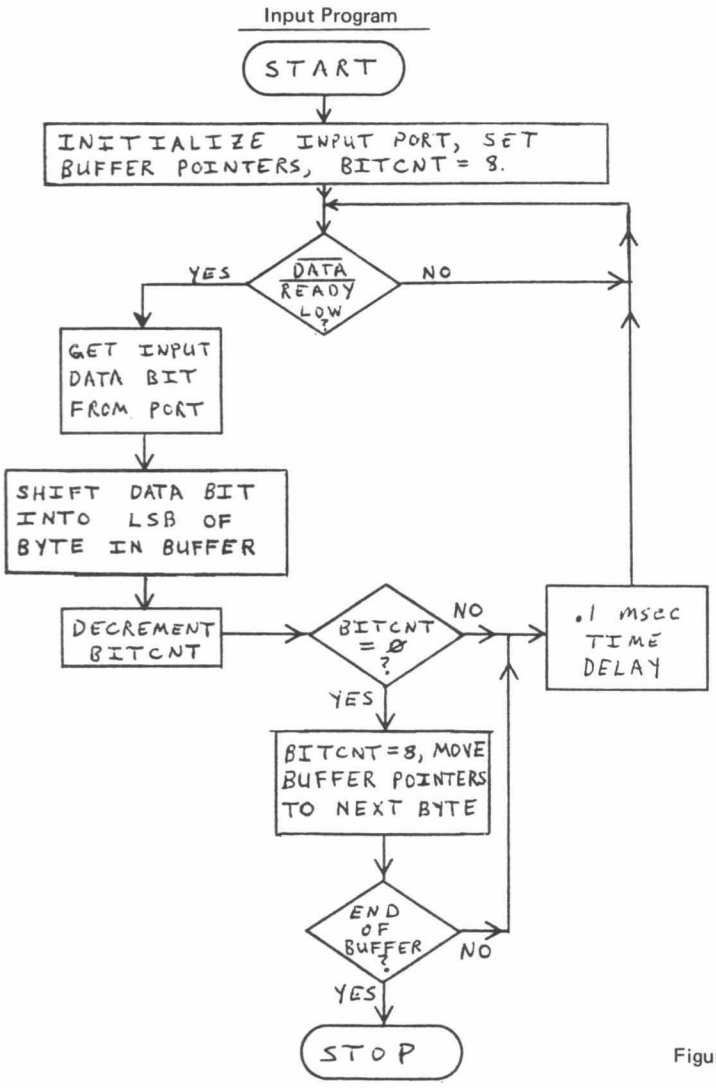


Figure 2



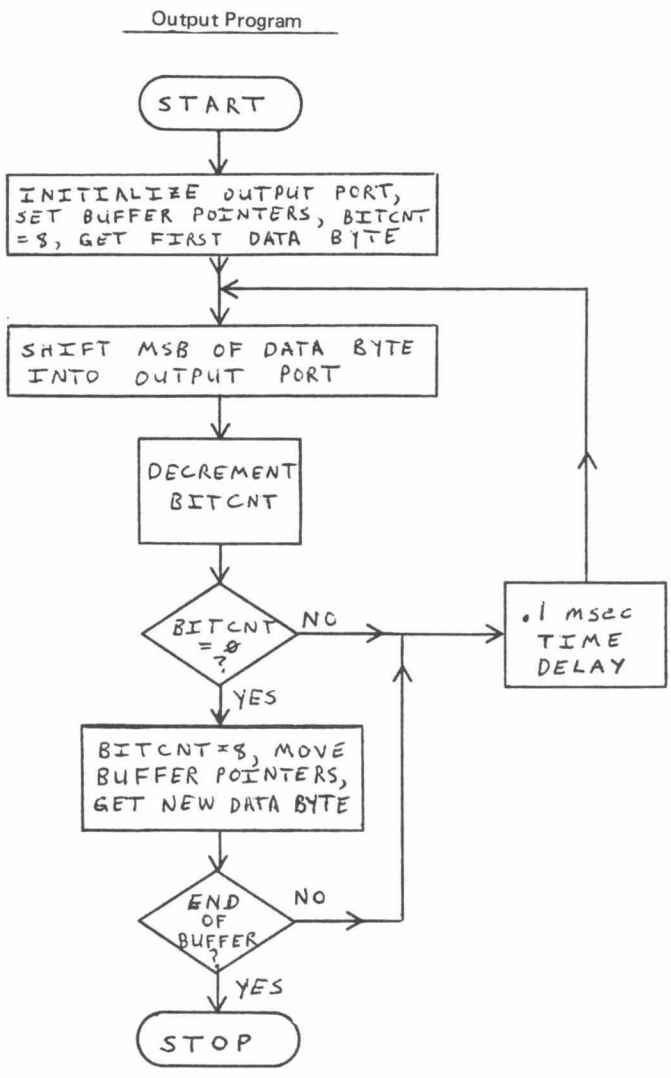
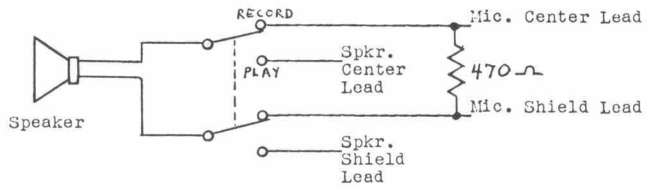


Figure 3

If only half-duplex operation is required, the speaker can "double" as a microphone. Speech quality may degrade somewhat, depending on choice of speaker. The microphone input leads should be terminated (preferably shorted) when not in use, to prevent spurious signals from triggering the DATA READY line. One possible configuration is shown:



If higher power output to the speaker or external volume control is desired, an additional power amplifier can be used.

Speech Storage Space vs. Intelligibility.

A common method for evaluating speech intelligibility is the "articulation test." Typically, a person reads a list of syllables or unrelated words to an "untrained" group of listeners (recognition ability improves with practice) and the percentage of items identified correctly is taken as the articulation score. By choosing test material representative of the sound statistics of a language, a realistic test of the system can be made. Word articulation scores for speech which has been processed by the Data-Boy™ input circuit, sampled at a 10 KHz. rate, and played back through the output circuit are in the neighborhood of 90% for trained listeners. When words are used in sentences, contextual information is present which leads to considerably higher articulation scores. To test your system, try recording the sentences "Joe took father's shoe bench out," and "She was waiting at my lawn." Together, these sentences contain most of the sounds in the English language.

For a fixed memory size, the amount of speech which can be stored is inversely proportional to the sampling rate. For example, if one spoken word can be stored in 5000 bits of memory with a 10 KHz. sampling rate, then two spoken words (each of the same duration as the original spoken word) can be stored in the same memory with a 5 KHz sampling rate. Thus it pays memory-wise to have the lowest sampling rate possible. However, there is a trade-off in intelligibility. To minimize this trade-off, the following two methods of speech recording are recommended.

1. Words containing mostly low frequencies (eg., "one") can be recorded with a much lower sampling rate than words containing mostly high frequencies (eg., "fix"). Simply store the sampling rate which was used to record the word, and use that same rate to output the data for playback.
2. The effective amount of storage required can be reduced somewhat by using phoneme concatenation. For example, the spoken word "seven" can be stored as an "s" sound followed by an "eh-vun" sound. The same "s" sound can also be used in other words such as "six," "s" plus "ick" plus "s," without having to re-record it. Similarly, one recording of the word "teen" will allow you to generate "seventeen" with a simple program which outputs "s" plus "eh-vun" plus "teen." The method is quite limited with this hardware, however, and in general will not produce acceptable speech. For example, when the spoken word "dog" is broken up into "duh" plus "aw" plus "guh," the resulting audio does not sound like the intended word. This is due to the fact that in natural sounding speech, the end of one phoneme often blends into the start of the next (but not always, as was shown in "seventeen"). If all of the phonemes are recorded separately, some method is needed to blend them together—a formidable task indeed for this method of speech production.

Other methods, such as encoding the bit stream, are possible but may prove difficult to program or result in little memory savings by comparison. Note that the two methods cited above apply to the recording of discrete spoken words.

A reasonable vocabulary can be stored on a standard floppy disk: 240,000 bytes can contain 240 seconds of continuous speech (or about 480 discrete spoken words) if an 8 KHz sampling rate is used. Alternatively, a cheap semiconductor RAM or ROM system can be built to any size, using slow and defective chips (with random bit errors). A few random bit failures will not significantly affect speech quality, as the speech data itself is highly redundant. However, bit failures of a periodic nature (eg., every eighth bit set to a logical "one") will tend to introduce undesirable "notes" or "tones" into the reproduced speech. In any case, it is more cost-effective to use an inexpensive "speech storage" peripheral mass memory rather than expensive computer main memory for vocabularies of more than 20 words or so.

Intelligibility of speech produced by the Data-Boy™ speech processor can generally be described according to the sampling rate used. A 5 KHz. rate gives marginally intelligible recorded phrases. An 8 to 10 KHz. rate gives highly intelligible single words or sounds and extremely intelligible phrases. Sampling rates greater than 15 KHz. merely waste memory space while offering no appreciable improvement in speech quality.

```
                    * STORE VOICE PROGRAM
                    * HARDWARE BY JIM ANDERSON
                    * SOFTWARE BY BOB GRAPPEL
                    * USES PIA-A SIDE AT 8014 HEX
                    *      OUTPUT ON BIT 7
                    * USES PIA-B SIDE AT 8016 HEX
                    *      INPUT ON BIT 0
                    *
0100                        ORG     $100
                    * INITIALIZE PIA
0100 CE 80 14               LDX     #PIA        ADDRESS OF PORT
0103 6F 00                  CLR     0,X
0105 63 00                  COM     0,X
0107 6F 01                  CLR     1,X
0109 6F 02                  CLR     2,X
010B 6F 03                  CLR     3,X
010D 86 04                  LDA A   #4
010F A7 01                  STA A   1,X         SET CONTROLS
0111 A7 03                  STA A   3,X
0113 CE 01 89               LDX     #BUFFER     CLEAR BUFFER AREA
0116 6F 00          CLEAR   CLR     0,X
0118 08                     INX
0119 BC 01 87               CPX     BUFEND      DONE?
011C 26 F8                  BNE     CLEAR
                    *
011E 7F 80 14               CLR     PIA         TURN OFF OUTPUT
0121 86 08                  LDA A   #8
0123 B7 01 84               STA A   BITCNT
0126 86 42                  LDA A   #'B         "BEGIN" PROMPT
0128 BD E1 D1               JSR     OUTEEE
012B CE 01 89               LDX     #BUFFER
                    *
                    * STORE VOICE DATA BITS HERE
                    *
012E 74 80 16       LOOPI   LSR     PIAB        GET DATA BIT
0131 69 00                  ROL     0,X         PUT BIT IN BYTE
0133 7A 01 84               DEC     BITCNT      DONE WITH BYTE?
0136 26 0B                  BNE     DELAY       YES
                    *
0138 08                     INX                 MOVE POINTERS
0139 BC 01 87               CPX     BUFEND      DONE?
013C 27 0D                  BEQ     DONE        YES
                    *
013E 86 08                  LDA A   #8          RESET BITCNT
0140 B7 01 84               STA A   BITCNT
                    *
0143 B6 01 86       DELAY   LDA A   TIMER       DELAY LOOP
0146 4A                     DEC A
0147 26 FD                  BNE     *-1         MAKE SAMPLE RATE 8 KHZ
                    *
0149 20 E3                  BRA     LOOPI       CONTINUE
                    *
014B 86 45          DONE    LDA A   #'E         "END" PROMPT
014D BD E1 D1               JSR     OUTEEE
0150 7F 80 14               CLR     PIA         TURN OFF OUTPUT
0153 7E E0 E3       QUIT    JMP     MONTOR      RETURN TO SYSTEM
                    *
                    * VOICE OUTPUT SECTION HERE
                    *
0156 CE 01 89       OUTS    LDX     #BUFFER
0159 A6 00                  LDA A   0,X         GET DATA BYTE
015B B7 01 85               STA A   TEMP
                    *
015E 78 01 85       LOOPO   ASL     TEMP        GET DATA BIT FROM BYTE
0161 76 80 14               ROR     PIA         PUT INTO PIA
0164 7A 01 84               DEC     BITCNT      DONE WITH BYTE?
0167 26 10                  BNE     DELAY2      YES
                    *
0169 08                     INX                 MOVE POINTERS
016A BC 01 87               CPX     BUFEND      DONE?
016D 27 12                  BEQ     STOP        YES
                    *
016F 86 08                  LDA A   #8
0171 B7 01 84               STA A   BITCNT      RESET BITCNT
0174 A6 00                  LDA A   0,X         GET NEW BYTE
0176 B7 01 85               STA A   TEMP
                    *
0179 B6 01 96       DELAY2  LDA A   TIMER       DELAY LOOP
017C 4A                     DEC A               MAKE SAMPLE RATE 8 KHZ
017D 26 FD                  BNE     *-1
                    *
017F 20 DD                  BRA     LOOPO
                    *
0181 7E E0 E3       STOP    JMP     MONTOR      RETURN TO SYSTEM
                    *
8014                PIA     EQU     $8014
8016                PIAB    EQU     $8016
E1D1                OUTEEE  EQU     $E1D1
E0E3                MONTOR  EQU     $E0E3
                    *
0184                BITCNT  RMB     1
0185                TEMP    RMB     1
0186 08             TIMER   FCB     8
0187 6E 76          BUFEND  FDB     $6FFF-BUFFER
0189                BUFFER  EQU     *
                            END
      NO ERROR(S) DETECTED

   SYMBOL TABLE:
BITCNT  0184    BUFEND  0187    BUFFER  0189    CLEAR   0116
DELAY   0143    DELAY2  0179    DONE    014B    LOOPI   012E
LOOPO   015E    MONTOR  E0E3    OUTEEE  E1D1    OUTS    0156
PIA     8014    PIAB    8016    QUIT    0153    STOP    0181
TEMP    0185    TIMER   0186
```

Whatis:
An Expression Evaluator for the 8080

BY MIKE GABRIELSON
Box 2692
Stanford, Calif. 94305

Programmers often need to manually verify the results obtained by a computer, especially while sitting at a console and debugging code. Pocket calculators with capabilities like hex or octal readouts and Boolean functions are a convenient tool, but ignore the fact that the computer itself can operate as a handy desk calculator.

Whatis

Whatis is a program for the 8080 that can evaluate arbitrary arithmetic and logical expressions, such as:

5 * (3 – (6 * 2)) <= (#FFE / 2)

Whatis operates as a subroutine. It accepts the address of an ASCII string that contains the expression to be evaluated. The end of the expression string is marked by a character called EOS. In the accompanying listing, EOS is equated to zero. Whatis will evaluate any string and return the expression's value in the HL register pair. If the result of an expression is a string, the value returned in HL is the address of the string. However, if a fatal error (such as division by zero) occurs, Whatis will load the address of an error message string into HL and then call an external routine named $ABORT. Whatis does not expect $ABORT to return control.

Whatis Input

Whatis breaks the expression string into a sequence of logical syntactic elements called tokens, which are special sequences of adjacent characters. A token is either an operator or a constant.

Operators

Operators are used in conjunction with constants to form expressions. The operators are:

&	dyadic and
*	dyadic multiply
+	dyadic add or monadic plus
–	dyadic subtract or monadic negate (two's complement)
/	dyadic divide
//	dyadic remainder
<	dyadic less than
>	dyadic greater than
<=	dyadic less than or equal
==	dyadic equal
>=	dyadic greater than or equal
<<	dyadic or monadic rotate left
>>	dyadic or monadic rotate right
|	dyadic inclusive or
||	dyadic exclusive or
~	monadic one's complement
~=	dyadic not equal

Constants

Constants occur in one of two forms: numbers or strings. Numbers represent values for use in expressions, and can be represented in decimal or hexadecimal. A decimal number is a sequence of decimal digits. A hexadecimal number is a sequence of hexadecimal digits prefixed by a pound sign (#) or enclosed in single quotes and prefixed by an X. All numbers are evaluated with 16 bits of precision, and overflow is ignored, so that numbers are limited to the range 0 to 65535 decimal (0 to #FFFF hexadecimal).

Examples: 1234

X' face'	(64206 decimal)
#32B	(811 decimal)
99999	(evaluates to 34463 decimal because of overflow)

Strings are any sequence of characters enclosed in single quotes. Each character represents the corresponding seven bit ASCII code, with a high order eighth bit of zero. Two consecutive single quotes are required to represent one single quote within a string. Null strings are ignored.

Examples: 'Pack My Box With Five Dozen Liquor Jugs'
'USERS"S MANUAL'

Expressions

An expression has the form:

{{operadl}operator} operand2

where "operator" is one of the monadic or dyadic operators, and "operandl" and "operand2" are constants or expressions. The brackets { } denote optional items. Dyadic operators require both operands, monadic operators require only "operand2."

All operators are of equal precedence, and expressions are evaluated from left to right. Parentheses around an expression may be used to alter precedence; the innermost expression in parentheses is evaluated first. Expressions are evaluated to 16 bits of precision with no check for overflow. All arithmetic operations are performed using unsigned two's complement arithmetic.

Examples:	3	Evaluates to:	3
	((3))		3
	-(+3)		#FFFD
	~3		#FFFC
	1000 * 1000		16960
	#FF & 300		#2C
	16//6		4
	1 + 2*3		9
	1 + (2*3)		7
	(#2B - (7 | 12)) + (5 | | 13)		36

Relational Operators

The dyadic operators

== ~= <= < >= >

are used to form expressions that evaluate either to zero or to one. All relational operators perform an unsigned magnitude comparison. If the relation is true, the operators produce a result of one. A false comparison results in zero.

Examples:	5 == 5	Evaluates to:	1
	3 + (2 ~= 6)		4
	(1 <= 2) <= 3		1
	#FFFF < 0		0

The operators

<< >>

can be used as dyadic or monadic operators to rotate (not shift) 16 bit operands. Rotation implies wraparound of bits. Shifting can be accomplished with the multiply and divide operators. When used as monadic operators, operands are rotated one bit. When used as dyadic operators, the first operand specifies the number of bits to rotate.

Examples:	<< 2	Evaluates to:	4
	>> 3		#8001
	> > (< < 2)		2
	3 > > #f0		#1E

String Operations

If a string constant appears in an expression with operators, the value of the string is the ASCII code of the string's first character. An exception to this rule occurs when two strings appear with a relational operator: then a character by character comparison of the two strings is performed, using the ASCII code to determine relative order. If the two strings are of unequal length, the shorter is padded on the right with blanks.

Examples:	'A' + 1	Evaluates to:	#42
	'ABC' & #ff		#41
	'CAT' < 'dog'		1
	'PAD' == 'PAD '		1
	'123' == #31		1

Final Notes

The Whatis program listing was generated by a one-pass assembler. As a result, all forward references show up as question marks in the object code field.

Provided with the necessary input, output, and error handling, Whatis can be a useful addition to a programmer's toolbox.

```
                                Page    1 of *whatis

                        ;*****************************************************
                        ;* Whatis ... an expression evaluator for the 8080   *
                        ;*                                                   *
                        ;* Mike Gabrielson  7/22/78                          *
                        ;*                                                   *
                        ;* Accepts:  HL = address of expression string       *
                        ;*               terminated by EOS                   *
                        ;* Returns:     A = result type (NUMRAN or STRRAN)   *
                        ;*              HL = value                           *
                        ;*                                                   *
                        ;* Fatal errors cause $ABORT to be called and        *
                        ;* Whatis does not expect to regain control.  If     *
                        ;* the result is a string (A = STRRAN), then HL is   *
                        ;* a pointer to the string.                          *
                        ;*****************************************************

                                !EXTRN  $ABORT          ;FATAL ERROR HANDLER
                                !ENTRY  WHATIS          ;THE ONLY GLOBAL SYMBOL

                                !LINES  #FFFF           ;SUPPRESS PAGE EJECTS

0000 CD ?? ??   WHATIS: CALL    INIT            ;INITIALIZE VARIABLES
0003 CD ?? ??           CALL    EVAL            ;EVALUATE THE EXPRESSION
0006 C9                 RET

                        EOS = 0                 ;END OF STRING CHARACTER

0007 45 78 70   MSG1:   'Expression missing',EOS
001A 4D 69 73   MSG2:   'Missing left parenthesis',EOS
0033 49 6C 6C   MSG3:   'Illegal character',EOS
0045 4F 70 65   MSG4:   'Operator stack underflow',EOS
005E 4D 69 73   MSG5:   'Missing end of string quote',EOS
007A 53 69 6E   MSG6:   'Single quote must follow X and x',EOS
009B 49 6C 6C   MSG7:   'Illegal hex digit',EOS
00AD 4D 69 73   MSG8:   'Missing end of hex constant quote',EOS
00CF 49 6C 6C   MSG9:   'Illegal left parenthesis',EOS
00E8 4F 70 65   MSG10:  'Operand stack overflow',EOS
00FF 4F 70 65   MSG11:  'Operator stack overflow',EOS
0117 49 6E 76   MSG12:  'Invalid equals operator',EOS
012E 44 69 76   MSG13:  'Division by zero',EOS
0140 4F 70 65   MSG16:  'Operand stack underflow',EOS
0158 4F 70 65   MSG17:  'Operator missing',EOS

                        OPRSIZ = 100    * = * + OPRSIZ          ;OPERAND, OPERATOR STACK SIZE
                        OPRAND: * = * + OPRSIZ                  ;OPERAND STACK
                        OPRATR: * = * + OPRSIZ                  ;OPERATOR STACK
                        RANPTR: * = * + 2                       ;OPERAND STACK POINTER
                        RTRPTR: * = * + 2                       ;OPERATOR STACK POINTER

                        BEFORE: * = * + 1               ;PREVIOUS TOKEN CODE
                        GETPTR: * = * + 2               ;ADDRESS OF NEXT EXPRESSION CHARACTER

                        NUMRAN =        0               ;NUMERIC OPERAND TYPE CODE
                        STRRAN =        1               ;STRING OPERAND TYPE CODE

                        ;-----
                        ; TOKEN CODES...
                        ;-----
                        PLUS    =       0               ;DYADIC +
                        MINUS   =       PLUS+1          ;DYADIC -
                        TIMES   =       MINUS+1         ;*
                        DIVIDE  =       TIMES+1         ;/
                        REMAIN  =       DIVIDE+1        ;//
                        LEFT    =       REMAIN+1        ;DYADIC <<
                        RIGHT   =       LEFT+1          ;DYADIC >>
                        LAND    =       RIGHT+1         ;&
                        IOR     =       LAND+1          ;|
                        XOR     =       IOR+1           ;||
                        EQ      =       XOR+1           ;==
                        LT      =       EQ+1            ;<
                        LE      =       LT+1            ;<=
                        GT      =       LE+1            ;>
                        GE      =       GT+1            ;>=
                        NEQ     =       GE+1            ;~=
                        LEFT1   =       NEQ+1           ;MONADIC <<
                        RIGHT1  =       LEFT1+1         ;MONADIC >>
                        NOT     =       RIGHT1+1        ;~
                        NEGATE  =       NOT+1           ;MONADIC -
                        NIL     =       NEGATE+1        ;MONADIC +
                        LPAREN  =       NIL+1           ;(
                        RPAREN  =       LPAREN+1        ;)
                        NUMBER  =       RPAREN+1        ;DECIMAL OR HEXADECIMAL CONSTANT
                        STRING  =       NUMBER+1        ;STRING CONSTANT
                        EOFILE  =       STRING+1        ;END OF EXPRESSION
                        DUMMY   =       EOFILE+1        ;INITIAL PREVIOUS TOKEN

                        ;-----
                        ; ADDAHL - ADD A TO HL
                        ; SAVES A!
                        ;-----
0238 C5         ADDAHL: PUSH    B
0239 4F                 MOV     C,A
023A 06 00              MVI     B,0
023C 09                 DAD     B
023D C1                 POP     B
```

```
                        RET
                ;-----
                ; AHEAD - LOOK AHEAD FOR A TWO CHARACTER TOKEN
                ; ACCEPTS:  D = CURRENT TOKEN CODE
                ;           B = CHARACTER TO LOOK FOR
                ;           C = NEW TOKEN CODE IF CHARACTER FOUND
                ; RETURNS:  A = NEW TOKEN CODE IF CHARACTER FOUND ELSE
                ;             = CURRENT TOKEN CODE
                ;-----
AHEAD:  CALL    GETC            ;LOOK AT NEXT CHARACTER
        CMP     B               ;IS IT WHAT WE'RE LOOKING FOR?
        JNZ     TOOFAR          ;NO
        MOV     A,C             ;YES, GET NEW TOKEN CODE
        RET
;
; LOOK AHEAD FAILED
;
TOOFAR: CALL    BACKUP          ;PUT BACK THE CHARACTER
        MOV     A,D             ;RESTORE CURRENT TOKEN CODE
        RET

;-----
; BACKUP - BACKUP TO PREVIOUS EXPRESSION CHARACTER
;-----
BACKUP: PUSH    H
        LHLD    GETPTR          ;GET ADDRESS OF NEXT CHARACTER
        DCX     H               ;GET ADDRESS OF PREVIOUS CHARACTER
        SHLD    GETPTR          ;SAVE IT
        POP     H
        RET

;-----
; COMPARE - COMPARE TWO OPERANDS
; ACCEPTS:  A = TYPE OF BOTH OPERANDS (NUMRAN OR STRRAN)
;          DE = 1ST OPERAND
;          HL = 2ND OPERAND
; RETURNS:  CONDITION BITS SET APPROPRIATELY
;          HL = 1
; IF THE OPERANDS ARE STRINGS, THEY ARE TREATED AS
; POINTERS TO STRINGS
;-----
COMPARE:
        CPI     STRRAN          ;ARE THE OPERANDS STRINGS?
        JZ      STLOOP          ;YES, HANDLE SEPARATELY
        CALL    DE_HL           ;NO, COMPARE
        LXI     H,1             ;SET HL FOR CALLER
        RET

STLOOP: LDAX    D               ;GET CHARACTER FROM 1ST STRING
        MOV     B,A             ;SAVE COPY IN B
        CPI     EOS             ;END OF 1ST STRING?
        JZ      END1ST          ;YES
        INX     D               ;NO, POINT TO NEXT CHARACTER
        MVI     A,EOS
        CMP     M               ;END OF 2ND STRING?
        JZ      END2ND          ;YES
GET2ND: MOV     C,M             ;NO, GET CHARACTER FROM 2ND STRING
        INX     H               ;POINT TO NEXT CHARACTER
COMCHR: MOV     A,B
        CMP     C               ;COMPARE STRING CHARACTERS
        JZ      STLOOP          ;CONTINUE IF STILL EQUAL
        LXI     H,1             ;SET HL FOR CALLER
        RET

END1ST: CMP     M               ;END OF 2ND STRING?
        MVI     B,' '           ;(PAD 1ST STRING IN CASE 'NO')
        JNZ     GET2ND          ;NO, 2ND LONGER THAN 1ST
        LXI     H,1             ;SET HL FOR CALLER
        RET

END2ND: MVI     C,' '           ;1ST LONGER THAN 2ND, PAD 2ND
        JMP     COMCHR

;-----
; DE_HL - COMPARE DE WITH HL
;-----
DE_HL:  MOV     A,D             ;COMPARE HIGH ORDER BYTES
        CMP     H
        RNZ
        MOV     A,E             ;COMPARE LOW ORDER BYTES
        CMP     L
        RET

;-----
; DECDIG - TEST FOR A DECIMAL DIGIT
; ACCEPTS:  A = CHARACTER
; RETURNS:  CARRY SET IF A = '0'-'9'
;-----
DECDIG: CPI     '9'+1           ;IS IT A DIGIT?
        RNC                     ;NO, >'9'
        CPI     '0'             ;MAYBE, >='0'?
        CMC                     ;SET CARRY IF YES, CLEAR IF NOT
        RET

;-----
; DIVREM - DIVISION AND REMAINDER ROUTINE
; ACCEPTS:  DE = DIVIDEND
;           HL = DIVISOR
; RETURNS:  BC = DE/HL
;           HL = DE//HL
; ADAPTED FROM PALO ALTO TINY BASIC!
;-----
DIVREM: MOV     A,H             ;DIVISOR ZERO?
        ORA     L
        JZ      DIVBY0          ;YES
        XCHG
        PUSH    H
        MOV     L,H             ;DIVIDE H BY DE
        MVI     H,0
        CALL    DIV1
        MOV     B,C             ;SAVE RESULT IN B
        MOV     A,L             ;(REMAINDER+L)/DE
        POP     H
        MOV     H,A
DIV1:   MVI     C,-1            ;RESULT IN C
DIV2:   INR     C               ;DUMB ROUTINE,
        MOV     A,L             ;(HL := HL - DE)
        SUB     E               ;DIVIDES BY SUBTRACTING
        MOV     L,A
        MOV     A,H
        SBB     D
        MOV     H,A
        JNC     DIV2            ;AND COUNTING
        DAD     D
        RET

DIVBY0: LXI     H,MSG13         ;GET ADDRESS OF ERROR MESSAGE
        CALL    $ABORT

;-----
; DYADIC - TEST FOR DYADIC OPERATOR
; ACCEPTS:  B = MONADIC TOKEN CODE
;           C = DYADIC TOKEN CODE
; RETURNS:  A = CORRECT (MONADIC OR DYADIC) CODE
;-----
DYADIC: LDA     BEFORE          ;GET PREVIOUS TOKEN CODE
        CPI     DUMMY           ;WAS THERE ANY?
        JZ      UNARY           ;NO, SO THIS OPERATOR IS MONADIC
        CPI     LPAREN+1        ;YES, WAS IT ( OR LOWER ORDER?
        JC      UNARY           ;YES, OPERATOR IS MONADIC
        MOV     A,C             ;NO, RETURN DYADIC TOKEN CODE
        RET

UNARY:  MOV     A,B             ;RETURN MONADIC TOKEN CODE
        RET

;-----
; EVAL - EVALUATE THE EXPRESSION
; RETURNS:  A = TYPE (NUMRAN OR STRRAN)
;          HL = VALUE
;-----
EVAL:   CALL    GETOKN          ;GET NEXT TOKEN
        STA     BEFORE          ;REMEMBER FOR NEXT TIME AROUND
        CPI     LPAREN          ;IS IT A LEFT PARENTHESIS?
        JZ      E_LPAR          ;YES
        CPI     RPAREN          ;NO, IS IT A RIGHT PARENTHESIS?
        JZ      E_RPAR          ;YES
        CPI     EOFILE          ;NO, IS IT THE END OF THE EXPRESSION?
        JZ      E_EOF           ;YES
        CPI     NUMBER          ;NO, IS IT A NUMBER?
        JZ      E_ABS           ;YES
        CPI     STRING          ;NO, IS IT A STRING?
        JZ      E_STR           ;YES
;
; TOKEN IS AN OPERATOR ... WHILE THE OPERATOR STACK IS NOT EMPTY AND
; THE TOP OF THE STACK IS NOT ( DO:
; POP AND EXECUTE THE OPERATOR. THEN STACK THE NEW OPERATOR
;
        PUSH    PSW             ;SAVE THE OPERATOR TOKEN CODE
E_OPR:  LHLD    RTRPTR          ;GET OPERATOR STACK POINTER
        LXI     D,OPRATR        ;GET ADDRESS OF STACK
        CALL    DE_HL           ;STACK EMPTY?
        JZ      NO_OPS          ;YES
        DCX     H               ;NO, GET ADDRESS OF TOP OF STACK
        MOV     A,M             ;GET LAST OPERATOR
        CPI     LPAREN          ;IS IT '('?
        JZ      NO_OPS          ;YES
        CALL    POPOP           ;NO, POP AND EXECUTE OPERATOR
        JMP     E_OPR

NO_OPS: POP     PSW             ;GET THE OPERATOR CODE
        CALL    PSHRAT          ;PUSH IT ON THE STACK
        JMP     EVAL
;
; TOKEN IS A ( ... STACK IT
;
E_LPAR: CALL    PSHRAT          ;PUSH ONTO THE OPERATOR STACK
        JMP     EVAL
;
; TOKEN IS A ) ... WHILE A ( IS NOT SEEN (AND THEN IGNORED),
; POP AND EXECUTE OPERATORS
;
POPPING:
```

```
0314 CD ?? ??           CALL    POPOP           ;POP AND EXECUTE AN OPERATOR
0317 2A 33 02   E_RPAR: LHLD    RTRPTR          ;GET OPERATOR STACK POINTER
031A 11 CD 01           LXI     D,OPRATR        ;GET ADDRESS OF STACK
031D CD 8B 02           CALL    DE_HL           ;STACK EMPTY?
0320 CA ?? ??           JZ      ERRPAR          ;YES, ERROR
0323 2B                 DCX     H               ;NO, GET ADDRESS OF LAST OPERATOR
0324 7E                 MOV     A,M             ;GET LAST OPERATOR
0325 FE 15              CPI     LPAREN          ;IS IT '('?
0327 C2 14 03           JNZ     POPPING         ;NO, KEEP POPPING
032A 22 33 02           SHLD    RTRPTR          ;YES, POP IT
032D C3 CE 02           JMP     EVAL
0330
0330 21 1A 00   ERRPAR: LXI     H,MSG2          ;GET ADDRESS OF ERROR MESSAGE
0333 CD -- --           CALL    $ABORT
0336                    ;
0336                    ; END OF EXPRESSION ... EXECUTE REMAINING OPERATORS
0336                    ;
0336 CD ?? ??   EOFPOP: CALL    POPOP
0339 2A 33 02   E_EOF:  LHLD    RTRPTR          ;GET OPERATOR STACK POINTER
033C 11 CD 01           LXI     D,OPRATR        ;GET ADDRESS OF STACK
033F CD 8B 02           CALL    DE_HL           ;STACK EMPTY?
0342 C2 36 03           JNZ     EOFPOP          ;NO
0345 CD ?? ??           CALL    POPANY          ;GET WHAT'S LEFT ON THE STACK
0348 F5                 PUSH    PSW             ;SAVE OPERAND DESCRIPTORS
0349 E5                 PUSH    H
034A 2A 31 02           LHLD    RANPTR          ;GET OPERAND STACK POINTER
034D 11 69 01           LXI     D,OPRAND        ;GET ADDRESS OF OPERAND STACK
0350 CD 8B 02           CALL    DE_HL           ;STACK EMPTY?
0353 21 58 01           LXI     H,MSG17         ;GET ADDRESS OF MESSAGE IN CASE NOT
0356 C4 -- --           CNZ     $ABORT          ;NO, MISSING OPERATOR
0359 E1                 POP     H
035A F1                 POP     PSW
035B C9                 RET
035C                    ;
035C                    ; TOKEN IS A NUMBER ... STACK THE OPERAND
035C                    ;
035C CD ?? ??   E_ABS:  CALL    PSHRAN          ;STACK THE OPERAND
035F C3 CE 02           JMP     EVAL
0362                    ;
0362                    ; TOKEN IS A STRING ... STACK THE OPERAND
0362                    ;
0362 7C         E_STR:  MOV     A,H             ;PUSH ADDRESS OF STRING
0363 CD ?? ??           CALL    PSHBYT
0366 7D                 MOV     A,L
0367 CD ?? ??           CALL    PSHBYT
036A 3E 01              MVI     A,STRRAN        ;GET CODE FOR STRING OPERAND
036C CD ?? ??           CALL    PSHBYT          ;PUSH IT, TOO
036F C3 CE 02           JMP     EVAL
0372
0372                    ;-----
0372                    ; GETC - GET NEXT EXPRESSION CHARACTER
0372                    ; RETURNS:  A = NEXT CHARACTER
0372                    ;-----
0372 E5         GETC:   PUSH    H
0373 2A 36 02           LHLD    GETPTR          ;GET POINTER TO EXPRESSION STRING
0376 7E                 MOV     A,M             ;GET NEXT CHARACTER
0377 23                 INX     H               ;POINT TO FOLLOWING CHARACTER
0378 22 36 02           SHLD    GETPTR          ;FOR NEXT TIME AROUND
037B E1                 POP     H
037C C9                 RET
037D
037D                    ;-----
037D                    ; GETOKN -  GET NEXT TOKEN FROM THE EXPRESSION
037D                    ; RETURNS:  A = TOKEN CODE
037D                    ;         HL  = ADDRESS OF 1ST CHARACTER IF A STRING ELSE
037D                    ;         HL  = VALUE IF A NUMBER
037D                    ;-----
037D CD 72 03   GETOKN: CALL    GETC            ;GET NEXT CHARACTER
0380 47                 MOV     B,A             ;SAVE IT IN B
0381 FE 20              CPI     ' '             ;BLANK?
0383 CA 7D 03           JZ      GETOKN          ;YES, IGNORE
0386 FE 27              CPI     ''''            ;QUOTE?
0388 CA ?? ??           JZ      QUOTE           ;YES
038B FE 2B              CPI     '+'             ;NO, PLUS?
038D CA ?? ??           JZ      NILADD          ;YES
0390 FE 2D              CPI     '-'             ;NO, MINUS?
0392 CA ?? ??           JZ      NEGSUB          ;YES
0395 FE 7C              CPI     '|'             ;NO, OR?
0397 CA ?? ??           JZ      BAR             ;YES
039A FE 2F              CPI     '/'             ;NO, DIVIDE?
039C CA ?? ??           JZ      SLASH           ;YES
039F FE 3D              CPI     '='             ;NO, EQUALS?
03A1 CA ?? ??           JZ      EQUALS          ;YES
03A4 FE 7E              CPI     '~'             ;NO, NOT?
03A6 CA ?? ??           JZ      TILDE           ;YES
03A9 FE 3C              CPI     '<'             ;NO, LT?
03AB CA ?? ??           JZ      LANGLE          ;YES
03AE FE 3E              CPI     '>'             ;NO, GT?
03B0 CA ?? ??           JZ      RANGLE          ;YES
03B3 FE 58              CPI     'X'             ;NO, X?
03B5 CA ?? ??           JZ      XHXNUM          ;YES
03B8 FE 78              CPI     'x'             ;NO, x?
03BA CA ?? ??           JZ      XHXNUM          ;YES
03BD FE 23              CPI     '#'             ;NO, HEX?
03BF CA ?? ??           JZ      HEXNUM          ;YES
03C2 FE 00              CPI     EOS             ;NO, END OF EXPRESSION?
03C4 3E 19              MVI     A,EOFILE        ;(ASSUME SO:  GET TOKEN CODE)
03C6 C8                 RZ                      ;YES
03C7 78                 MOV     A,B             ;NO, RESTORE CHARACTER
03C8 FE 26              CPI     '&'             ;AND?
03CA 3E 07              MVI     A,LAND          ;(ASSUME SO)
03CC C8                 RZ                      ;YES
03CD 78                 MOV     A,B             ;NO,
03CE FE 28              CPI     '('             ;LEFT PAREN?
03D0 3E 15              MVI     A,LPAREN        ;(ASSUME SO)
03D2 C8                 RZ                      ;YES
03D3 78                 MOV     A,B             ;NO,
03D4 FE 29              CPI     ')'             ;RIGHT PAREN?
03D6 3E 16              MVI     A,RPAREN        ;(ASSUME SO)
03D8 C8                 RZ                      ;YES
03D9 78                 MOV     A,B             ;NO,
03DA FE 2A              CPI     '*'             ;TIMES?
03DC 3E 02              MVI     A,TIMES         ;(ASSUME SO)
03DE C8                 RZ                      ;YES
03DF 78                 MOV     A,B             ;NO,
03E0 FE 30              CPI     '0'             ;DIGIT?
03E2 DA ?? ??           JC      BADCHR          ;NO
03E5 FE 3A              CPI     '9'+1           ;MAYBE...
03E7 DA ?? ??           JC      DECNUM          ;YES
03EA                    ;
03EA                    ; ILLEGAL TOKEN CHARACTER
03EA                    ;
03EA 21 33 00   BADCHR: LXI     H,MSG3          ;GET ADDRESS OF ERROR MESSAGE
03ED CD -- --           CALL    $ABORT
03F0                    ;
03F0                    ; A SINGLE QUOTE STARTS A STRING CONSTANT
03F0                    ;
03F0 2A 36 02   QUOTE:  LHLD    GETPTR          ;GET ADDRESS OF FIRST STRING CHARACTER
03F3 E5                 PUSH    H               ;AND SAVE
03F4 CD 72 03   GETSTR: CALL    GETC            ;GET NEXT STRING CHARACTER
03F7 77                 MOV     M,A             ;KEEP STRING LEFT JUSTIFIED
03F8 23                 INX     H
03F9 FE 00              CPI     EOS             ;END OF EXPRESSION?
03FB CA ?? ??           JZ      STRERR          ;YES, ERROR
03FE FE 27              CPI     ''''            ;ENDING QUOTE?
0400 C2 F4 03           JNZ     GETSTR          ;NO, THIS CHARACTER IS PART OF THE STRING
0403 CD 72 03           CALL    GETC            ;MAYBE, WHAT'S AFTER THE QUOTE?
0406 FE 27              CPI     ''''            ;TWO QUOTES IN A ROW?
0408 CA F4 03           JZ      GETSTR          ;YES, QUOTE IS PART OF THE STRING
040B CD 4D 02           CALL    BACKUP          ;NO, WE WENT TOO FAR
040E 2B                 DCX     H               ;GET ADDRESS OF ENDING QUOTE
040F 36 00              MVI     M,EOS           ;REPLACE ENDING QUOTE WITH EOS
0411 2B                 DCX     H               ;GET ADDRESS OF LAST STRING CHARACTER
0412 EB                 XCHG                    ;INTO DE
0413 E1                 POP     H               ;RESTORE ADDRESS OF 1ST STRING CHARACTER
0414 CD 8B 02           CALL    DE_HL           ;ARE THERE ANY CHARACTERS IN THE STRING?
0417 DA 7D 03           JC      GETOKN          ;NO, NULL STRING, IGNORE
041A 3E 18              MVI     A,STRING        ;YES, VALID STRING
041C C9                 RET
041D
041D 21 5E 00   STRERR: LXI     H,MSG5          ;GET ADDRESS OF ERROR MESSAGE
0420 CD -- --           CALL    $ABORT
0423                    ;
0423                    ; A + IS MONADIC OR DYADIC
0423                    ;
0423 06 14      NILADD: MVI     B,NIL           ;GET TOKEN CODES FOR MONADIC VERSION
0425 0E 00              MVI     C,PLUS          ;AND DYADIC VERSION
0427 CD BD 02           CALL    DYADIC          ;SELECT CORRECT CODE
042A C9                 RET
042B                    ;
042B                    ; A - IS MONADIC OR DYADIC
042B                    ;
042B 06 13      NEGSUB: MVI     B,NEGATE        ;GET TOKEN CODES FOR MONADIC VERSION
042D 0E 01              MVI     C,MINUS         ;AND DYADIC VERSION
042F CD BD 02           CALL    DYADIC          ;CHECK OPERATOR TYPE
0432 C9                 RET
0433                    ;
0433                    ; A | IS THE | OPERATOR OR THE START OF THE || OPERATOR
0433                    ;
0433 0E 05      BAR:    MVI     C,XOR           ;GET TOKEN CODES FOR ||
0435 16 08              MVI     D,IOR           ;AND |
0437 CD 3F 02           CALL    AHEAD           ;LOOK AHEAD FOR SECOND |
043A C9                 RET
043B                    ;
043B                    ; A / IS THE / OPERATOR OR THE START OF THE // OPERATOR
043B                    ;
043B 0E 04      SLASH:  MVI     C,REMAIN        ;GET TOKEN CODES FOR //
043D 16 03              MVI     D,DIVIDE        ;AND /
043F CD 3F 02           CALL    AHEAD           ;LOOK AHEAD FOR SECOND /
0442 C9                 RET
0443                    ;
0443                    ; A = IS THE START OF THE == OPERATOR
0443                    ;
0443 16 1A      EQUALS: MVI     D,DUMMY         ;THERE'S NO VALID TOKEN CODE FOR A SINGLE =
0445 0E 0A              MVI     C,EQ            ;GET TOKEN CODE FOR ==
0447 CD 3F 02           CALL    AHEAD           ;LOOK AHEAD FOR SECOND =
044A FE 0A              CPI     EQ              ;IS IT THERE?
044C C8                 RZ                      ;YES
044D 21 17 01           LXI     H,MSG12         ;NO, GET ADDRESS OF ERROR MESSAGE
0450 CD -- --           CALL    $ABORT
0453                    ;
0453                    ; A ~ IS THE ~ OPERATOR OR THE START OF THE ~= OPERATOR
0453                    ;
0453 0E 0F      TILDE:  MVI     C,NEQ           ;GET TOKEN CODES FOR ~=
0455 16 12              MVI     D,NOT           ;AND ~
0457 06 3D              MVI     B,'='           ;SEE IF AN =
```

```
0459 CD 3F 02           CALL    AHEAD           ;IS AHEAD
045C C9                 RET
                ;
                ; A < IS THE < OPERATOR OR THE START OF THE << OR <= OPERATORS
                ;
045D 0E 05      LANGLE: MVI     C,LEFT          ;GET TOKEN CODES FOR <<
045F 16 0B              MVI     D,LT            ;AND FOR <
0461 CD 3F 02           CALL    AHEAD           ;LOOK AHEAD FOR SECOND <
0464 FE 05              CPI     LEFT            ;IS IT THERE?
0466 CA ?? ??           JZ      L_MORD          ;YES, DETERMINE IF MONADIC OR DYADIC
0469 0E 0C              MVI     C,LE            ;NO, GET TOKEN CODE FOR <=
046B 06 3D              MVI     B,'='           ;SEE IF AN =
046D CD 3F 02           CALL    AHEAD           ;IS AHEAD
0470 C9                 RET
0471
0471 06 10      L_MORD: MVI     B,LEFT1         ;GET CODE FOR MONADIC VERSION
0473 CD BD 02           CALL    DYADIC          ;CHECK OPERATOR TYPE
0476 C9                 RET
                ;
                ; A > IS THE > OPERATOR OR THE START OF THE >> OR >= OPERATORS
                ;
0477 0E 06      RANGLE: MVI     C,RIGHT         ;GET TOKEN CODES FOR >>
0479 16 0D              MVI     D,GT            ;AND >
047B CD 3F 02           CALL    AHEAD           ;LOOK AHEAD FOR SECOND >
047E FE 06              CPI     RIGHT           ;IS IT THERE?
0480 CA ?? ??           JZ      R_MORD          ;YES, DETERMINE IF MONADIC OR DYADIC
0483 0E 0E              MVI     C,GE            ;NO, GET TOKEN CODE FOR >=
0485 06 3D              MVI     B,'='           ;SEE IF AN =
0487 CD 3F 02           CALL    AHEAD           ;IS AHEAD
048A C9                 RET
048B
048B 06 11      R_MORD: MVI     B,RIGHT1        ;GET CODE FOR MONADIC VERSION
048D CD BD 02           CALL    DYADIC          ;CHECK OPERATOR TYPE
0490 C9                 RET
                ;
                ; A DECIMAL DIGIT STARTS A DECIMAL CONSTANT.  COLLECT ALL FOLLOWING
                ; DIGITS TO FORM A NUMBER
                ;
0491 D6 30      DECNUM: SUI     '0'             ;CONVERT TO BINARY
0493 6F                 MOV     L,A             ;MOVE DIGIT TO HL
0494 26 00              MVI     H,0             ;TO FORM INITIAL VALUE
0496 CD 72 03   NXTDEC: CALL    GETC            ;GET THE NEXT DIGIT
0499 CD 91 02           CALL    DECDIG          ;IS IT 0-9?
049C D2 ?? ??           JNC     NOTDEC          ;NO
049F 29                 DAD     H               ;YES, DOUBLE OLD VALUE
04A0 54                 MOV     D,H             ;AND SAVE
04A1 5D                 MOV     E,L             ;IN DE
04A2 29                 DAD     H               ;QUADRUPLE
04A3 29                 DAD     H               ;OCTUPLE
04A4 19                 DAD     D               ;ADD DOUBLE. HL HAS BEEN MULTIPLIED BY 10
04A5 D6 30              SUI     '0'             ;CONVERT NEXT DIGIT TO BINARY
04A7 CD 38 02           CALL    ADDAHL          ;ADD TO MULTIPLIED VALUE
04AA C3 96 04           JMP     NXTDEC
04AD            ;
                ; END OF DECIMAL DIGITS
                ;
04AD CD 4D 02   NOTDEC: CALL    BACKUP          ;PUSH NON-DIGIT BACK ONTO INPUT
04B0 3E 17              MVI     A,NUMBER        ;GET TOKEN CODE
04B2 C9                 RET
04B3            ;
                ; THE LETTER X IS THE START OF A HEX CONSTANT
                ;
04B3 CD 72 03   XHXNUM: CALL    GETC            ;GET THE LETTER AFTER X
04B6 FE 27              CPI     ''''            ;IS IT A QUOTE?
04B8 CA ?? ??           JZ      HEXNUM          ;YES, WE HAVE A HEX CONSTANT
04BB 21 7A 00           LXI     H,MSG6          ;NO, GET ADDRESS OF ERROR MESSAGE
04BE CD -- --           CALL    $ABORT
04C1            ;
                ; AN X' OR A # HAS BEEN SEEN. COLLECT ALL FOLLOWING HEX DIGITS TO FORM A NUMBER
                ;
04C1 C5         HEXNUM: PUSH    B               ;SAVE FIRST CHARACTER
04C2 CD 72 03           CALL    GETC            ;GET THE FIRST DIGIT
04C5 CD ?? ??           CALL    HEXDIG          ;IS IT 0-9 OR A-F?
04C8 21 9B 00           LXI     H,MSG7          ;(ASSUME NOT, GET MESSAGE ADDRESS)
04CB D4 -- --           CNC     $ABORT          ;NO
04CE 6F                 MOV     L,A             ;YES, MOVE DIGIT
04CF 26 00              MVI     H,0             ;TO FORM INITIAL VALUE
04D1 CD 72 03   NXTHEX: CALL    GETC            ;GET NEXT DIGIT
04D4 CD ?? ??           CALL    HEXDIG          ;IS IT 0-9 OR A-F?
04D7 D2 ?? ??           JNC     NOTHEX          ;NO
04DA 29                 DAD     H               ;YES,
04DB 29                 DAD     H               ;MULTIPLY
04DC 29                 DAD     H               ;OLD VALUE
04DD 29                 DAD     H               ;BY 16
04DE CD 38 02           CALL    ADDAHL          ;ADD IN NEW DIGIT
04E1 C3 D1 04           JMP     NXTHEX
04E4            ;
                ; END OF CONSTANT FOUND
                ;
04E4 C1         NOTHEX: POP     B               ;RESTORE FIRST CHARACTER
04E5 4F                 MOV     C,A             ;SAVE THE NON-DIGIT IN C
04E6 78                 MOV     A,B             ;WAS X'
04E7 FE 58              CPI     'X'             ;THE CONSTANT'S PREFIX?
04E9 CA ?? ??           JZ      QUOTED          ;YES
04EC FE 78              CPI     'x'             ;MAYBE...
04EE CA ?? ??           JZ      QUOTED          ;YES
04F1 CD 4D 02           CALL    BACKUP          ;NO, PUSH BACK ONTO INPUT
04F4 3E 17      RETNUM: MVI     A,NUMBER        ;GET APPROPRIATE TOKEN CODE
04F6 C9                 RET
04F7            ;
04F7            ; HEX CONSTANT SHOULD END WITH QUOTE
04F7            ;
04F7 3E 27      QUOTED: MVI     A,''''          ;IS '
04F9 B9                 CMP     C               ;THE END OF THE CONSTANT?
04FA CA F4 04           JZ      RETNUM          ;YES, CONSTANT IS WELL FORMED
04FD 21 AD 00           LXI     H,MSG8          ;NO, GET ADDRESS OF ERROR MESSAGE
0500 CD -- --           CALL    $ABORT
0503
0503            ;-----
0503            ; HEXDIG - TEST FOR HEX DIGIT
0503            ; ACCEPTS:  A = CHARACTER
0503            ; RETURNS:  CARRY SET IF CHARACTER = '0'-'9', 'A'-'Z', OR 'a'-'z'
0503            ;           (A = BINARY VALUE)
0503            ;           ELSE CARRY CLEARED (A SAVED)
0503            ;-----
0503 CD 91 02   HEXDIG: CALL    DECDIG          ;IS IT '0'-'9'?
0506 D2 ?? ??           JNC     NOT1_9          ;NO
0509 D6 30              SUI     '0'             ;YES, CONVERT TO BINARY
050B 37                 STC
050C C9                 RET
050D
050D FE 67      NOT1_9: CPI     'f'+1           ;IS IT > 'f'?
050F D0                 RNC                     ;YES
0510 FE 61              CPI     'a'             ;NO, IS IT < 'a'?
0512 DA ?? ??           JC      NOTA_F          ;YES
0515 D6 57              SUI     'a'-#A          ;NO, CONVERT TO BINARY
0517 37                 STC
0518 C9                 RET
0519
0519 FE 47      NOTA_F: CPI     'F'+1           ;IS IT > 'F'?
051B D0                 RNC                     ;YES
051C FE 41              CPI     'A'             ;NO, IS IT < 'A'?
051E 3F                 CMC                     ;SET CARRY IF NOT, CLEAR IF YES
051F D0                 RNC                     ;YES, < 'A'
0520 D6 37              SUI     'A'-#A          ;NO, CONVERT TO BINARY
0522 37                 STC
0523 C9                 RET
0524
0524            ;-----
0524            ; INIT - INITIALIZE VARIABLES
0524            ;-----
0524 7E         INIT:   MOV     A,M             ;GET NEXT CHARACTER
0525 23                 INX     H               ;GET ADDRESS OF NEXT CHARACTER
0526 FE 00              CPI     EOS             ;END OF EXPRESSION?
0528 CA ?? ??           JZ      ARGERR          ;YES, EXPRESSION MISSING
052B FE 20              CPI     ' '             ;NO, IS IT BLANK?
052D CA 24 05           JZ      INIT            ;YES, KEEP SEARCHING
0530 2B                 DCX     H               ;NO,
0531 22 36 02           SHLD    GETPTR          ;SAVE ADDRESS OF NEXT CHARACTER
0534 3E 1A              MVI     A,DUMMY         ;INITIALIZE THE PREVIOUS TOKEN
0536 32 35 02           STA     BEFORE          ;TO NOTHING
0539 21 69 01           LXI     H,OPRAND        ;EMPTY THE
053C 22 31 02           SHLD    RANPTR          ;OPERAND STACK
053F 21 CD 01           LXI     H,OPRATR        ;AND THE
0542 22 33 02           SHLD    RTRPTR          ;OPERATOR STACK
0545 C9                 RET
0546
0546 21 07 00   ARGERR: LXI     H,MSG1          ;GET @ ERROR MESSAGE
0549 CD -- --           CALL    $ABORT
054C
054C            ;-----
054C            ; POP1 - POP ONE 16 BIT NUMERIC OPERAND
054C            ; RETURNS:  A = NUMERIC OPERAND TYPE CODE (NUMRAN)
054C            ;          HL = OPERAND VALUE
054C            ;-----
054C CD ?? ??   POP1:   CALL    POPANY          ;POP ANY ONE OPERAND
054F FE 00              CPI     NUMRAN          ;IS IT A NUMBER?
0551 C8                 RZ                      ;YES
0552 6E                 MOV     L,M             ;NO,
0553 26 00              MVI     H,0             ;CREATE VALUE FROM 1ST STRING CHARACTER
0555 3E 00              MVI     A,NUMRAN        ;GET TYPE CODE
0557 C9                 RET
0558
0558            ;-----
0558            ; POP2 - POP TWO 16 BIT NUMERIC OPERANDS
0558            ; RETURNS:  A = NUMERIC OPERAND TYPE CODE (NUMRAN)
0558            ;           DE = 1ST OPERAND'S VALUE
0558            ;           HL = 2ND OPERAND'S VALUE
0558            ;-----
0558 CD 4C 05   POP2:   CALL    POP1            ;GET ONE NUMERIC OPERAND
055B EB                 XCHG                    ;SAVE 2ND'S VALUE IN DE
055C CD 4C 05           CALL    POP1            ;GET 1ST OPERAND
055F EB                 XCHG                    ;DE = 1ST'S VALUE, HL = 2ND'S VALUE
0560 C9                 RET
0561
0561            ;-----
0561            ; POP2NS - POP TWO OPERANDS (NUMBERS OR STRINGS) FOR
0561            ;          USE WITH A RELATIONAL OPERATOR
0561            ; RETURNS: A = OPERAND'S TYPE CODE (NUMRAN OR STRRAN)
0561            ;          DE = 1ST OPERAND VALUE
0561            ;          HL = 2ND OPERAND VALUE
0561            ; BOTH OPERANDS HAVE THE SAME TYPE.  IF THEY ARE STRINGS,
0561            ; THE VALUES ARE POINTERS TO THE STRINGS
0561            ;-----
```

```
0561 CD ?? ??   POP2NS: CALL    POPANY          ;POP ONE NUMBER OR STRING
0564 4F                 MOV     C,A             ;SAVE TYPE OF 2ND OPERAND IN C
0565 EB                 XCHG                    ;AND VALUE IN DE
0566 CD ?? ??           CALL    POPANY          ;POP 1ST OPERAND
0569 47                 MOV     B,A             ;SAVE TYPE IN B
056A EB                 XCHG                    ;DE = 1ST'S VALUE, HL = 2ND'S VALUE
056B 3E 01              MVI     A,STRRAN        ;GET CODE FOR STRING OPERAND
056D B8                 CMP     B               ;1ST OPERAND A STRING?
056E CA ?? ??           JZ      ONESTR          ;YES
0571 B9                 CMP     C               ;NO, 2ND OPERAND A STRING?
0572 3E 00              MVI     A,NUMRAN        ;GET OPERAND CODE IN CASE "NO"
0574 C0                 RNZ                     ;NO
0575 6E                 MOV     L,M             ;YES,
0576 26 00              MVI     H,0             ;CREATE VALUE FROM 1ST STRING CHARACTER
0578 C9                 RET
0579                ;
0579                ; 1ST OPERAND IS A STRING
0579                ;
0579 B9             ONESTR: CMP     C               ;2ND OPERAND A STRING?
057A C8                 RZ                      ;YES, BOTH ARE
057B EB                 XCHG                    ;NO,
057C 6E                 MOV     L,M             ;CREATE VALUE FROM 1ST STRING CHARACTER
057D 26 00              MVI     H,0
057F EB                 XCHG
0580 3E 00              MVI     A,NUMRAN
0582 C9                 RET
0583
0583                ;-----
0583                ; POPANY - POP ANY ONE OPERAND (NUMBER OR STRING)
0583                ; RETURNS:  A = OPERAND TYPE CODE (NUMRAN OR STRRAN)
0583                ;           HL = OPERAND VALUE
0583                ; IF THE OPERAND IS A STRING, ITS VALUE IS A POINTER
0583                ; TO THE STRING
0583                ;-----
0583 D5             POPANY: PUSH    D
0584 2A 31 02           LHLD    RANPTR          ;GET OPERAND STACK POINTER
0587 11 69 01           LXI     D,OPRAND        ;GET ADDRESS OF STACK
058A CD 8B 02           CALL    DE_HL           ;STACK EMPTY?
058D CA ?? ??           JZ      ERRANY          ;YES
0590 2B                 DCX     H               ;NO,
0591 7E                 MOV     A,M             ;POP TYPE CODE
0592 2B                 DCX     H
0593 5E                 MOV     E,M             ;GET VALUE INTO DE
0594 2B                 DCX     H
0595 56                 MOV     D,M
0596 22 31 02           SHLD    RANPTR          ;SAVE NEW STACK POINTER
0599 EB                 XCHG                    ;MOVE VALUE INTO HL
059A D1                 POP     D
059B C9                 RET
059C
059C 21 40 01       ERRANY: LXI     H,MSG16
059F CD -- --           CALL    $ABORT
05A2
05A2                ;-----
05A2                ; POPOP - POP OPERATOR AND EXECUTE
05A2                ;-----
05A2 2A 33 02       POPOP:  LHLD    RTRPTR          ;GET OPERATOR STACK POINTER
05A5 11 CD 01           LXI     D,OPRATR        ;GET ADDRESS OF STACK
05A8 CD 8B 02           CALL    DE_HL           ;STACK EMPTY?
05AB C2 ?? ??           JNZ     NOTEMP          ;NO
05AE 21 45 00           LXI     H,MSG4          ;YES, ERROR
05B1 CD -- --           CALL    $ABORT
05B4
05B4 2B             NOTEMP: DCX     H               ;GET ADDRESS OF LAST OPERATOR
05B5 22 33 02           SHLD    RTRPTR          ;SAVE NEW STACK POINTER
05B8 7E                 MOV     A,M             ;GET OPERATOR
05B9 21 ?? ??           LXI     H,POPTBL        ;GET ADDRESS OF DISPATCH TABLE
05BC CD 38 02           CALL    ADDAHL          ;GET POINTER
05BF CD 38 02           CALL    ADDAHL          ;INTO TABLE
05C2 5E                 MOV     E,M             ;GET TABLE ENTRY
05C3 23                 INX     H
05C4 56                 MOV     D,M             ;INTO DE
05C5 EB                 XCHG                    ;MOVE DISPATCH ADDRESS TO HL
05C6 E9                 PCHL                    ;JUMP TO OPERATOR ROUTINE
05C7                ;
05C7                ; DYADIC +
05C7                ;
05C7 CD 58 05       P_ADD:  CALL    POP2            ;GET TWO OPERANDS OFF THE STACK
05CA 19             P_ADD3: DAD     D               ;ADD THE OPERANDS
05CB CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
05CE C9                 RET
05CF                ;
05CF                ; DYADIC -
05CF                ;
05CF CD 58 05       P_SUB:  CALL    POP2            ;GET TWO OPERANDS OFF THE STACK
05D2 7B             P_SUB2: MOV     A,E             ;HL := DE - HL
05D3 95                 SUB     L
05D4 6F                 MOV     L,A
05D5 7A                 MOV     A,D
05D6 9C                 SBB     H
05D7 67                 MOV     H,A
05D8 CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
05DB C9                 RET
05DC                ;
05DC                ; *
05DC                ;
05DC CD 58 05       P_MUL:  CALL    POP2            ;GET TWO OPERANDS OFF THE STACK
05DF 44                 MOV     B,H             ;HL := DE * HL
05E0 4D                 MOV     C,L
05E1 21 00 00           LXI     H,0
05E4 7A             P_MUL1: MOV     A,D
05E5 B3                 ORA     E
05E6 CA ?? ??           JZ      P_MUL2
05E9 09                 DAD     B
05EA 1B                 DCX     D
05EB C3 E4 05           JMP     P_MUL1
05EE CD ?? ??       P_MUL2: CALL    PSHRAN          ;PUSH RESULT ON STACK
05F1 C9                 RET
05F2                ;
05F2                ; /
05F2                ;
05F2 CD 58 05       P_DIV:  CALL    POP2            ;GET TWO OPERANDS OFF STACK
05F5 CD 98 02           CALL    DIVREM          ;BC := DE/HL
05F8 60                 MOV     H,B
05F9 69                 MOV     L,C
05FA CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
05FD C9                 RET
05FE                ;
05FE                ; //
05FE                ;
05FE CD 58 05       P_REM:  CALL    POP2            ;GET TWO OPERANDS OFF STACK
0601 CD 98 02           CALL    DIVREM          ;HL := DE//HL
0604 CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
0607 C9                 RET
0608                ;
0608                ; DYADIC <<
0608                ;
0608 CD 58 05       P_LEFT: CALL    POP2            ;GET TWO OPERANDS OFF STACK
060B 7A             P_LFT1: MOV     A,D             ;DONE ROTATING?
060C B3                 ORA     E               ;(COUNT ZERO?)
060D CA ?? ??           JZ      P_LFT2          ;YES
0610 7C                 MOV     A,H             ;NO, << HL
0611 17                 RAL
0612 7D                 MOV     A,L
0613 17                 RAL
0614 6F                 MOV     L,A
0615 7C                 MOV     A,H
0616 17                 RAL
0617 67                 MOV     H,A
0618 1B                 DCX     D               ;DECREMENT COUNT
0619 C3 0B 06           JMP     P_LFT1
061C CD ?? ??       P_LFT2: CALL    PSHRAN          ;PUSH RESULT ON STACK
061F C9                 RET
0620                ;
0620                ; DYADIC >>
0620                ;
0620 CD 58 05       P_RGHT: CALL    POP2            ;GET TWO OPERANDS OFF STACK
0623 7A             P_RGT1: MOV     A,D             ;DONE ROTATING?
0624 B3                 ORA     E               ;(COUNT ZERO?)
0625 CA ?? ??           JZ      P_RGT2          ;YES
0628 7C                 MOV     A,H             ;NO, >> HL
0629 1F                 RAR
062A 7D                 MOV     A,L
062B 1F                 RAR
062C 6F                 MOV     L,A
062D 7C                 MOV     A,H
062E 1F                 RAR
062F 67                 MOV     H,A
0630 1B                 DCX     D               ;DECREMENT COUNT
0631 C3 23 06           JMP     P_RGT1
0634 CD ?? ??       P_RGT2: CALL    PSHRAN          ;PUSH RESULT ON STACK
0637 C9                 RET
0638                ;
0638                ; &
0638                ;
0638 CD 58 05       P_AND:  CALL    POP2            ;POP TWO OPERANDS OFF STACK
063B 7A                 MOV     A,D             ;HL := DE & HL
063C A4                 ANA     H
063D 67                 MOV     H,A
063E 7B                 MOV     A,E
063F A5                 ANA     L
0640 6F                 MOV     L,A
0641 CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
0644 C9                 RET
0645                ;
0645                ; |
0645                ;
0645 CD 58 05       P_IOR:  CALL    POP2            ;POP TWO OPERANDS OFF STACK
0648 7A                 MOV     A,D             ;HL := DE | HL
0649 B4                 ORA     H
064A 67                 MOV     H,A
064B 7B                 MOV     A,E
064C B5                 ORA     L
064D 6F                 MOV     L,A
064E CD ?? ??           CALL    PSHRAN          ;PUSH RESULT ON STACK
0651 C9                 RET
0652                ;
0652                ; ||
0652                ;
0652 CD 58 05       P_XOR:  CALL    POP2            ;POP TWO OPERANDS OFF STACK
0655 7A                 MOV     A,D             ;HL := DE || HL
0656 AC                 XRA     H
0657 67                 MOV     H,A
0658 7B                 MOV     A,E
0659 AD                 XRA     L
065A 6F                 MOV     L,A
```

```
0876 065B CD ?? ??          CALL    PSHRAN          ;PUSH RESULT ON STACK
0877 065E C9                RET
0878 065F               ;
0879 065F               ; ==
0880 065F               ;
0881 065F CD 61 05      P_EQ:   CALL    POP2NS          ;POP ANY TWO OPERANDS
0882 0662 CD 57 02              CALL    COMPARE         ;DE == HL?
0883 0665 CA ?? ??              JZ      P_EQ1           ;YES, RESULT IS 1
0884 0668 2B            P_EQ0:  DCX     H               ;NO, RESULT IS 0
0885 0669 CD ?? ??      P_EQ1:  CALL    PSHRAN          ;PUSH RESULT
0886 066C C9                    RET
0887 066D               ;
0888 066D               ; ~=
0889 066D               ;
0890 066D CD 61 05      P_NEQ:  CALL    POP2NS
0891 0670 CD 57 02              CALL    COMPARE
0892 0673 C2 69 06              JNZ     P_EQ1           ;DE ~= HL
0893 0676 C3 68 06              JMP     P_EQ0           ;DE == HL
0894 0679               ;
0895 0679               ; <
0896 0679               ;
0897 0679 CD 61 05      P_LT:   CALL    POP2NS
0898 067C CD 57 02      P_LT0:  CALL    COMPARE
0899 067F DA 69 06              JC      P_EQ1           ;DE < HL
0900 0682 C3 68 06              JMP     P_EQ0           ;DE >= HL
0901 0685               ;
0902 0685               ; <=
0903 0685               ;
0904 0685 CD 61 05      P_LE:   CALL    POP2NS
0905 0688 CD 57 02      P_LE0:  CALL    COMPARE
0906 068B DA 69 06              JC      P_EQ1           ;DE < HL (DE <= HL)
0907 068E CA 69 06              JZ      P_EQ1           ;DE == HL (DE <= HL)
0908 0691 C3 68 06              JMP     P_EQ0           ;DE > HL
0909 0694               ;
0910 0694               ; >
0911 0694               ;
0912 0694 CD 61 05      P_GT:   CALL    POP2NS
0913 0697 EB                    XCHG                    ;SWAP,
0914 0698 C3 88 06              JMP     P_LE0           ;USE <=
0915 069B               ;
0916 069B               ; >=
0917 069B               ;
0918 069B CD 61 05      P_GE:   CALL    POP2NS
0919 069E EB                    XCHG                    ;SWAP,
0920 069F C3 7C 06              JMP     P_LT0           ;USE <
0921 06A2               ;
0922 06A2               ; MONADIC <<
0923 06A2               ;
0924 06A2 CD 4C 05      P_MLF:  CALL    POP1            ;POP ONE OPERAND OFF STACK
0925 06A5 11 01 00              LXI     D,1             ;PROVIDE FIRST OPERAND
0926 06A8 C3 0B 06              JMP     P_LFT1          ;JUMP INTO DYADIC ROUTINE
0927 06AB               ;
0928 06AB               ; MONADIC >>
0929 06AB               ;
0930 06AB CD 4C 05      P_MRT:  CALL    POP1            ;POP ONE OPERAND OFF STACK
0931 06AE 11 01 00              LXI     D,1             ;PROVIDE FIRST OPERAND
0932 06B1 C3 23 06              JMP     P_RGT1          ;JUMP INTO DYADIC ROUTINE
0933 06B4               ;
0934 06B4               ; ~
0935 06B4               ;
0936 06B4 CD 4C 05      P_NOT:  CALL    POP1            ;POP ONE OPERAND OFF STACK
0937 06B7 7C                    MOV     A,H             ;HL := ~HL
0938 06B8 2F                    CMA
0939 06B9 67                    MOV     H,A
0940 06BA 7D                    MOV     A,L
0941 06BB 2F                    CMA
0942 06BC 6F                    MOV     L,A
0943 06BD CD ?? ??              CALL    PSHRAN          ;PUSH RESULT ON STACK
0944 06C0 C9                    RET
0945 06C1               ;
0946 06C1               ; MONADIC -
0947 06C1               ;
0948 06C1 CD 4C 05      P_NEG:  CALL    POP1            ;GET ONE OPERAND OFF STACK
0949 06C4 11 00 00              LXI     D,0             ;PROVIDE FIRST OPERAND
0950 06C7 C3 D2 05              JMP     P_SUB2          ;JUMP INTO DYADIC ROUTINE
0951 06CA               ;
0952 06CA               ; MONADIC +
0953 06CA               ;
0954 06CA CD 4C 05      P_POS:  CALL    POP1            ;GET ONE OPERAND OFF STACK
0955 06CD 11 00 00              LXI     D,0             ;PROVIDE FIRST OPERAND
0956 06D0 C3 CA 05              JMP     P_ADD3          ;JUMP INTO DYADIC ROUTINE
0957 06D3               ;
0958 06D3               ; (   (UNEXECUTABLE)
0959 06D3               ;
0960 06D3 21 CF 00      P_LPAR: LXI     H,MSG9          ;GET ADDRESS OF ERROR MESSAGE
0961 06D6 CD -- --              CALL    $ABORT
0962 06D9
0963 06D9               ;-----
0964 06D9               ; PSHBYT - PUSH ONE BYTE ONTO OPERAND STACK
0965 06D9               ; ACCEPTS:  A = BYTE
0966 06D9               ;-----
0967 06D9 D5            PSHBYT: PUSH    D
0968 06DA E5                    PUSH    H
0969 06DB 47                    MOV     B,A             ;SAVE THE BYTE IN B
0970 06DC 2A 31 02              LHLD    RANPTR          ;GET OPERAND STACK POINTER
0971 06DF 11 CD 01              LXI     D,OPRAND+OPRSIZ ;GET (ADDRESS OF END OF STACK) + 1
0972 06E2 CD 8B 02              CALL    DE_HL           ;STACK FULL?
0973 06E5 CA ?? ??              JZ      RANFUL          ;YES
0974 06E8 70                    MOV     M,B             ;NO, STACK THE BYTE
0975 06E9 23                    INX     H
0976 06EA 22 31 02              SHLD    RANPTR          ;SAVE NEW STACK POINTER
0977 06ED E1                    POP     H
0978 06EE D1                    POP     D
0979 06EF C9                    RET
0980 06F0
0981 06F0 21 E8 00      RANFUL: LXI     H,MSG10
0982 06F3 CD -- --              CALL    $ABORT
0983 06F6
0984 06F6               ;-----
0985 06F6               ; PSHRAN - PUSH ONE 16 BIT NUMERIC OPERAND ONTO OPERAND STACK
0986 06F6               ; ACCEPTS: HL = OPERAND
0987 06F6               ;-----
0988 06F6 3E 00         PSHRAN: MVI     A,NUMRAN        ;GET NUMERIC OPERAND CODE
0989 06F8 F5                    PUSH    PSW             ;SAVE THE TYPE CODE
0990 06F9 7C                    MOV     A,H             ;GET FIRST OPERAND BYTE
0991 06FA CD D9 06              CALL    PSHBYT          ;PUSH THE FIRST BYTE
0992 06FD 7D                    MOV     A,L             ;GET THE SECOND BYTE
0993 06FE CD D9 06              CALL    PSHBYT          ;PUSH IT, TOO
0994 0701 F1                    POP     PSW             ;RESTORE TYPE CODE
0995 0702 CD D9 06              CALL    PSHBYT          ;FINAL PUSH
0996 0705 C9                    RET
0997 0706
0998 0706               ;-----
0999 0706               ; PSHRAT - PUSH OPERATOR ONTO OPERATOR STACK
1000 0706               ; ACCEPTS:  A = OPERATOR CODE
1001 0706               ;-----
1002 0706 47            PSHRAT: MOV     B,A             ;SAVE THE OPERATOR CODE IN B
1003 0707 2A 33 02              LHLD    RTRPTR          ;GET OPERATOR STACK POINTER
1004 070A 11 31 02              LXI     D,OPRATR+OPRSIZ ;GET (ADDRESS OF END OF STACK) + 1
1005 070D CD 8B 02              CALL    DE_HL           ;STACK FULL?
1006 0710 CA ?? ??              JZ      RATFUL          ;YES
1007 0713 70                    MOV     M,B             ;NO, STACK THE OPERATOR
1008 0714 23                    INX     H
1009 0715 22 33 02              SHLD    RTRPTR          ;SAVE NEW STACK POINTER
1010 0718 C9                    RET
1011 0719
1012 0719 21 FF 00      RATFUL: LXI     H,MSG11
1013 071C CD -- --              CALL    $ABORT
1014 071F
1015 071F               ;-----
1016 071F               ; POPOP DISPATCH TABLE...
1017 071F               ;-----
1018 071F C7 05         POPTBL: @ P_ADD         ;DYADIC +
1019 0721 CF 05                 @ P_SUB         ;DYADIC -
1020 0723 DC 05                 @ P_MUL         ;*
1021 0725 F2 05                 @ P_DIV         ;/
1022 0727 FE 05                 @ P_REM         ;//
1023 0729 08 06                 @ P_LEFT        ;DYADIC <<
1024 072B 20 06                 @ P_RGHT        ;DYADIC >>
1025 072D 38 06                 @ P_AND         ;&
1026 072F 45 06                 @ P_IOR         ;|
1027 0731 52 06                 @ P_XOR         ;||
1028 0733 5F 06                 @ P_EQ          ;==
1029 0735 79 06                 @ P_LT          ;<
1030 0737 85 06                 @ P_LE          ;<=
1031 0739 94 06                 @ P_GT          ;>
1032 073B 9B 06                 @ P_GE          ;>=
1033 073D 6D 06                 @ P_NEQ         ;~=
1034 073F A2 06                 @ P_MLF         ;MONADIC <<
1035 0741 AB 06                 @ P_MRT         ;MONADIC >>
1036 0743 B4 06                 @ P_NOT         ;~
1037 0745 C1 06                 @ P_NEG         ;MONADIC -
1038 0747 CA 06                 @ P_POS         ;MONADIC +
1039 0749 D3 06                 @ P_LPAR        ;(
1040 074B
```

Letters

A MICRO-BASED PROJECT FOR THE HANDICAPPED

Dear Madam or Sir: *Received: 78 July 31*

The Spain Rehabilitation Center at the University of Alabama Medical Center has a project underway to demonstrate both the utility and economic feasibility of the new generation of 'personal' computers for use by the severely disabled. The programmability of the computer will allow it to serve as a general purpose appliance to be used as an aid to communication and education as well as for environmental control and entertainment.

This system, as currently envisioned, will consist of a microcomputer, an on-line storage device for programs and data, two T.V. monitors for user feedback and information display, a printing device for typed output, a speech recognition device for vocal input of commands, data, and text, a powerline controller for environmental control, and a telephone dialing/answering device. We are attempting to select components which are widely distributed and serviced as well as being plug compatible and economically priced.

Programs will be written or purchased to perform specific functions in each of the four general areas mentioned above. However, we would be very interested in receiving ideas from your readers, particularly those who are disabled, those who have disabled friends or relatives, and those who have personal computers and would like to develop hardware or software for the system on their own, regarding specific functions which they would like to see developed and which could be accommodated by the proposed micro-computer system.

Sincerely,
Charles Healey
Research Associate

Spain Rehabilitation Center
U.A.B. University Station
Birmingham, AL 35294
(205) 934-3320

COMMENTS ON KIM CASSETTE PROGRAM

Dear Mr. Warren: *Received: 78 Aug 14*

Now that *Dr. Dobb's* is slowly beginning to publish more 6502-oriented material, it's left the never-never land of "good computer magazines" to become a publication that I can actually use . . . and perhaps even take issue with from time to time. The basic motive for this letter is to comment on the "High Speed KIM Cassette I/O" in *DDJ* #26. While the high-speed facility is useful (its usefulness being, roughly, in direct proportion to the KIM users' exasperation with the normal monitor program's rate of cassette loading and dumping), the approach in #26 is the third version of a high-speed binary load/dump to be published in the past year. To my mind, the best of the three is Robert Tripp's "Ultratape," which is discussed in issue #1 of his 6502-oriented magazine, *MICRO*. "Ultratape"'s superiority derives from the fact that the program search capabilities of the KIM monitor aren't lost as they are in the other two versions; the tape header and the first byte of tape data are in a high-speed ASCII format (Jim Butterfield's "Hypertape" format—which is the normal KIM format, and monitor-compatible, at 6x normal KIM rate) before dropping automatically into binary format for the rest of the dump. Not noticeably slower than the version you published; but for those of us who need the tape search provision, a superior program. As well, the program in #26 uses parallel port bits PB2/PB3 to control cassette deck function. While author Bridge has kept this compatible with the original control subroutines in MICRO-ADE, similar cassette control routines are used in Tripp's "HELP" utility file programs for the KIM—but toggling PB0/PB1 instead. Since changing a constant or two in a program is a lot easier than rewiring the parallel port (although the circuitry change is fairly trivial), in the interest of compatibility the constants in Bridge's tape control subroutines (and in MICRO-ADE) could easily be changed to toggle bits 0 and 1 instead of bits 2 and 3.

While I've got your ear, I should pass on to you the names of three distribution points for computing equipment and insight whom I've found invariably reliable and fast:

1. Forethought Products, POB 386, Cobourg, OR 97401; the manufacturers of the KIMSI KIM-S100 interface/motherboard, they produce a quality product with excellent documentation and service, and have responded to my many queries and comments quickly and thoroughly;
2. Arisia Microsystems, 1455 Gregwood Rd., Mississauga, Ont. L5H 2T5 and
3. Orthon Computers, 12411 Stony Plain Rd., Edmonton, Alta. T5N 3N3 are two Canadian dealers (primarily of S-100 equipment) whose prices are fair (in most cases better than I would do importing myself and paying duties/post/exchange) and service fast and reliable.

Peace.
Barry Tepperman, M.D.

25 St. Mary Street #411
Toronto, Ontario M4Y 1R2 Canada

DC HAYES RESPONSE TO MODEM COMPARISON

Dear Sirs: *Received: 78 July 18*

In Volume 3 Issue 6, Jun-Jul 1978, you printed an article by Tim Pugh entitled "Intelligent Terminal Implementation for the S-100 Bus." I enjoyed reading the article. It was informative and well-written and the program looks good

though I haven't tried it yet. Unfortunately, I am writing you because the article contains several inaccurate statements about our S-100 modem, the D.C. Hayes 80-103A Data Communications Adapter.

Mr. Pugh points out that we chose to use the simpler MC 14412 modem chip instead of the MC6860, which provides several automatic functions. This consists primarily of some simple answering logic. We found it to be less expensive and much more flexible to implement these functions with a separate I/O port and some software (less than 10 instructions in 8080 code), which can give the user such capabilities as answering on the Nth ring, etc. For example, one of our users has programmed his machine to listen for the phone to ring. If the caller rings once and then hangs up, it answers the next call. Otherwise it lets the phone ring. This makes it unlikely that someone who doesn't know about modems will get a squeal in their ear when they try to call you. The 6860 always answers as soon as the phone starts ringing (well 51 milliseconds after). This will work even if the CPU is not running. This means it will answer and accept a long-distance call even if the system has crashed.

The design criteria for our modem was to get the most performance and flexibility possible for the least money. As Mr. Pugh notes, our modem is about $100 less expensive than the IDS board.

The filter specifications in the Motorola application note are quite conservative. We do publish a recieve sensitivity figure (–37dBm), which is sufficient for reliable operation on about 90% of all long-distance calls. Since the digitally-created signals from the MC14412 chip contain very little energy in the harmonics below the 13th and 15th (which occur in the 25-40 kHz band), all that is need on the transmit signal is a simple low-pass filter to keep these harmonics below –40dBm. As Mr. Pugh points out, filters are pretty complicated and require a lot of expensive parts. Since we wanted to keep the price low, we felt that the additional parts to get 98% were not justified. We are sorry that Mr. Pugh did not try our modem in a full-duplex configuration, because it does work. On page 44 of the same issue, there is a letter from Ward Christensen describing a system using our modem. It normally operates full-duplex, and has been successfully used by dozens of people from all over the country.

We only provided two baud rates (110 and 300) because those are the only standard rates for ASCII. Our manual does provide detailed information on rewiring the baud rate generator for non-standard baud rates, however.

We didn't provide a dial tone detector because a simple 2-second delay is adequate. Telephone company design rules virtually guarantee that properly functioning telephone exchanges will provide dial tone within that time.

The auto-dialing capability of the IDS modem merely creates the pulses, which is *not* the most difficult part of the dialing software. It still requires software to get the phone number, and to feed the digits to the dialer. I have rewritten Mr. Pugh's subroutine DIALIT (which dials a single digit) to work with the 80-103A. The original consisted of 11 instructions occupying 28 bytes, the modified version contains 22 instructions occupying 49 bytes. Since I don't believe any program works until I see it with my own eyes and I haven't run this one yet, I can't guarantee that it works, but it is pretty simple so I doubt that I made very many mistakes in it.

We did not provide power for the coupler because the standard S-100 bus voltages are not high enough to meet the telephone company's specs for the coupler (20-26 vdc), and the telephone comapny specifically requests non-switched power. Telephone company power is usually about the cheapest option available for a coupler. And if it's not, a 24vdc power supply is not very expensive.

The 80-103A is designed to use a DAA, and our manual is quite clear that although it will work when directly connected (it *does* have a ring detector), it is not legal to do so in most places.

```
DIALIT: ANI     0FH     ;ISOLATE DIGIT
        JNZ     D1      ;CHECK FOR 0
        MVI     A,10    ;AND CHANGE TO 10
D1:     PUSH    PSW     ;SAVE THE COUNT OF PULSES
        OUT     MODEM2  ;A DIAL PULSE IS CREATED BY OUTPUTTING
                        ;A VALUE CONTAINING 0 IN BIT 7 TO THE
                        ;MODEM CONTROL REGISTER
        MVI     B,2     ;STANDARD (10 PPS) DIAL PULSES LAST
        CALL    DELAY   ;ABOUT 60 MILLISECONDS
        MVI     A,80H   ;END THE PULSE BY SETTING BIT 7 BACK
        OUT     MODEM2  ;TO A 1 (OFF HOOK CONDITION)
        MVI     B,1     ;DELAY ABOUT 30 MILLISECONDS
        CALL    DELAY   ;BETWEEN PULSES
        CALL    CSTAT   ;NEXT 3 INSTRUCTIONS UNMODIFIED
        JZ      D2
        CALL    CR$ST
D2:     POP     PSW     ;GET THE COUNT BACK
        DCR     A       ;DECREMENT IT
        JNZ     D1      ;KEEP MAKING PULSES TIL IT GOES TO 0
        MVI     B,15    ;INSERT A .5 SEC DELAY BETWEEN DIGITS
        CALL    DELAY
        MVI     A,'-'   ;UNMODIFIED FROM HERE ON
        CALL    TCHAR
        RET
```

We recommend renting the DAA from the phone company. The DAA is a pretty silly gizmo and really doesn't do much other than introduce another couple dB loss into the circuit. But when you rent the DAA, you get treated as a data customer. Most data customers are big companies using big computers. They demand the best service and get it. With the DAA, you are paying for that service and you can demand it too. Telephone company design rules for local loops (the pair of wires from your house to the central office) are more stringent for lines with DAA's, and when they install your DAA, they take more time to make sure you get a good loop. This is important because local loops lose more data than long-distance lines. With a bad long-distance line, at least you can dial again, but if your loop is noisy, you're out of luck.

We don't sell kits because we believe that they don't make sense economically. We found that the actual difference in cost to us between a kit and an assembled board is only about $10. Rather than create an artificial price difference (and a lot of headaches for us and our customers), we decided to assemble them all in our factory. All of our boards are tested, burned in for 48 hours or more, and tested again to assure that they meet *all* of the published specs.

I am presently modifying Mr. Pugh's program to run with our modem and intend to submit it to the CP/M user's group as soon as I get it working.

If you would like information about our modem, call or write us. We make a real effort to answer our user's questions and to help them make their systems work.

Respectfully,
Donald J. Hyde
Director of Software Development

D.C. Hayes Associates
P.O. Box 9884
Atlanta, GA 30319

MORE ON FORTH IMPLEMENTATIONS

Gentlemen: *Received: 78 Aug 18*

I read John James' "Forth for Microcomputers" in Volume 3, Issue 5 with great interest. I have been involved with Forth and Forth-like languages since late 1975, and am one of the promoters of it at the University of Rochester. To clarify your article's last page, the University of Arizona, (primarily the Kitt Peak Branch of AURA, Association of Universities doing Research in Astronomy) is running single user versions of Forth on CDC 6400, HP 2100 and Varian 620I.

At the University we are running a multi-tasking Forth called URTH on the following machines: IBM 360/65, INTEL 8080, MODCOMP II, HP 2114, HP 2100, PDP 11/34, and soon the LSI-11. These are used for applications as diverse as laser control, fusion diagnostics, data entry, telecommunications and image processing.

We also have a manual, and will be setting up software and manual distribution shortly. Anyone interested should write me at: The Laboratory for Laser Energetics, 250 East River Road, Rochester, NY 14623.

A technical Forth/URTH internals manual is being prepared from a series of talks and workshops given at the University during the past year.

In reference to your comment about micro-coding the Forth interpreter, it has been done for an HP 21MX-M computer, and was reported in the European Southern Observatory technical note LS 76-119 by S. Lorenson dated 10/29/76.

Good luck to all who are interested in Forth.

Sincerely yours,
Lawrence P. Forsley

University of Rochester
Laboratory for Laser Energetics
Rochester, NY 14623

SOFTWARE [IDEAS] FOR 50¢

Gentlemen: *Received: 78 July 12*

Perhaps your readers would be interested in another software source – the U.S. Patent Office. Statutory disclosure requirements are such that systems patent applications should include software. A patent recently issued to Hewlett-Packard (4,089,059) contains software for a 6800 based system. While not directly usable because of hardware differences, the source code may be of interest to those with 6800 based systems. Patents may be obtained for fifty cents each from

Box 9
Patent &Trademark Office
Washington, D.C. 20231

Anyone interested should be prepared to spend a fair amount of time studying: in the patent mentioned above there are 226 drawing figures, 20 pages of text and 150 pages of source code – the patent is 1¼ inches thick.

Very truly yours,
Paul F. Wille

165 Hurd Road
Aurora, OH 44202

Just remember that the software mentioned is also *protected by the patent in which you read it. The purpose of patent disclosure is, of course, to inspire further invention but care should be taken to avoid infringement.* *– TRW*

CLEARING 6800 MEMORY

Dear Dr. Dobb's, *Received: 78 July 28*

Your September 1976 issue contained a letter on page 32 from Richard Kaapke which described, among other things, a nine byte 8080 program for clearing memory. Last night I decided to try something similar for the 6800, and devised the following.

Routine 1 (8 bytes): fill memory with NOPs. Properly assembled, this routine will execute anywhere in memory.

```
2000 8E 20 03          LDS  #STACK
2003 86 01      STACK  LDAA #NOP
2005 36         PUSH   PSHA
2006 20 FD             BRA  PUSH
```

Routine 2 (5 bytes): fill memory with zeroes. This routine must reside at the addresses shown, and assumes that executing an opcode of zero is identical to executing an NOP.

```
FFFB 8E FF FD          LDS  #HERE-1
FFFE 8D FE      HERE   BSR  HERE
```

Routine 3 (8 bytes): fill memory with ones (all bits on). This routine must reside at the addresses shown.

```
FFF7 8E FF FA          LDS  #THERE
FFFA CE FF FF   THERE  LDX  #HERE+2
FFFD 8D FE      HERE   BSR  HERE
```

Assuming that every location in the address space acts like RAM (ie, can be read, written, and executed), then all three routines fill ALL of memory, obliterating themselves in the process.

Sincerely,
Mike Gabrielson

Box 2692
Stanford, Calif. 94305
(415) 493-5217

REPORT ON WELLES FLOPPY CONTROLLER

Dear Dr. Dobbs, *Received: 78 July 20*

I think you may be interested in one readers's experience with the Welles Floppy Disk controller and FDOS Software. After reading about this modern wonder in BYTE, I ordered a blank PC board from Dr. Welles. My order was confirmed after a week or so, and I had my board in about 5 weeks. This is not in any way a kit, you should know all about TTL logic (or be ready to learn). I replaced all the 74XX and 74LXX chips (except the 7438) with 74LSXX, so that the card now runs a lot cooler and some of the signal lines look a bit cleaner. There were a few errors on my card, but they were all covered in the documentation that came with it. After working with this system for around 10 weeks I can say that it really works, the FDOS software works well, though it lacks that "polished" look and may give you some odd results every once in a while. It has not yet dropped a file, however, believe it when it says I/O error. I feel that nothing else I've seen comes close to it for the cost. (A little under $600.) Shopping around lately I'd found everything from about $450, the key is finding a drive for $350 – it can be done.

Sincerely,
Al Marshall

408 Oakwood
Angola, IN 46703

DR. DOBB'S JOURNAL of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 30 | November-December 1978 | Volume 3, Issue 10

SOFTWARE

A LISP interpreter for the 8080 *Darrel Van Buer* 4
Documentation and complete source listing

Curve-Fitting Using a Focal Interpreter on the KIM-1 *Theodore E. Bridge* 12

Binary Tree Manipulation on the 8080 *Mike Gabrielson* 15
A tutorial treatment

Programming Languages and Standards *David Chapman* 18

SAM-76 Language Update *Ancelme Roichel* 19

BASEX: A Fast, Compact Interactive Compiler *Paul K. Warme* 26
Description of a new intermediate-level language

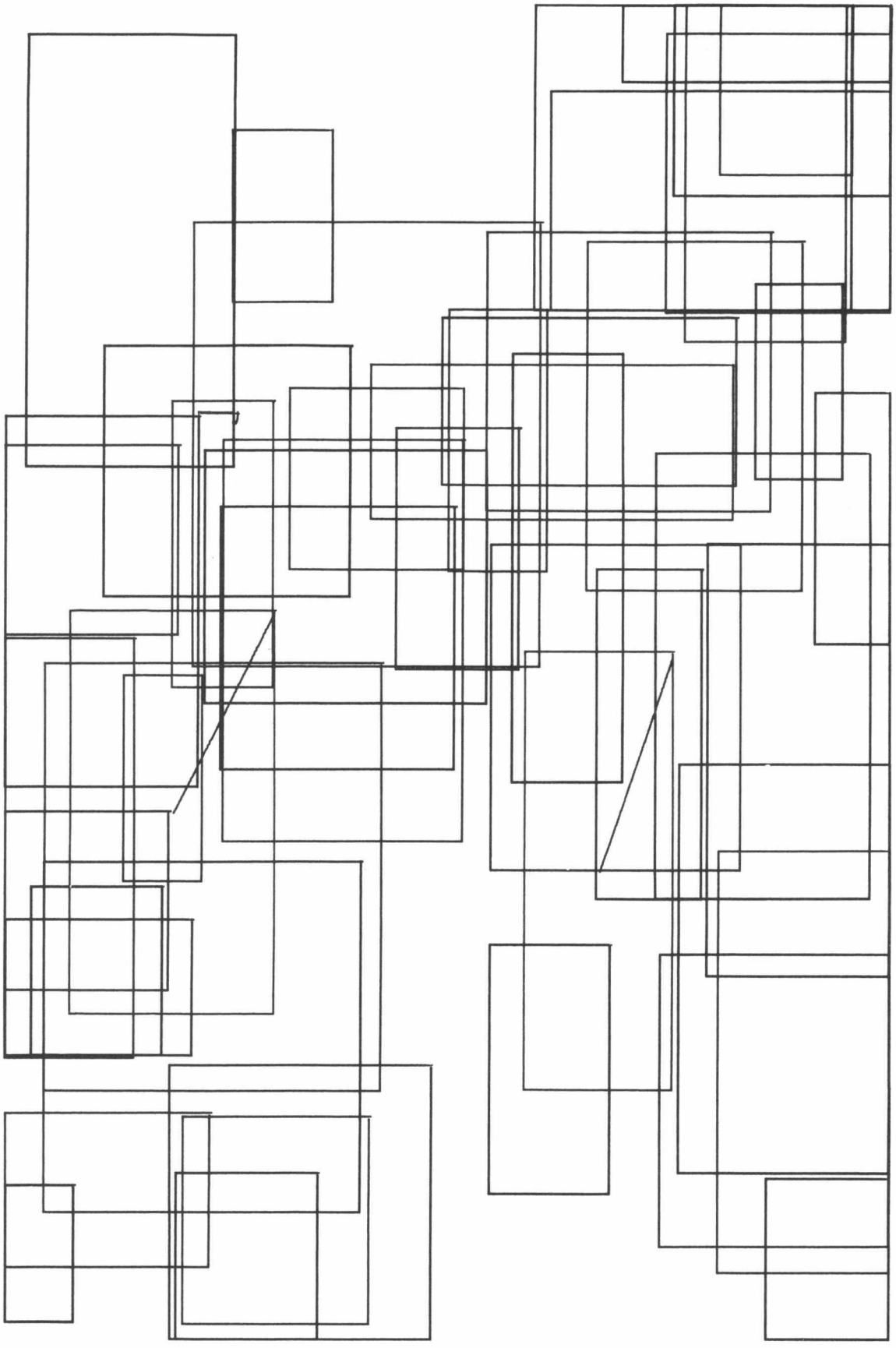
Tiny GRAFIX for Tiny BASIC *Joel Swank* 32
A subroutine package for Tom Pittman's Tiny BASIC

Copying a Diskette on Intel Double Density DOS *Paul M. Julich* 40

HARDWARE

The Heath H-8 System *Tom Williams* 22
Featuring Heath's new diskette system

The STD Bus *Keith Britton* 34
A critical look at a proposed new standard



© David Dameron 1979

A LISP Interpreter for the 8080

BY DARREL VAN BUER
1522 Brockton Ave. Apt. 5
Los Angeles, CA 90025

As assembled here, entry is at zero. Most of the code may be anywhere in memory but all LISP data elements (CONS cells and atoms) must be in the first 32K of memory. The interpreter is completely self contained except for terminal I/O. On input from the terminal, all control characters are ignored, and thus may be freely added to improve readability. All input is parsed as literal atoms and S-expressions. An atom is any string of alphanumeric characters not over 16 characters long. Some examples of legal atoms are:

ATOM 934 A47 47A

Some examples of illegal atoms are:

VERYLONGSTRINGOFLETTERS	(More than 16 characters)
AD-P	(Contains non alphanumeric –)

The only other characters recognized by the system are left and right parentheses, commas, periods and blanks. Consecutive atoms in lists and S-expressions must be separated by at least one of these symbols to be recognized. Blanks may be optionally inserted before left parens and after right parens.

Both input and output routines can handle mixed S-expressions (with dot notation) and list notation. The output routines will always use list notation whenever possible, but will use dotted notation when necessary. E.g., an S-expression input as: ((INTEL.NIL).8080) would print as ((INTEL).8080).

Logical values returned by EQ, ATOM, NULL and EQUAL are T and F. These must be quoted when used as constants in a program. These are also the logical values expected by COND (actually COND only tests for T).

The interpreter is an EVALQUOTE mode interpreter so two S-expressions must be entered for evaluation to occur. Both of these S-expressions must be followed by a blank to allow for the lookahead used in READ. After recognizing the second S-expression, which is a list of arguments to which the first S-expression (typically an atom) will be applied, both of these expressions are effectively quoted, so all atoms in the expressions stand for themselves, not any DEFINEd binding. Once the interpreter recognizes the second expression, it does a new line and types '>>' followed by the result of the computation.

For example, if the following is entered as input:

```
DEFINE(((T T) (F F) (FF(LAMBDA(X) (COND((ATOM X)X)
  (T(FF(CAR X))))))))
```

the interpreter will respond with:

```
>>(T F FF)
```

If you then enter:

```
FF (((A.B).C))
```

the interpreter will respond with:

```
>>A
```

The following functions are defined as built in subroutines:

ATOM[x] T if the value of X is a literal atom, F otherwise. NIL is an atom.

NULL[x] T if X has value NIL, F otherwise.

CONS[x;y] Returns a new S-expression whose CAR is X and CDR is Y.

CAR[x] Returns the CAR portion of a CONS cell. The result is undefined when applied to atoms.

CDR[x] Returns the CDR portion of a CONS cell. Undefined for atoms.

EQ[x;y] Returns T if X and Y are the same atom or the same CONS cell, F otherwise.

EQUAL[x;y] Returns T if X and Y are EQ or are congruent trees. Objects which are EQUAL will print identically although they may use different CONS cells, and thus not be EQ.

CADR[x] Same as CAR[CDR[x]]

CADDR[x] Same as CAR[CDR [CDR [x]]]

DEFINE[x] X must be a list of definitions. Each definition is a list whose first element is the name receiving the definition and whose second element is the definition. The value is a list of the names defined.

EVAL[e;a] E is an expression to be evaluated. A is the association list defining the environment for evaluation. The value is the result of this evaluation.

APPLY [fn;x;a] FN is a function name or functional form, X is a list of arguments for the function and A is the association list to be used. The result is the result of applying FN to the arguments in the specified environment.

ASSOC [x;a] X is an atom whose association is fetched from the association list A. The value is the first CONS cell from A of the form CONS [x;value].

PAIRLIS[x;y;a] X is a list of atoms, Y is a list of values, A is an association list. The result is a new association list consisting of the old association list preceded by new association cells constructed from pairwise joining of elements from the list X (for names) and Y (for values).

The following are special forms recognized by the interpreter:

COND[(t1 v1) (t1 v2) ...] COND is the conditional ("if") element of LISP. It takes an indefinite number of arguments, each of which is a two element list. COND successively evaluates the first element of each of these lists until one is found with the value T. The value of the COND is then the result of evaluating the second element of the same list. If none are T, the result is an error. (For this reason, the last pair is usually of the form ((QUOTE T) expr)).

LAMBDA[v;e] V is a list of variable parameter names. LAMBDA is the function designator of the LISP world. When a LAMBDA expression is applied to a list of arguments, the arguments are paired with the variable list V (via PAIRLIS) on the front of the current association list in EVAL/APPLY. Finally the value of the expression E is evaluated in this new environment and returned.

LABEL[l;e] LABEL is used to associate the expression E with the label L then evaluate E in this environment. The primary use of LABEL is the definition of temporary recursive functions (as opposed to permanent definition via DEFINE). For example, the example with DEFINE and FF earlier could have been done as:

```
(LABEL FF (LAMBDA (X) (COND ( (ATOM X) X) (
(QUOTE T) (FF (CAR X) ) ) ) ) ) ( ( (A.B) .C) ) >>A
```

Here, FF is only meaningful inside this expression, whereas after DEFINE, FF is a permanently known function.

Internal Organization of the Interpreter

All free memory in the first 32K is used as non-reclaimed storage for lists and atoms. Because there is presently no garbage collector, problems are limited to those with at most 8000 CONSes (in 32K).

For built in, assembled functions, the calling conventions are as follows: HL passes the first parameter and returns the result (except that functions which return only T or F such as ATOM return T by setting the Zero flag and F by non-zero flag). DE passes the second parameter and BC passes the third parameter. BC and DE must *always* be restored to their original values before returning. HL must also be restored for T/F functions using Z/NZ returns. For functions with more than three parameters, BC and DE are not used and HL passes a list of the parameters. Note that there are no functions with more than three parameters in the system at present, but APPLY has code to follow this convention. Assembled functions may call one another directly.

The interpreter has two permanent internal association lists. One is the list ASMFN (at 0637H in this listing) which is a list of all assembled functions. The value of these atoms are CONS cells in which CAR is the real memory address of the code and the CDR is a flag describing the function. This information is in the L byte. The high order bit (80H) is on if the function returns T/F as Z/NZ. The low order bits are the number of arguments. Zero is used if more than three arguments.

The other list is initially NIL, but is modified by DEFINE to include new definitions. This list is the initial association list used by EVALQUOTE when it calls EVAL. This list is stored in EVQAL (AT 04B4H).

The interpreter also maintains a tree used by READ to record all atoms known to the system so that only one copy of each atom is kept. The initial state of the tree is computed following START (at 04CDH) and stored in NAMETREE (at 042FH).

The interpreter is fully reinitialized following reentry at zero. To reenter the interpreter and keep old function definitions and known atoms, the interpreter should be entered at INLOOP (at 0599H).

The I/O routines required for the interpreter must pass the character in the accumulator with high order bit zero. Further, they should restore all registers before returning. Equates for the address of the pertinent routines are in the first few lines of code. If you patch the object code instead, OUTC is called in 5 places following OUTPUT (0050H) and two places in EVALQUOTE (028EH). EVALQUOTE also calls CRLF, a routine which should send CR, LF and whatever nulls needed in your system. The only call to INCH occurs in NEXTCHAR (0348H).

```
2G6 ASSM 0 0000

0000                    0001 OUTC   EQU  0F009H    START ADDRESS OF FREE MEMORY
0000                    0002 MEM    EQU  700H     #256 BYTE BLOCKS
0000                    0003 FREBLK EQU  64
0000                    0004 INCH   EQU  0F006H
0000                    0005 CRLF   EQU  0F021H
0000                    0006 ABEND  EQU  0F000H   MONITOR REENTRY
0000                    0007 *
0000                    0008 *
0000                    0009 *
0000 C3 B6 04           0010        JMP  START
0003 CD 10 00           0011 CADDR  CALL CDR
0006 CD 10 00           0012 CADR   CALL CDR
0009 F5                 0013 CAR    PUSH PSW      SAVE A,F
000A 7E                 0014 CAR2   MOV  A,M      GET CAR/CDR PTR
000B 23                 0015        INX  H
000C 66                 0016        MOV  H,M      HL:=CAR(HL)
000D 6F                 0017        MOV  L,A
000E F1                 0018        POP  PSW
000F C9                 0019        RET
0010 F5                 0020 CDR    PUSH PSW      SKIP CAR PTR
0011 23                 0021        INX  H        FOR HL:=CDR(HL)
0012 23                 0022        INX  H
0013 C3 0A 00           0023        JMP  CAR2
0016 D5                 0024 CONS   PUSH D        SAVE NEW CDR VALUE
0017 E5                 0025        PUSH H        SAVE NEW CAR VALUE
0018 11 04 00           0026        LXI  D,4      NEED 4 BYTES
001B CD AD 05           0027        CALL GTMEM    GET MEMORY
001E D1                 0028        POP  D        CAR VALUE
001F 73                 0029        MOV  M,E      INTO LINK
0020 23                 0030        INX  H
0021 72                 0031        MOV  M,D
0022 23                 0032        INX  H
0023 D1                 0033        POP  D        CDR VALUE
0024 73                 0034        MOV  M,E
```

```
0025 23
0026 72
0027 2B
0028 2B
0029 2B
002A C9
002B 7C
002C E6 80
002E FE 80
0030 C9
0031 7C
0032 BA
0033 C0
0034 7D
0035 BB
0036 C9
0037 E5
0038 CD 2B 00
003B CA 44 00
003E CD 10 00
0041 C3 38 00
0044 CD 49 00
0047 E1
0048 C9
0049 3E 80
004B BC
004C C0
004D AF
004E BD
004F C9
0050 E5
0051 CD 49 00
0054 C2 5D 00
0057 21 69 04
005A C3 63 00
005D CD 2B 00
0060 C2 75 00
0063 C5
0064 7C
0065 E6 7F
0067 67
0068 46
0069 23
006A 7E
006B CD 09 F0
006E 05
006F C2 69 00
0072 C1
0073 E1
0074 C9
0075 3E 28
0077 CD 09 F0
007A CD 37 00
007D C2 99 00
0080 E5
0081 CD 09 00
0084 CD 50 00
0087 E1
0088 CD 10 00
008B CD 49 00
008E CA AC 00
0091 3E 20
0093 CD 09 F0
0096 C3 80 00
0099 CD 09 00
009C CD 50 00
009F 3E 2E
00A1 CD 09 F0
00A4 E1
00A5 E5
00A6 CD 10 00
00A9 CD 50 00
00AC 3E 29
00AE CD 09 F0
00B1 E1

0035        INX  H
0036        MOV  M,D
0037        DCX  H          HL:=CONS(HL,DE)
0038        DCX  H
0039        DCX  H
0040        RET
0041 ATOM   MOV  A,H        ATOM(HL)
0042        ANI  80H        TEST HIGH ORDER BIT
0043        CPI  80H        ATOM IF BIT SET
0044        RET             Z-FLAG IS TRUE
0045 EQ     MOV  A,H        EQ(HL;DE)
0046        CMP  D          EQUAL IF SAME POINTER
0047        RNZ
0048        MOV  A,L
0049        CMP  E
0050        RET             Z-FLAG SET IF TRUE
0051 ISLIST PUSH H          SAVE.  ISLIST(HL)
0052 TEST   CALL ATOM       LOOK FOR ATOMIC
0053        JZ   EOL        END OF LIST
0054        CALL CDR        TRUE IF C(D**N)R=NIL
0055        JMP  TEST
0056 EOL    CALL NULL       NIL IS ATOM
0057        POP  H
0058        RET             Z-FLAG SET IF TRUE
0059 NULL   MVI  A,80H      NULL(HL) IF PTR IS 8000H
0060        CMP  H
0061        RNZ             NZ-FALSE
0062        XRA  A          A=0
0063        CMP  L
0064        RET
0065 OUTPUT PUSH H
0066        CALL NULL
0067        JNZ  NNL
0068        LXI  H,NILAT    PRINT 'NIL'
0069        JMP  PAT        GO PRINT ATOM
0070 NNL    CALL ATOM       OUTPUT AN ATOM?
0071        JNZ  NAT
0072 PAT    PUSH B          SAVE
0073        MOV  A,H
0074        ANI  7FH        CLEAR ATOM FLAG
0075        MOV  H,A
0076        MOV  B,M        ATOM STRING LENGTH
0077 ALP    INX  H
0078        MOV  A,M        GET CHAR
0079        CALL OUTC       PRINT CHAR
0080        DCR  B
0081        JNZ  ALP
0082        POP  B
0083        POP  H
0084        RET
0085 NAT    MVI  A,'('      BEGIN LIST SEP
0086        CALL OUTC
0087        CALL ISLIST     LIST OR SEXP?
0088        JNZ  SEXP
0089 NIS    PUSH H          SAVE CURRENT PTR
0090        CALL CAR
0091        CALL OUTPUT     RECURSIVE CALL FOR LEFT SUBT.
0092        POP  H          CURRENT
0093        CALL CDR        NEXT LIST ELEMENT
0094        CALL NULL       END OF LIST?
0095        JZ   CLOSE      YES, FINISH
0096        MVI  A,' '      NO, SEPARATOR
0097        CALL OUTC
0098        JMP  NIS
0099 SEXP   CALL CAR
0100        CALL OUTPUT     RECURSIVE
0101        MVI  A,'.'      SEXP SEPARATOR
0102        CALL OUTC
0103        POP  H          REVIVE PARM
0104        PUSH H          AND SAVE
0105        CALL CDR        RIGHT HALF
0106        CALL OUTPUT     RECURSIVE
0107 CLOSE  MVI  A,')'
0108        CALL OUTC
0109        POP  H

00B2 C9
00B3 CD 2B 00
00B6 EB
00B7 C2 C3 00
00BA CD 2B 00
00BD EB
00BE C0
00BF CD 31 00
00C2 C9
00C3 CD 2B 00
00C6 EB
00C7 CA E8 00
00CA E5
00CB D5
00CC CD 09 00
00CF EB
00D0 CD 09 00
00D3 CD B3 00
00D6 D1
00D7 E1
00D8 C0
00D9 E5
00DA D5
00DB CD 10 00
00DE EB
00DF CD 10 00
00E2 CD B3 00
00E5 D1
00E6 E1
00E7 C9
00E8 F6 01
00EA C9
00EB CD 49 00
00EE C2 F4 00
00F1 60
00F2 69
00F3 C9
00F4 D5
00F5 E5
00F6 D5
00F7 CD 09 00
00FA EB
00FB CD 09 00
00FE EB
00FF CD 16 00
0102 D1
0103 E3
0104 CD 10 00
0107 EB
0108 CD 10 00
010B EB
010C CD EB 00
010F D1
0110 EB
0111 CD 16 00
0114 D1
0115 C9
0116 D5
0117 D5
0118 EB
0119 CD 49 00
011C C2 22 01
011F E1
0120 D1
0121 C9
0122 CD 09 00
0125 CD 09 00
0128 CD B3 00
012B C2 34 01
012E E1
012F D1
0130 CD 09 00
0133 C9
0134 E1
0135 CD 10 00

0110        RET
0111 EQUAL  CALL ATOM       EQUAL(HL;DE)
0112        XCHG            Z-FLAG SET IF EQUAL
0113        JNZ  XNA
0114        CALL ATOM
0115        XCHG
0116        RNZ             NOT BOTH ATOMS, NONEQ
0117        CALL EQ         BOTH ATOMS, USE EQ
0118        RET
0119 XNA    CALL ATOM
0120        XCHG
0121        JZ   SETNZ      NOT BOTH ATOMS
0122        PUSH H          SAVE ROOTS
0123        PUSH D
0124        CALL CAR
0125        XCHG
0126        CALL CAR
0127        CALL EQUAL      RECURSIVE ON SUBTREES
0128        POP  D
0129        POP  H
0130        RNZ             BOTH SUBTREES MUST MATCH
0131        PUSH H
0132        PUSH D
0133        CALL CDR
0134        XCHG
0135        CALL CDR
0136        CALL EQUAL      RECURSIVE
0137        POP  D
0138        POP  H
0139        RET             WITH RESULTS OF LAST CALL
0140 SETNZ  ORI  1          SET NZ FLAG
0141        RET
0142 PAIRLIS CALL NULL PAIRLIS(HL;DE;BC)
0143        JNZ  FULC       PAIRLIS(X,Y,A)
0144        MOV  H,B
0145        MOV  L,C
0146        RET             X=NULL SO RETURN A
0147 FULC   PUSH D          SAVE Y
0148        PUSH H
0149        PUSH D
0150        CALL CAR        (X)
0151        XCHG
0152        CALL CAR        (Y)
0153        XCHG
0154        CALL CONS       (CAR(X),CAR(Y))
0155        POP  D          RESTORE Y
0156        XTHL            SAVE CONS, GET X
0157        CALL CDR        (X)
0158        XCHG
0159        CALL CDR        (Y)
0160        XCHG
0161        CALL PAIRLIS (CDR(X),CDR(Y),A) RECURSIVE
0162        POP  D          FROM CONS
0163        XCHG
0164        CALL CONS       (CONS(...),PAIRLIS(...))
0165        POP  D          RESTORE Y
0166        RET
0167 ASSOC  PUSH D          (HL,DE) OR (X,A)
0168 ASSO2  PUSH D          SAVE A
0169        XCHG
0170        CALL NULL
0171        JNZ  ASOGO
0172        POP  H          NULL ALIST, QUIT
0173        POP  D
0174        RET             RESULT=NIL
0175 ASOGO  CALL CAR        (A)
0176        CALL CAR        (CAR(A))
0177        CALL EQUAL      (CAR(...),X)
0178        JNZ  NOTYET
0179        POP  H          A FROM STACH
0180        POP  D
0181        CALL CAR        (A)
0182        RET             VALUE ASSOC WITH A
0183 NOTYET POP  H
0184        CALL CDR        DOWN ASSOC LIST X
```

```
0138 EB
0139 C3 17 01
013C CD 2B 00
013F C2 53 01
0142 CD 49 00
0145 C8
0146 CD 16 01
0149 CD 49 00
014C CA 00 F0
014F CD 10 00
0152 C9
0153 E5
0154 CD 09 00
0157 CD 2B 00
015A C2 60 01
015D D5
015E 11 61 84
0161 CD 31 00
0164 C2 6D 01
0167 D1
0168 E1
0169 CD 06 00
016C C9
016D 11 3E 84
0170 CD 31 00
0173 C2 7F 01
0176 D1
0177 E1
0178 CD 10 00
017B CD 49 02
017E C9
017F D1
0180 E3
0181 CD 10 00
0184 CD 75 02
0187 EB
0188 78
0189 44
018A 67
018B 79
018C 4D
018D 6F
018E E3
018F CD 96 01
0192 50
0193 59
0194 C1
0195 C9
0196 CD 2B 00
0199 C2 F9 01
019C D5
019D E5
019E 11 37 06
01A1 CD 16 01
01A4 CD 49 00
01A7 CA EF 01
01AA F1
01AB CD 10 00
01AE 54
01AF 5D
01B0 CD 10 00
01B3 EB
01B4 CD 09 00
01B7 7B
01B8 D1
01B9 C5
01BA D5
01BB 01 EC 01
01BE A7
01BF F2 C5 01
01C2 01 E3 01
01C5 C5
01C6 E5
01C7 62
01C8 6B
01C9 E6 07

0185            XCHG
0186            JMP   ASSO2     =CALL,RET
0187 EVAL       CALL  ATOM      (E) (HL,DE) IS (E,A)
0188            JNZ   NOTAT
0189            CALL  NULL
0190            RZ              VALUE(NIL)=NIL
0191            CALL  ASSOC     (E,A)
0192            CALL  NULL
0193            JZ    ABEND     ERROR, NOT DEFFINED
0194            CALL  CDR       (ASSOC(...))
0195            RET
0196 NOTAT      PUSH  H         SAVE E
0197            CALL  CAR       (E)
0198            CALL  ATOM      (CAR(E))
0199            JNZ   NOTAAT
0200            PUSH  D         SAVE A
0201            LXI   D,QUOTEAT+ATOMIC
0202            CALL  EQ        (CAR(E),'QUOTE')
0203            JNZ   NOTQ
0204            POP   D         A PROCESS 'QUOTE'
0205            POP   H         E
0206            CALL  CADR      (E)
0207            RET
0208 NOTQ       LXI   D,CONDAT+ATOMIC
0209            CALL  EQ        (CAR(E),'COND')
0210            JNZ   NOTCD
0211            POP   D         A-PROCESS 'COND'
0212            POP   H         E
0213            CALL  CDR       (E)
0214            CALL  EVCON     (CDR(E),A)
0215            RET
0216 NOTCD      POP   D         A
0217 NOTAAT     XTHL            SAVE CAR(E)
0218            CALL  CDR       (E)
0219            CALL  EVLIS     (CDR(E),A)
0220            XCHG
0221            MOV   A,B       SWAP BC,HL
0222            MOV   B,H
0223            MOV   H,A
0224            MOV   A,C
0225            MOV   C,L
0226            MOV   L,A
0227            XTHL            SAVE BC
0228            CALL  APPLY     (CAR(E),EVLIS(...),A)
0229            MOV   D,B       RESTORE A
0230            MOV   E,C
0231            POP   B
0232            RET
0233 APPLY      CALL  ATOM      (FN) (FN;X;A) IS (HL;DE;BC)
0234            JNZ   APNA
0235            PUSH  D         SAVE X
0236            PUSH  H         SAVE FN
0237            LXI   D,ASMFN   ASSOC LIST OF ASSEMBLED
0238            CALL  ASSOC     FUNCTIONS
0239            CALL  NULL      ON LIST?
0240            JZ    NOTDEF    NO, GO USE EVAL
0241            POP   PSW       DISCARD FN
0242            CALL  CDR       FUNCTION DATA
0243            MOV   D,H       DE=HL
0244            MOV   E,L
0245            CALL  CDR       GET FN FLAGS
0246            XCHG
0247            CALL  CAR       GET FN ADDR
0248            MOV   A,E       FN FLAGS
0249            POP   D         GET X
0250            PUSH  B         SAVE APPLY ARGS
0251            PUSH  D
0252            LXI   B,APRET   RETURN POINT
0253            ANA   A         TEST T/F VS LIST FN RETURN
0254            JP    WBLIST    RETURNS LISTS
0255            LXI   B,APTFR   RETURNS Z/NZ FOR T/F
0256 WBLIST     PUSH  B         SET FN RETURN POINT
0257            PUSH  H         SET FN CALL POINT
0258            MOV   H,D       HL=X
0259            MOV   L,E
0260            ANI   A,3       ISOLATE #ARGS

01CB C8
01CC FE 02
01CE CA DB 01
01D1 DA DF 01
01D4 CD 03 00
01D7 44
01D8 4D
01D9 62
01DA 6B
01DB CD 06 00
01DE EB
01DF CD 09 00
01E2 C9
01E3 21 67 84
01E6 CA EC 01
01E9 21 52 84
01EC D1
01ED C1
01EE C9
01EF E1
01F0 50
01F1 59
01F2 CD 3C 01
01F5 D1
01F6 C3 96 01
01F9
01F9 E5
01FA CD 09 00
01FD D5
01FE 11 5A 84
0201 CD 31 00
0204 C2 1A 02
0207 D1
0208 E1
0209 E5
020A CD 06 00
020D CD EB 00
0210 EB
0211 E3
0212 CD 03 00
0215 CD 3C 01
0218 D1
0219 C9
021A 11 54 84
021D CD 31 00
0220 C2 44 02
0223 D1
0224 E1
0225 E5
0226 CD 03 00
0229 E3
022A CD 06 00
022D EB
022E E3
022F EB
0230 CD 16 00
0233 D5
0234 50
0235 59
0236 CD 16 00
0239 44
023A 4D
023B E1
023C EB
023D E3
023E EB
023F CD 96 01
0242 C1
0243 C9
0244 3E FF
0246 CD 00 F0
0249 CD 49 00
024C CA 1A 02
024F E5
0250 CD 09 00
0253 CD 09 00

0261            RZ              NO ARGS, CALL FN WITH HL=X
0262            CPI   2         HOW MANY ARGS
0263            JZ    ARG2
0264            JC    ARG1      ONLY ONE
0265            CALL  CADDR     THREE ARGS, BC=3RD
0266            MOV   B,H
0267            MOV   C,L
0268            MOV   H,D       HL=X
0269            MOV   L,E
0270 ARG2       CALL  CADR
0271            XCHG            DE=2ND ARG
0272 ARG1       CALL  CAR       HL=1ST ARG
0273            RET             CALL FUNCTION
0274 APTFR      LXI   H,TAT+ATOMIC ASSUME TRUE
0275            JZ    APRET     RIGHT
0276            LXI   H,FAT+ATOMIC NO, FALSE
0277 APRET      POP   D         RESTORE REGS
0278            POP   B
0279            RET
0280 NOTDEF     POP   H         GET FN
0281            MOV   D,B       DE:=A
0282            MOV   E,C
0283            CALL  EVAL      (FN,A)
0284            POP   D         RESTORE X
0285            JMP   APPLY     (EVAL(...),X,A)
0286 * JMP = CALL, RET
0287 APNA       PUSH  H         SAVE FN
0288            CALL  CAR       (FN)
0289            PUSH  D         SAVE X
0290            LXI   D,LAMBDAAT+ATOMIC
0291            CALL  EQ        (CAR(FN),'LAMBDA')
0292            JNZ   NOTLAM
0293            POP   D         X
0294            POP   H         FN
0295            PUSH  H
0296            CALL  CADR      (FN)
0297            CALL  PAIRLIS   (CADR(FN),X,A)
0298            XCHG
0299            XTHL            SAVE X, GET FN
0300            CALL  CADDR     (FN)
0301            CALL  EVAL      (CADDR(FN)),PAIRLIS(...)
0302            POP   D
0303            RET
0304 NOTLAM     LXI   D,LABELAT+ATOMIC
0305            CALL  EQ        (CAR(FN),'LABEL')
0306            JNZ   NOTLB
0307            POP   D         X
0308            POP   H         FN
0309            PUSH  H
0310            CALL  CADDR     (FN)
0311            XTHL            GET FN, SAVE CADDR(FN)
0312            CALL  CADR      (FN)
0313            XCHG
0314            XTHL            X ONTO STACK
0315            XCHG
0316            CALL  CONS      (CADR(FN),CADDR(FN))
0317            PUSH  D         SAVE CADDR(FN)
0318            MOV   D,B       DE:=A
0319            MOV   E,C
0320            CALL  CONS      (CONS(...),A)
0321            MOV   B,H       BC:=CONS(...)
0322            MOV   C,L
0323            POP   H         CADDR(FN)
0324            XCHG
0325            XTHL            SAVE A, GET X
0326            XCHG
0327            CALL  APPLY     (CADDR(FN),X,CONS(...))
0328            POP   B         A RESTORED
0329            RET
0330 NOTLB      MVI   A,0FFH
0331            CALL  ABEND     ERROR IN LISP PROGRAM
0332 EVCON      CALL  NULL      (C,A) IS (HL,DE)
0333            JZ    NOTLAB    ERROR IF C=NULL
0334            PUSH  H         SAVE C
0335            CALL  CAR       (C)
0336            CALL  CAR       (CAR(C))
```

```
0256 CD 3C 01          0337          CALL EVAL     (CAAR(C),A)
0259 D5                0338          PUSH D        SAVE A
025A 11 67 84          0339          LXI  D,TAT+ATOMIC
025D CD B3 00          0340          CALL EQUAL    (EVAL(...),'T')
0260 D1                0341          POP  D        A
0261 E1                0342          POP  H        C
0262 C2 6F 02          0343          JNZ  CALEC
0265 CD 09 00          0344          CALL CAR      (C)
0268 CD 06 00          0345          CALL CADR     (CAR(C))
026B CD 3C 01          0346          CALL EVAL     (CADAR(C),A)
026E C9                0347          RET
026F CD 10 00          0348 CALEC    CALL CDR      (C)
0272 C3 49 02          0349          JMP  EVCON    (CDR(C)),A) -(CALL;RET)
0275 CD 49 00          0350 EVLIS    CALL NULL     (M,A) IS (HL,DE)
0278 C8                0351          RZ            EVLIS(NULL;A)=NULL
0279 E5                0352          PUSH H        SAVE M
027A CD 09 00          0353          CALL CAR      (M)
027D CD 3C 01          0354          CALL EVAL     (CAR(M))
0280 E3                0355          XTHL          SAVE EVAL, GET M
0281 CD 10 00          0356          CALL CDR      (M)
0284 CD 75 02          0357          CALL EVLIS    (CDR(M)) RECURSIVE
0287 EB                0358          XCHG
0288 E3                0359          XTHL          SAVE A, GET EVAL
0289 CD 16 00          0360          CALL CONS     (EVAL(CAR(M)),EVLIS(CDR(M)))
028C D1                0361          POP  D
028D C9                0362          RET
028E E5                0363 EVALQUOTE PUSH H
028F 2A B4 04          0364          LHLD EVQAL
0292 44                0365          MOV  B,H
0293 4D                0366          MOV  C,L
0294 E1                0367          POP  H
0295 CD 96 01          0368          CALL APPLY    (FN=HL,X=DE,NULL)
0298 CD 21 F0          0369          CALL CRLF
029B 3E 3E             0370          MVI  A,'>'
029D CD 09 F0          0371          CALL OUTC
02A0 CD 09 F0          0372          CALL OUTC
02A3 CD 50 00          0373          CALL OUTPUT
02A6 C9                0374          RET
02A7 D5                0375 MAKEATOM PUSH D
02A8 C5                0376          PUSH B
02A9 F5                0377          PUSH PSW
02AA EB                0378          XCHG          D->INPUT STRING
02AB 21 69 04          0379          LXI  H,NILAT  'NIL'
02AE CD 25 03          0380          CALL ACOMP
02B1 C2 BB 02          0381          JNZ  TREES    STRING NOT 'NIL'
02B4 21 00 80          0382          LXI  H,8000H  NIL POINTER
02B7 F1                0383 MAKEOUT  POP  PSW
02B8 C1                0384          POP  B
02B9 D1                0385          POP  D
02BA C9                0386          RET
02BB 2A 2F 04          0387 TREES    LHLD NAMETREE
02BE CD C4 02          0388          CALL AN       SEARCH KNOWN ATOM TREE
02C1 C3 B7 02          0389          JMP  MAKEOUT
02C4 44                0390 AN       MOV  B,H      BC->TREE (SAVE)
02C5 4D                0391          MOV  C,L
02C6 CD 09 00          0392          CALL CAR      TO ATOM
02C9 CD 25 03          0393          CALL ACOMP
02CC C8                0394          RZ            FOUND IDENTICAL ATOM
02CD 60                0395          MOV  H,B      RESTORE HL
02CE 69                0396          MOV  L,C
02CF CD 10 00          0397          CALL CDR
02D2 F5                0398          PUSH PSW      SAVE FLAGS
02D3 CD 49 00          0399          CALL NULL     END OF TREE?
02D6 C2 E7 02          0400          JNZ  ISLR     JUMP IF MORE
02D9 D5                0401          PUSH D        SAVE
02DA 54                0402          MOV  D,H      DE:=NIL
02DB 5D                0403          MOV  E,L
02DC CD 16 00          0404          CALL CONS     (NIL;NIL) ADDING TO TREE
02DF D1                0405          POP  D
02E0 03                0406          INX  B        INSERT INTO EXISTING TREE
02E1 03                0407          INX  B
02E2 7D                0408          MOV  A,L      CHANGE LINK
02E3 02                0409          STAX B
02E4 03                0410          INX  B
02E5 7C                0411          MOV  A,H
02E6 02                0412          STAX B
02E7 F1                0413 ISLR     POP  PSW      FLAGS
02E8 DA ED 02          0414          JC   ISLEFT   JUMP IF NEW LESS
02EB 23                0415          INX  H        GO RIGHT, INCR TO
02EC 23                0416          INX  H        OTHER PTR
02ED 44                0417 ISLEFT   MOV  B,H      SAVE NEW TREE ROOT
02EE 4D                0418          MOV  C,L
02EF CD 09 00          0419          CALL CAR      FOLLOW HL
02F2 CD 49 00          0420          CALL NULL
02F5 C2 C4 02          0421          JNZ  AN       IF NOT NULL(=CNZ,RNZ)
02F8 D5                0422          PUSH D
02F9 1A                0423          LDAX D        GET NAME LENGTH
02FA C6 04             0424          ADI  4        1 FOR LENGTH FIELD,3 TO CEIL
02FC E6 FC             0425          ANI  0FCH     DOWN TO 4*N
02FE 5F                0426          MOV  E,A
02FF 16 00             0427          MVI  D,0      DE=BYTES NEEDED
0301 CD AD 05          0428          CALL GTMEM
0304 11 00 80          0429          LXI  D,8000H  NIL
0307 E5                0430          PUSH H        SAVE NEW ATOM
0308 19                0431          DAD  D        SET ATOMIC
0309 CD 16 00          0432          CALL CONS     (NEW ATOM,NIL)
030C 7D                0433          MOV  A,L
030D 02                0434          STAX B        CONNECT TO CURRENT TREE
030E 03                0435          INX  B
030F 7C                0436          MOV  A,H
0310 02                0437          STAX B
0311 E1                0438          POP  H
0312 D1                0439          POP  D
0313 E5                0440          PUSH H        SAVE FOR RETURN
0314 1A                0441          LDAX D
0315 47                0442          MOV  B,A      LENGTH COUNT
0316 77                0443          MOV  M,A
0317 23                0444 COPYAT   INX  H        COPY ATOM TEXT TO ALLOC AREA
0318 13                0445          INX  D
0319 1A                0446          LDAX D
031A 77                0447          MOV  M,A
031B 05                0448          DCR  B
031C C2 17 03          0449          JNZ  COPYAT
031F E1                0450          POP  H
0320 7C                0451          MOV  A,H
0321 F6 80             0452          ORI  80H      SET ATOMIC FLAG
0323 67                0453          MOV  H,A
0324 C9                0454          RET
0325 E5                0455 ACOMP    PUSH H        SAVE REGS
0326 D5                0456          PUSH D
0327 C5                0457          PUSH B
0328 7C                0458          MOV  A,H
0329 E6 7F             0459          ANI  7FH      NO ATOM FLAG
032B 67                0460          MOV  H,A
032C 1A                0461          LDAX D        COUNT 2ND
032D 4F                0462          MOV  C,A
032E 46                0463          MOV  B,M      COUNT OF 1ST
032F 23                0464 COMPLP   INX  H
0330 13                0465          INX  D
0331 1A                0466          LDAX D
0332 96                0467          SUB  M        COMPARE CHARS
0333 C2 40 03          0468          JNZ  NEQL
0336 05                0469          DCR  B
0337 CA 44 03          0470          JZ   DONE1
033A 0D                0471          DCR  C
033B C2 2F 03          0472          JNZ  COMPLP
033E B0                0473          ORA  B        SET NZ
033F 37                0474          STC           HL LONGER, SET '>'
0340 C1                0475 NEQL     POP  B
0341 D1                0476          POP  D
0342 E1                0477          POP  H        RESTORE
0343 C9                0478          RET
0344 0D                0479 DONE1    DCR  C
0345 C3 40 03          0480          JMP  NEQL     SET FLAG BY RESIDUAL
0348 CD 06 F0          0481 NEXTCHAR CALL INCH
034B FE 20             0482          CPI  ' '
034D DA 48 03          0483          JC   NEXTCHAR IGNORE CONTROL CHAR
0350 32 9C 04          0484          STA  CURCH
0353 C9                0485          RET
0354 CD C4 03          0486 INPUT    CALL TAKELP   Z=TRUE GET '(' IF AVAIL
```

```
0357 C2 9C 03           0487          JNZ  S139     JUMP IF NOT THERE
035A CD D7 03           0488          CALL TAKERP   GET ')' IF THERE
035D C2 64 03           0489          JNZ  S127     JUMP IF NOT THERE
0360 21 00 80           0490          LXI  H,8000H NIL IS ()
0363 C9                 0491          RET
0364 D5                 0492 S127     PUSH D
0365 CD 54 03           0493          CALL INPUT    RECURSIVE CALL
0368 EB                 0494          XCHG          SAVE 'INPUT'
0369 CD CF 03           0495          CALL TAKEDOT TRY FOR '.' (SEXP)
036C C2 75 03           0496          JNZ  S130     JUMP IF LIST NOTATION
036F CD 54 03           0497          CALL INPUT    GET CDR SIDE
0372 C3 90 03           0498          JMP  S135     GO GET ')'
0375 CD ED 03           0499 S130     CALL TAKEBL   LIST:GET SEP ' ' OR ','
0378 C2 81 03           0500          JNZ  S132     NO SEPARATOR
037B CD A9 03           0501          CALL INPTL    GET REST OF LIST
037E C3 90 03           0502          JMP  S135     THEN ')'
0381 CD DF 03           0503 S132     CALL CTRP     SEE IF ')', BUT DONT TAKE
0384 C2 8D 03           0504          JNZ  S134
0387 21 00 80           0505          LXI  H,8000H NIL
038A C3 90 03           0506          JMP  S135
038D CD A9 03           0507 S134     CALL INPTL
0390 CD D7 03           0508 S135     CALL TAKERP   TAKE ')'
0393                    0509 BADIN    EQU  NOTLB
0393 C2 44 02           0510          JNZ  BADIN    ILLEGAL STRING
0396 EB                 0511          XCHG
0397 CD 16 00           0512          CALL CONS
039A D1                 0513          POP  D
039B C9                 0514          RET
039C CD E5 03           0515 S139     CALL CTATOM
039F C2 44 02           0516          JNZ  BADIN
03A2 CD 11 04           0517          CALL TAKEATOM PUT ATOM TEXT IN ATTXT
03A5 CD A7 02           0518          CALL MAKEATOM GET POINTER FOR IT
03A8 C9                 0519          RET
03A9 CD DF 03           0520 INPTL    CALL CTRP
03AC C2 B3 03           0521          JNZ  S144
03AF 21 00 80           0522          LXI  H,8000H NIL
03B2 C9                 0523          RET
03B3 D5                 0524 S144     PUSH D
03B4 CD 54 03           0525          CALL INPUT
03B7 EB                 0526          XCHG
03B8 CD ED 03           0527          CALL TAKEBL
03BB CD A9 03           0528          CALL INPTL
03BE EB                 0529          XCHG
03BF CD 16 00           0530          CALL CONS
03C2 D1                 0531          POP  D
03C3 C9                 0532          RET
03C4 3A 9C 04           0533 TAKELP   LDA  CURCH
03C7 FE 28              0534          CPI  '('
03C9 C0                 0535 TAKANY   RNZ
03CA CD 48 0?           0536          CALL NEXTCHAR
03CD AF                 0537          XRA  A        SET Z-FLAG
03CE C9                 0538          RET
03CF 3A 9C 04           0539 TAKEDOT  LDA  CURCH
03D2 FE 2E              0540          CPI  '.'
03D4 C3 C9 03           0541          JMP  TAKANY
03D7 3A 9C 04           0542 TAKERP   LDA  CURCH
03DA FE 29              0543          CPI  ')'
03DC C3 C9 03           0544          JMP  TAKANY
03DF 3A 9C 04           0545 CTRP     LDA  CURCH
03E2 FE 29              0546          CPI  ')'
03E4 C9                 0547          RET
03E5 3A 9C 04           0548 CTATOM   LDA  CURCH
03E8 FE 30              0549          CPI  '0'
03EA D8                 0550          RC            (NZ SET FOR NOT ATOM)
03EB AF                 0551          XRA  A        SET Z-FLAG
03EC C9                 0552          RET
03ED 3A 9C 04           0553 TAKEBL   LDA  CURCH
03F0 FE 20              0554          CPI  ' '
03F2 CA FC 03           0555          JZ   TAKING
03F5 DA FC 03           0556          JC   TAKING   CTRL OK
03F8 AF                 0557          XRA  A        SET Z-FLAG
03F9 FE 2C              0558          CPI  ','
03FB C0                 0559          RNZ
03FC CD 48 03           0560 TAKING   CALL NEXTCHAR
03FF 3A 9C 04           0561          LDA  CURCH
0402 FE 20              0562          CPI  ' '
0404 CA FC 03           0563          JZ   TAKING
0407 DA FC 03           0564          JC   TAKING
040A FE 2C              0565          CPI  ','
040C CA FC 03           0566          JZ   TAKING
040F AF                 0567          XRA  A        SET Z-FLAG (GOT BLANKS)
0410 C9                 0568          RET
0411 C5                 0569 TAKEATOM PUSH B SAVE
0412 06 00              0570          MVI  B,0      INITIAL ATOM LENGTH
0414 21 9E 04           0571          LXI  H,ATTXT
0417 CD E5 03           0572 TATL     CALL CTATOM
041A C2 29 04           0573          JNZ  FINISHT
041D 3A 9C 04           0574          LDA  CURCH
0420 77                 0575          MOV  M,A
0421 23                 0576          INX  H
0422 04                 0577          INR  B        CHAR COUNT
0423 CD 48 03           0578          CALL NEXTCHAR
0426 C3 17 04           0579          JMP  TATL
0429 21 9D 04           0580 FINISHT LXI H,ATL
042C 70                 0581          MOV  M,B      SET LENGTH
042D C1                 0582          POP  B
042E C9                 0583          RET
042F                    0584 ATOMIC EQU  8000H    ATOM PTR FLAG
042F 00 00              0585 NAMETREE DW 0
0431 04                 0586 ATOMAT DB   4
0432 41 54              0587          DW   'TA'     'ATOM'
0434 4F 4D              0588          DW   'MO'
0436 03                 0589 CARAT    DB   3
0437 43 41              0590          DW   'AC'     'CAR'
0439 52                 0591          DB   'R'
043A 03                 0592 CDRAT    DB   3
043B 43 44              0593          DW   'DC'     'CDR'
043D 52                 0594          DB   'R'
043E 04                 0595 CONDAT   DB   4
043F 43 4F              0596          DW   'OC'     'COND'
0441 4E 44              0597          DW   'DN'
0443 04                 0598 CONSAT   DB   4
0444 43 4F              0599          DW   'OC'     'CONS'
0446 4E 53              0600          DW   'SN'
0448 06                 0601 DEFINAT  DB 6  'DEFINE'
0449 44 45              0602          Dw   'ED'
044B 46 49              0603          Dw   'IF'
044D 4E 45              0604          Dw   'EN'
044F 02                 0605 EQAT     DB   2
0450 45 51              0606          DW   'QE'     'EQ'
0452 01                 0607 FAT      DB   1
0453 46                 0608          DB   'F'
0454 05                 0609 LABELAT  DB 5
0455 4C 41              0610          Dw   'AL'     'LABEL'
0457 42 45              0611          Dw   'EB'
0459 4C                 0612          DB   'L'
045A 06                 0613 LAMBDAAT DB 6
045B 4C 41              0614          DW   'AL'     'LAMBDA'
045D 4D 42              0615          DW   'BM'
045F 44 41              0616          DW   'AD'
0461 05                 0617 QUOTEAT  DB 5
0462 51 55              0618          Dw   'UQ'     'QUOTE'
0464 4F 54              0619          Dw   'TO'
0466 45                 0620          DB   'E'
0467 01                 0621 TAT      DB   1
0468 54                 0622          DB   'T'
0469 03                 0623 NILAT    DB   3
046A 4E 49              0624          DW   'IN'     'NIL'
046C 4C                 0625          DB   'L'
046D 05                 0626 CADDAT   DB   5
046E 43 41              0627          Dw   'AC'     'CADDR'
0470 44 44              0628          DW   'DD'
0472 52                 0629          DB   'R'
0473 04                 0630 CADRAT   DB   4
0474 43 41              0631          Dw   'AC'     'CADR'
0476 44 52              0632          DW   'RD'
0478 04                 0633 NULLAT   DB   4
0479 4E 55              0634          DW   'UN'     'NULL'
047B 4C 4C              0635          DW   'LL'
047D 05                 0636 EQULAT   DB   5
047E 45 51              0637          DW   'QE'     'EQUAL'
0480 55 41              0638          DW   'AU'
```

```
0482 4C
0483 07
0484 50 41
0486 49 52
0488 4C 49
048A 53
048B 05
048C 41 53
048E 53 4F
0490 43
0491 04
0492 45 56
0494 41 4C
0496 05
0497 41 50
0499 50 4C
049B 59
049C 00
049D 00
049E
04AE 00 00
04B0 00 00
04B2 00 00
04B4 00 00
04B6 21 00 07
04B9 22 AE 04
04BC 36 FF
04BE 23
04BF 36 FF
04C1 23
04C2 36 00
04C4 23
04C5 36 40
04C7 21 00 80
04CA 22 B4 04
04CD 21 67 84
04D0 11 00 80
04D3 CD 16 00
04D6 E5
04D7 21 83 84
04DA CD 16 00
04DD EB
04DE CD 16 00
04E1 EB
04E2 21 78 84
04E5 CD 16 00
04E8 D1
04E9 CD 16 00
04EC EB
04ED 21 61 84
04F0 CD 16 00
04F3 E5
04F4 21 54 84
04F7 11 00 80
04FA CD 16 00
04FD E5
04FE 21 91 84
0501 CD 16 00
0504 EB
0505 CD 16 00
0508 EB
0509 21 7D 84
050C CD 16 00
050F D1
0510 CD 16 00
0513 EB
0514 21 52 84
0517 CD 16 00
051A D1
051B CD 16 00
051E EB
051F 21 5A 84
0522 CD 16 00
0525 E5
0526 21 48 84
```

```
0639          DB    'L'
0640 PAIRAT   DB    7
0641          DW    'AP'      'PAIRLIS'
0642          DW    'RI'
0643          DW    'IL'
0644          DB    'S'
0645 ASOCAT   DB    5
0646          DW    'SA'      'ASSOC'
0647          DW    'OS'
0648          DB    'C'
0649 EVALAT   DB    4
0650          DW    'VE'      'EVAL'
0651          DW    'LA'
0652 APLYAT   DB    5
0653          DW    'PA'      'APPLY'
0654          DW    'LP'
0655          DB    'Y'
0656 CURCH    DB    0
0657 ATL      DB    0
0658 ATTXT    DS    16        MAX ATOM LENGTH
0659 CVTFQ    DW    0
0660 SPSAV    DW    0
0661 PREVFQE  DW    0
0662 EVQAL    DW    0
0663 START    LXI   H,MEM     START OF FREE MEMORY
0664          SHLD  CVTFQ
0665          MVI   M,0FFH    NO NEXT FREE BLOCK
0666          INX   H
0667          MVI   M,0FFH
0668          INX   H
0669          MVI   M,0       LENGTH OF FREE AREA
0670          INX   H
0671          MVI   M,FREBLK
0672          LXI   H,NIL
0673          SHLD  EVQAL
0674          LXI   H,TAT+ATOMIC
0675          LXI   D,NIL
0676          CALL  CONS      ('T',CONS(...))
0677          PUSH  H
0678          LXI   H,PAIRAT+ATOMIC
0679          CALL  CONS
0680          XCHG
0681          CALL  CONS
0682          XCHG
0683          LXI   H,NULLAT+ATOMIC
0684          CALL  CONS
0685          POP   D
0686          CALL  CONS
0687          XCHG
0688          LXI   H,QUOTEAT+ATOMIC
0689          CALL  CONS      ('QUOTE',NIL)
0690          PUSH  H
0691          LXI   H,LABELAT+ATOMIC
0692          LXI   D,NIL
0693          CALL  CONS      THESE LINES
0694          PUSH  H
0695          LXI   H,EVALAT+ATOMIC
0696          CALL  CONS
0697          XCHG            START THE
0698          CALL  CONS
0699          XCHG
0700          LXI   H,EQULAT+ATOMIC
0701          CALL  CONS
0702          POP   D
0703          CALL  CONS      TREE OF
0704          XCHG            KNOWN ATOMS
0705          LXI   H,FAT+ATOMIC
0706          CALL  CONS
0707          POP   D
0708          CALL  CONS
0709          XCHG
0710          LXI   H,LAMBDAAT+ATOMIC
0711          CALL  CONS
0712          PUSH  H
0713          LXI   H,DEFINAT+ATOMIC
```

```
0529 11 00 80
052C CD 16 00
052F E5
0530 21 3E 84
0533 CD 16 00
0536 EB
0537 CD 16 00
053A EB
053B 21 3A 84
053E CD 16 00
0541 D1
0542 CD 16 00
0545 EB
0546 21 43 84
0549 CD 16 00
054C E5
054D 21 6D 84
0550 11 00 80
0553 CD 16 00
0556 CD 16 00
0559 EB
055A 21 73 84
055D CD 16 00
0560 E5
0561 21 8B 84
0564 11 00 80
0567 CD 16 00
056A EB
056B CD 16 00
056E EB
056F 21 96 84
0572 CD 16 00
0575 D1
0576 CD 16 00
0579 EB
057A 21 31 84
057D CD 16 00
0580 D1
0581 CD 16 00
0584 EB
0585 21 36 84
0588 CD 16 00
058B D1
058C CD 16 00
058F EB
0590 21 4F 84
0593 CD 16 00
0596 22 2F 04
0599 CD 48 03
059C CD 54 03
059F CD ED 03
05A2 EB
05A3 CD 54 03
05A6 EB
05A7 CD 8E 02
05AA C3 99 05
05AD F5
05AE C5
05AF D5
05B0 21 00 00
05B3 39
05B4 22 B0 04
05B7 21 AE 04
05BA 22 B2 04
05BD F9
05BE E1
05BF 7C
05C0 A5
05C1 3C
05C2 CA FC 05
05C5 F9
05C6 C1
05C7 C1
05C8 78
05C9 92
```

```
0714          LXI   D,NIL
0715          CALL  CONS
0716          PUSH  H
0717          LXI   H,CONDAT+ATOMIC
0718          CALL  CONS
0719          XCHG
0720          CALL  CONS
0721          XCHG
0722          LXI   H,CDRAT+ATOMIC
0723          CALL  CONS
0724          POP   D
0725          CALL  CONS
0726          XCHG
0727          LXI   H,CONSAT+ATOMIC
0728          CALL  CONS
0729          PUSH  H
0730          LXI   H,CADDAT+ATOMIC
0731          LXI   D,NIL
0732          CALL  CONS
0733          CALL  CONS
0734          XCHG
0735          LXI   H,CADRAT+ATOMIC
0736          CALL  CONS
0737          PUSH  H
0738          LXI   H,ASOCAT+ATOMIC
0739          LXI   D,NIL
0740          CALL  CONS
0741          XCHG
0742          CALL  CONS
0743          XCHG
0744          LXI   H,APLYAT+ATOMIC
0745          CALL  CONS
0746          POP   D
0747          CALL  CONS
0748          XCHG
0749          LXI   H,ATOMAT+ATOMIC
0750          CALL  CONS
0751          POP   D
0752          CALL  CONS
0753          XCHG
0754          LXI   H,CARAT+ATOMIC
0755          CALL  CONS
0756          POP   D
0757          CALL  CONS
0758          XCHG
0759          LXI   H,EQAT+ATOMIC
0760          CALL  CONS
0761          SHLD  NAMETREE
0762 INLOOP   CALL  NEXTCHAR
0763          CALL  INPUT
0764          CALL  TAKEBL
0765          XCHG
0766          CALL  INPUT
0767          XCHG
0768          CALL  EVALQUOTE
0769          JMP   INLOOP
0770 GTMEM    PUSH  PSW       SAVE REGS
0771          PUSH  B
0772          PUSH  D         LENGTH REQUESTED
0773          LXI   H,0
0774          DAD   SP        SAVE PROGRAM SP
0775          SHLD  SPSAV
0776          LXI   H,CVTFQ TREE BASE LIKE 1ST FQE
0777 LP       SHLD  PREVFQE SAVE FOR EXACT CASE
0778          SPHL
0779          POP   H         NEXT FQE
0780          MOV   A,H       END OF CHAIN?
0781          ANA   L         (ALL FFFFH PTR)
0782          INR   A
0783          JZ    NOSPAC    NO MEMORY LEFT
0784          SPHL
0785          POP   B         SKIP LINK
0786          POP   B         FQE LENGTH
0787          MOV   A,B       FQE BIG ENOUGH?
0788          SUB   D         BC:=BC-DE
```

```
05CA CA D0 05            0789          JZ   HIEQ    JUMP IF LENGTHS CLOSE
05CD DA BA 05            0790          JC   LP      NO, TRY NEXT
05D0 47                  0791 HIEQ     MOV  B,A
05D1 79                  0792          MOV  A,C
05D2 C2 E9 05            0793          JNZ  BIGH    FQE MUCH BIGGER
05D5 93                  0794          SUB  E
05D6 CA F1 05            0795          JZ   EXACT   FQE SIZE=REQ SIZE
05D9 DA BA 05            0796          JC   LP      WON'T FIT
05DC 4F                  0797 RETPART  MOV  C,A BC NOW EXCESS
05DD C5                  0798          PUSH B       CHANGE FQE LENGTH
05DE 09                  0799          DAD  B       CALCULATE START OF RETURN AR
05DF EB                  0800 EXIT     XCHG         SAVE HL
05E0 2A B0 04            0801          LHLD SPSAV   RESTORE ORIGINAL SP
05E3 F9                  0802          SPHL
05E4 E1                  0803          POP  H       'DE' TO RESTORE
05E5 C1                  0804          POP  B
05E6 F1                  0805          POP  PSW
05E7 EB                  0806          XCHG         SWAP TO RIGHT REGS
05E8 C9                  0807          RET
05E9 93                  0808 BIGH     SUB  E       BC:=BC-DE
05EA D2 DC 05            0809          JNC  RETPART BC SET IF NO BORROW
05ED 05                  0810          DCR  B       BORROW
05EE C3 DC 05            0811          JMP  RETPART
05F1 F9                  0812 EXACT    SPHL
05F2 C1                  0813          POP  B       GET PTR TO NEXT FQE
05F3 2A B2 04            0814          LHLD PREVFQE CHAIN BACK
05F6 F9                  0815          SPHL         TO PREVIOUS
05F7 E1                  0816          POP  H       INCREMENT SP 2
05F8 C5                  0817          PUSH B       CLOSE CHAIN
05F9 C3 DF 05            0818          JMP  EXIT
05FC C3 00 F0            0819 NOSPAC   JMP  ABEND   NO MEMORY, EXECUTION ABORTED
05FF                     0820 NIL      EQU  0+ATOMIC
05FF CD 49 00            0821 DEFNE    CALL NULL
0602 C8                  0822          RZ
0603 E5                  0823          PUSH H       SAVE
0604 CD 10 00            0824          CALL CDR
0607 CD FF 05            0825          CALL DEFNE   RECURSE
060A E3                  0826          XTHL
060B CD 16 06            0827          CALL DEF
060E EB                  0828          XCHG
060F E3                  0829          XTHL         TOP=DEF,H=DEFINE
0610 EB                  0830          XCHG
0611 CD 16 00            0831          CALL CONS
0614 D1                  0832          POP  D
0615 C9                  0833          RET
0616 CD 09 00            0834 DEF      CALL CAR
0619 E5                  0835          PUSH H
061A D5                  0836          PUSH D
061B 54                  0837          MOV  D,H
061C 5D                  0838          MOV  E,L     SAVE ARG
061D CD 06 00            0839          CALL CADR    FN DEF
0620 EB                  0840          XCHG
0621 CD 09 00            0841          CALL CAR
0624 CD 16 00            0842          CALL CONS    FOR A-LIST
0627 EB                  0843          XCHG
0628 2A B4 04            0844          LHLD EVQAL   CURRENT LIST
062B EB                  0845          XCHG
062C CD 16 00            0846          CALL CONS
062F 22 B4 04            0847          SHLD EVQAL
0632 D1                  0848          POP  D
0633 E1                  0849          POP  H
0634 C3 09 00            0850          JMP  CAR     =CALL,RET
0637 6F 06               0851 ASMFN    DW   AF1     ASSOC LIST OF ASSM FNS
0639 3B 06               0852          DW   AF3
063B 73 06               0853 AF3      DW   AF4
063D 3F 06               0854          DW   AF6
063F 77 06               0855 AF6      DW   AF7
0641 43 06               0856          DW   AF9
0643 7B 06               0857 AF9      DW   AF10
0645 47 06               0858          DW   AF12
0647 7F 06               0859 AF12     DW   AF13
0649 4B 06               0860          DW   AF15
064B 83 06               0861 AF15     DW   AF16
064D 4F 06               0862          DW   AF18
064F 87 06               0863 AF18     DW   AF19
0651 53 06               0864          DW   AF21
0653 8B 06               0865 AF21     DW   AF22
0655 57 06               0866          DW   AF24
0657 8F 06               0867 AF24     DW   AF25
0659 5B 06               0868          DW   AF27
065B 93 06               0869 AF27     DW   AF28
065D 5F 06               0870          DW   AF30
065F 97 06               0871 AF30     DW   AF31
0661 63 06               0872          DW   AF33
0663 9B 06               0873 AF33     DW   AF34
0665 67 06               0874          DW   AF36
0667 9F 06               0875 AF36     DW   AF37
0669 6B 06               0876          DW   AF39
066B A3 06               0877 AF39     DW   AF40
066D 00 80               0878          DW   NIL
066F 36 84               0879 AF1      DW   CARAT+ATOMIC
0671 A7 06               0880          DW   AF2
0673 3A 84               0881 AF4      DW   CDRAT+ATOMIC
0675 AB 06               0882          DW   AF5
0677 43 84               0883 AF7      DW   CONSAT+ATOMIC
0679 AF 06               0884          DW   AF8
067B 31 84               0885 AF10     DW   ATOMAT+ATOMIC
067D B3 06               0886          DW   AF11
067F 4F 84               0887 AF13     DW   EQAT+ATOMIC
0681 B7 06               0888          DW   AF14
0683 48 84               0889 AF16     DW   DEFINAT+ATOMIC
0685 BB 06               0890          DW   AF17
0687 6D 84               0891 AF19     DW   CADDAT+ATOMIC
0689 BF 06               0892          DW   AF20
068B 73 84               0893 AF22     DW   CADRAT+ATOMIC
068D C3 06               0894          DW   AF23
068F 78 84               0895 AF25     DW   NULLAT+ATOMIC
0691 C7 06               0896          DW   AF26
0693 7D 84               0897 AF28     DW   EQULAT+ATOMIC
0695 CB 06               0898          DW   AF29
0697 83 84               0899 AF31     DW   PAIRAT+ATOMIC
0699 CF 06               0900          DW   AF32
069B 8B 84               0901 AF34     DW   ASOCAT+ATOMIC
069D D3 06               0902          DW   AF35
069F 91 84               0903 AF37     DW   EVALAT+ATOMIC
06A1 D7 06               0904          DW   AF38
06A3 96 84               0905 AF40     DW   APLYAT+ATOMIC
06A5 DB 06               0906          DW   AF41
06A7 09 00               0907 AF2      DW   CAR
06A9 01 00               0908          DW   1
06AB 10 00               0909 AF5      DW   CDR
06AD 01 00               0910          DW   1
06AF 16 00               0911 AF8      DW   CONS
06B1 02 00               0912          DW   2
06B3 2B 00               0913 AF11     DW   ATOM
06B5 81 00               0914          DW   81H
06B7 31 00               0915 AF14     DW   EQ
06B9 82 00               0916          DW   82H
06BB FF 05               0917 AF17     DW   DEFNE
06BD 01 00               0918          DW   1
06BF 03 00               0919 AF20     DW   CADDR
06C1 01 00               0920          DW   1
06C3 06 00               0921 AF23     DW   CADR
06C5 01 00               0922          DW   1
06C7 49 00               0923 AF26     DW   NULL
06C9 81 00               0924          DW   81H
06CB B3 00               0925 AF29     DW   EQUAL
06CD 82 00               0926          DW   82H
06CF EB 00               0927 AF32     DW   PAIRLIS
06D1 03 00               0928          DW   3
06D3 16 01               0929 AF35     DW   ASSOC
06D5 02 00               0930          DW   2
06D7 3C 01               0931 AF38     DW   EVAL
06D9 02 00               0932          DW   2
06DB 96 01               0933 AF41     DW   APPLY
06DD 03 00               0934          DW   3
```

A Curve-Fitting Program Using a Focal Interpreter on the KIM-1

BY THEODORE E. BRIDGE
54 Williamsburg Dr.
Springfield, MA 01108

I have three reasons for writing this article. First I want to report on my experience with the FCL65E interpreter that I purchased from the 6502 Program Exchange, 2920 Moana, Reno, NV 89509:

Program (HYPERTAPE on cassette)	$19.00
Mini Manual	6.00
User's Manual	12.00
Listing	35.00
Total	$72.00

Not being able to locate a suitable BASIC interpreter for my KIM-1, I decided to give this a try. My first project was this curve fitting program. I believe the FCL65E interpreter is more compact than BASIC, possibly a little slower, but quite adequate. It has excellent diagnostics. It even has string functions.

My second reason was that I would like to see a BASIC version of this program published in *DDJ*, hopefully with a timing comparison. With my program, it took 40 seconds to calculate the regression coefficients in example 1 with six degrees of freedom, and 16 seconds in example 2 with four degrees of freedom. I think there may be many readers who would like to have a BASIC version of this program.

My third reason is that I want to donate this program to the public domain. It is very handy if you want to read data from tables or curves, or even if you want to make sense out of a large mass of test data.

I once used a FORTRAN version of this program to analyze a large block of test data on the performance of a very big centrifugal pump. None of us could make any sense out of the data, but the computer could. It showed that the efficiency peaked at the end of the first month, and then started to fall. We were then able to optimize the maintenance schedule on the pump. You could use a similar program to optimize the spark and throttle setting on a gas engine as a function of speed, load, and temperature.

In the listing, I present a FOCAL program for finding a formula that will relate a dependent variable, W, to two independent variables, X and Y. When we say $W = f(X, Y)$ we simply mean that the variable W depends in a definite way on variables X and Y. For any value of X and Y there is one and only one possible value to W. To make this analysis, we assume that a polynomial equation can be found to express the relationship.

We will now consider an example having 3 degrees of freedom (a parabolic fit) with respect to X, and 2 degrees of freedom (a linear fit) with respect to Y.

The polynomial equation will look like this: W will be the given value, and V the calculated value of the dependent variable:

$$V = C(0) + C(1)X + C(2)X^2 + (C(3) + C(4)X + C(5)X^2)\,Y$$

So you can see, if we knew the coefficients in the array, C, we could calculate a value of V for any combination of values for X and Y.

Our first task is to find values for these coefficients that will give a minimum error. Since there are 6 unknowns in the array, C, we will need 6 equations. These regression equations have been developed in the science of statistics and are not hard to program. We will express these equations in matrix form. For each data point (one value of X, Y, and W) we will develop a correction to be added to the matrix and will collect the total in array, T.

We will build the first row of the correction matrix in array, U, and each of the following terms will represent consecutive elements in this array. There are 7 terms representing this first equation. The last term is the given value of the dependent varaible, W. Thus:

$$U = 1,\ X,\ X^2,\ Y,\ YX,\ YX^2,\ W$$

Using the following algorithm, we will build the regression matrix in array, T. It will have 6 rows and 7 columns. We initialized array T = 0. For each data point, we will update every element of array T as follows:

$$T(I, J) = T(I, J) + U(I) * U(J)$$

I varies from zero to M1, and J varies from zero to M. Let X1 = the degrees of freedom for X, and Y1 = the degrees of freedom for Y. Then M = X1 * Y1, and M1 = M − 1.

We must have at least M unique data points—as many as we have total degrees of freedom. That is, at least one data point for every row in the matrix T. The more, the better. After all data points have been processed, we will perform a Gaussian elimination to solve the simultaneous equations for the regression coefficients.

This program was written in FCL65E (FOCAL) for the KIM-1. FOCAL is very similar to BASIC:

D	for	(DO)	in	FCL65	is	like	GOSUB	in	BASIC
C	"	(COMMENT)	"	"	"	"	REM	"	"
F	"	(FOR)	"	"	"	"	FOR	"	"
G	"	(GO)	"	"	"	"	GOTO	"	"
I	"	(IF)	"	"	"	"	IF	"	"
S	"	(SET)	"	"	"	"	LET	"	"
T	"	(TYPE)	"	"	"	"	PRINT	"	"
A	"	(ASK)	"	"	"	"	INPUT	"	"

The DO statement in FOCAL assumes an implied return at the end of the line (i.e., D 6.8 in line 5.7 of the listing); or, at the end of the group called (i.e. D 3 at the end of line 7.1). However, a return will be made whenever the command R is encountered. The FOR statement in FCL65E assumes an implied NEXT command at the end of the line.

The IF statement in FCL65E is like a FORTRAN II IF in that it cannot handle logical operators. Rather, the expression within the parentheses is evaluated. A branch is made to one of three line numbers that follow the closing parenthesis, depending on the value of the expression. It branches to the first line number if the expression is negative, to the second if it is zero, and to the third if it is positive. If any of these line numbers are missing, control drops through to the next instruction.

I believe that the comments on the listing and on the examples that follow will complete the explanation of the program and its use.

Listing

```
C FOCAL-65 (V3D) 26-AUG-77

 1.10 S X1=3,Y1=2,P=2.3,G=1;C HERE,WE SPECIFY THE DEGFEES
 1.11 C  OF FREEDOM FOR EACH IND. VARIABLE
 1.14 T &1,!!!,"EXAMPLE NO. ",G,!!
 1.20 S L=5,M=X1*Y1,N1=M-1,N2=M+2,X2=X1-1,Y2=Y1-1;G 10.1
 2.01 C SQUARE ROOT ROUTINE
 2.10 S &3=.2000001;I (&-&3)2.3;S &1=&,&2=2
 2.20 S &2=&1/&;I (FABS(&2-&)-&*&3)2.3;S &=(&+&2)/2;G 2.2
 2.30 F
 3.09 C GAUSSIAN ELIMINATION ROUTINE
 3.10 I (K-1)3.2,3.3,3.2
 3.20 S I5=I+J,T(I5)=T(I5)-T(K1)*T(I+K2)
 3.30 F
 4.01 C FUNCTION TO PRINT IN SCIENT. NOTATION
 4.10 S &1=G;I (-&)4.2,4.9;T "-";S &=-&
 4.20 I (&-1)4.7;I (&-10)4.9;S &1=&1+1,&=&/10;G 4.2
 4.70 S &1=&1-1,&=&*10;G 4.2
 4.90 T %1.05,&,"E",%1,&1," ";F
 5.09 C IN U, WE BUILD FIRST ROW OF REGRESSION MATRIX
 5.10 S E=1;F J=0,X1,M1;S E=E*Y
 5.15 S U(M)=W
 5.17 C IN TU, WE BUILD THE MATRIX
 5.19 F J=0,M1;F I=0+J*M,T(I1)=T(I1)+U(I)*U(J)
 5.40 T 1;F
 5.60 S A=1;F I=0,X2;S U(I+J)=A*E,A=A*X
 5.70 T 1;F C=0,M,M2;F 6.8;COPY MATRIX
 6.09 COLUMN C IS PRINTED IN LINE 6.8
 6.40 S F=C+1;I (M1-D)6.7
 6.60 S Z=FSER(4,T(F+C));COPY IN SCIENT NOTATION
 6.70 T
 6.80 T 1;F F1=C,6,M1;T 1;F C=0,5;D 6;COPY 6 NOS./LINE
 7.01 COEFFICIENTS ARE PRINTED AFTER SOLVING EQUATIONS
 7.10 F K=0,M1;D F;T 2;S K=K*M,A=T(K+K2);F J1=K,M1;D 7.5;F I=0,M1;D 3
 7.15 T !!!,"REGRESSION COEFFS."
 7.20 T 1;S C=M2;D 6.8
 7.30 T 1;F;C K1 POINTS TO ROW K,COL J; K2 TO ROW 0,COL K
 7.50 S J=(J1+1)*M,K1=K+J,T(K1)=T(K1)/A
 8.09 C REPEAT DATA AND TABULATE RESULTS
 8.10 S V=0,K=N1+M2;F J=1,Y1;S V=V*Y;I 8.4;S V=V+V1
 8.20 S E=V-U,E2=E2+E*2,N=N+1;T !,N," "
 8.30 S A=FSER(4,X),A=FSER(4,Y),A=FSER(4,V),A=FSER(4,V),A=FSER(4,E)
 8.35 F
 8.40 S V1=0;F I=1,X1;S V1=V1*X+T(K),K=K-1
 8.50 I (L-8)8.7
 8.60 T !!!,"RMS ERROR ";S A=FSER(4,FSER(2,E2/N));T 1;G 20.1
 8.70 T 7
 8.80 T !!,"          X          Y          W          CALC      ERROR"
 8.90 S L=8,E2=0,N=0;G 10.1

10.01 COORDINATES FOR 6 DATA POINTS ARE ENTERED HERE. THEY ARE READ FROM
10.03 CURVE 1. THIS DATA IS READ TWICE BY THE PROGRAM:FIRST TO BUILD THE
10.05 C   6X6  REGRESSION MATRIX, AND AGAIN TO SOLVE FOR THE REGRESSION
10.07 COEFFICIENTS. FOR EACH POINT, THE CALCULATED VALUE V IS TO BE
10.09 COMPARED TO THE GIVEN VALUE W.
10.10 S X=1,W=3,Y=1;D L
10.20 S X=2,W=4;D L
10.25 S X=0,W=.75;D L
10.30 S X=1,W=6,Y=2;D L
10.40 S X=2,W=11;D L
10.60 S X=0,W=3.764;D L

20.01 G 8.5
20.02 C HERE, WE PLOT A GRAPH OF THE RESULTS
20.10 T !!!;F X=0,.1,2;T !;D 20.4;D 21
20.20 T !!!,"CURVE ",Q," ---  W  VS  X,Y";T !!;G
20.30 E;S X1=2,Y1=2,P=5.7,G=2;G 1.14
20.40 F Y=1,.5,2;D 8.1;S W(2*Y)=V*5+.5

21.04 S Z3=FINT(X+.05)-X
21.05 S Z=-1,A=32;I (Z3)21.1;S A=45
21.10 T "  ";F N=1,W(2);D 22
21.20 D 22.3;F N1=N+1,W(3);D 22
21.30 D 22.3;F N=N1+1,W(4);D 22
21.40 D 22.3

22.10 I (Z)22.2;S Z=Z-1,S1=FOUT(A);R
22.20 S Z=8;T "I";R
22.30 T ".";I (Z)22.4;S Z=Z-1;R
22.40 S Z=8;R
```

EXAMPLE NO. 1
The following magic number was typed by the program just to let you know that something is happening during the long wait.

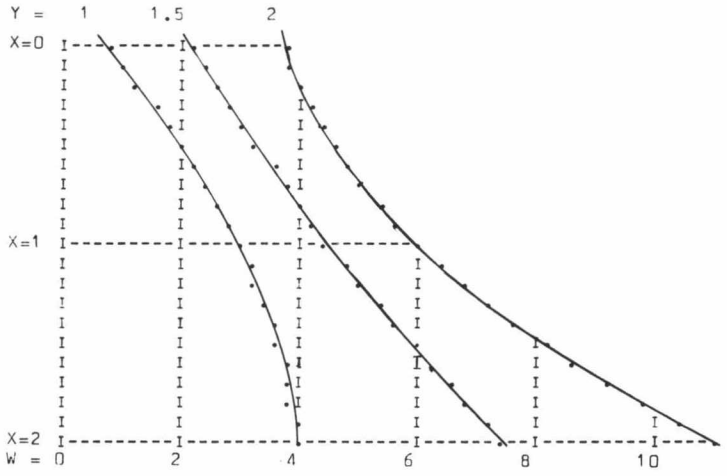
```
!!!!!!!!!!!!!!!
```

It took 40 seconds to calculate the REGRESSION COEFFS.

```
-2.2640CE0 4.8960CE0 -2.6320CE0 3.0140CE0 -2.0210CE0 2.0070CE0
```

```
          X          Y          W       CALC      ERROR
1  1.0000CE0  1.0000CE0  3.0000CE0  3.0000CE0 -8.9407CE-8
2  2.0000CE0  1.0000CE0  4.0000CE0  4.0000CE0  2.2351CE-8
3  0.0000CE0  1.0000CE0  7.5000CE-1 7.5000CE-1 -2.7939CE-8
4  1.0000CE0  2.0000CE0  6.0000CE0  6.0000CE0  8.1956CE-8
5  2.0000CE0  2.0000CE0  1.1000CE1  1.1000CE1 -4.4703CE-8
6  0.0000CE0  2.0000CE0  3.7640CE0  3.7640CE0 -7.4505CE-9
```

RMS ERROR 0.0000CE0



EXAMPLE NO. 2 In this example, we use the same data, but with 2 (not 3) degrees of freedom for variable X.

```
!!!!!!
```

```
6.0000CE0 6.0000CE0 9.0000CE0 9.0000CE0

6.0000CE0 1.0000CE1 9.0000CE0 1.5000CE1

9.0000CE0 9.0000CE0 1.5000CE1 1.5000CE1

9.0000CE0 1.5000CE1 1.5000CE1 2.5000CE1

2.8514CE1 3.9000CE1 4.9278CE1 6.7000CE1 2
```

You may find this printout useful for debugging, should you wish to translate this program into BASIC for use on your machine. It is printed before each sweep while solving the equations for the regression coefficients in group seven lines of the program.

```
6.0000CE0 6.0000CE0 9.0000CE0 9.0000CE0

1.0000CE0 4.0000CE0 0.0000CE0 6.0000CE0

1.5000CE0 0.0000CE0 1.5000CE0 1.5000CE0

1.5000CE0 6.0000CE0 1.5000CE0 1.1500CE1

4.7523CE0 1.0486CE1 6.5670CE0 2.4229CE1 2
```

```
6.0000CE0 6.0000CE0 9.0000CE0 9.0000CE0

1.0000CE0 4.0000CE0 0.0000CE0 6.0000CE0

1.5000CE0 0.0000CE0 1.5000CE0 1.5000CE0

0.0000CE0 1.5000CE0 1.5000CE0 2.5000CE0

2.1308CE0 2.6215CE0 6.5670CE0 8.5000CE0 2
```

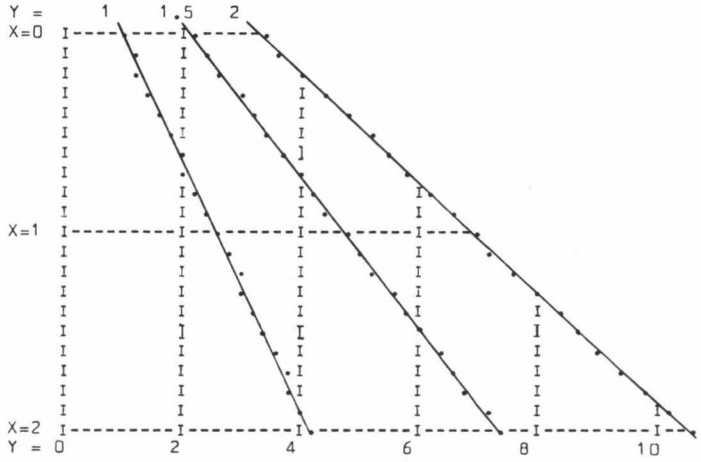
```
6.0000E0 6.0000E0 9.0000E0 9.0000E0

1.0000E0 4.0000E0 0.0000E0 6.0000E0

1.5000E0 0.0000E0 1.5000E0 1.5000E0

-1.5000E0 1.5000E0 1.0000E0 1.0000E0

-4.3761E0 2.6215E0 4.3380E0 1.9930E0 2
```

It took 16 seconds to calculate the REGRESSION COEFFS.

```
-1.3867E0 -3.6800E-1 2.3450E0 1.9930E0
```

```
          X          Y          W       CALC      ERROR
1  1.0000E0  1.0000E0  3.0000E0  2.5833E0 -4.1667E-1
2  2.0000E0  1.0000E0  4.0000E0  4.2083E0  2.0833E-1
3  0.0000E0  1.0000E0  7.5000E-1 9.5833E-1  2.0833E-1
4  1.0000E0  2.0000E0  6.0000E0  6.9213E0  9.2133E-1
5  2.0000E0  2.0000E0  1.1000E1  1.0539E1 -4.6067E-1
6  0.0000E0  2.0000E0  3.7640E0  3.3033E0 -4.6067E-1
```

RMS ERROR 5.0558E-1 Since we are trying to fit a straight line to parabolic data, we would expect some error.



Binary Tree Manipulation on the 8080

BY MIKE GABRIELSON
Box 2692
Stanford, Calif. 94305

Imagine that we are writing a compiler. How should the symbol table be organized? One possibility is to use a binary tree. Figure 1 illustrates a binary tree that contains seven symbols (strings representing variable names). Each symbol is contained in a *node* along with two *pointers*. The pointers are used to link together the nodes of the tree. A special pointer called the *root* points to the topmost node of the tree. (Computer trees grow upside down.) *Grounded* pointers on the bottommost nodes have nothing to point to.

On the 8080 and similar computers, we can let each pointer be a 16 bit address, and each symbol can be an ASCII string. Figure 2 shows a portion of our tree as it might actually be stored in an arbitrary region of memory. By convention, we will set grounded pointers to a zero value, and delimit each string with the NUL character. An actual symbol table for a compiler would include some extra bytes in each node for flags, values, and other information about each symbol, but we'll ignore that to simplify the examples.

Figure 1. A Binary Tree

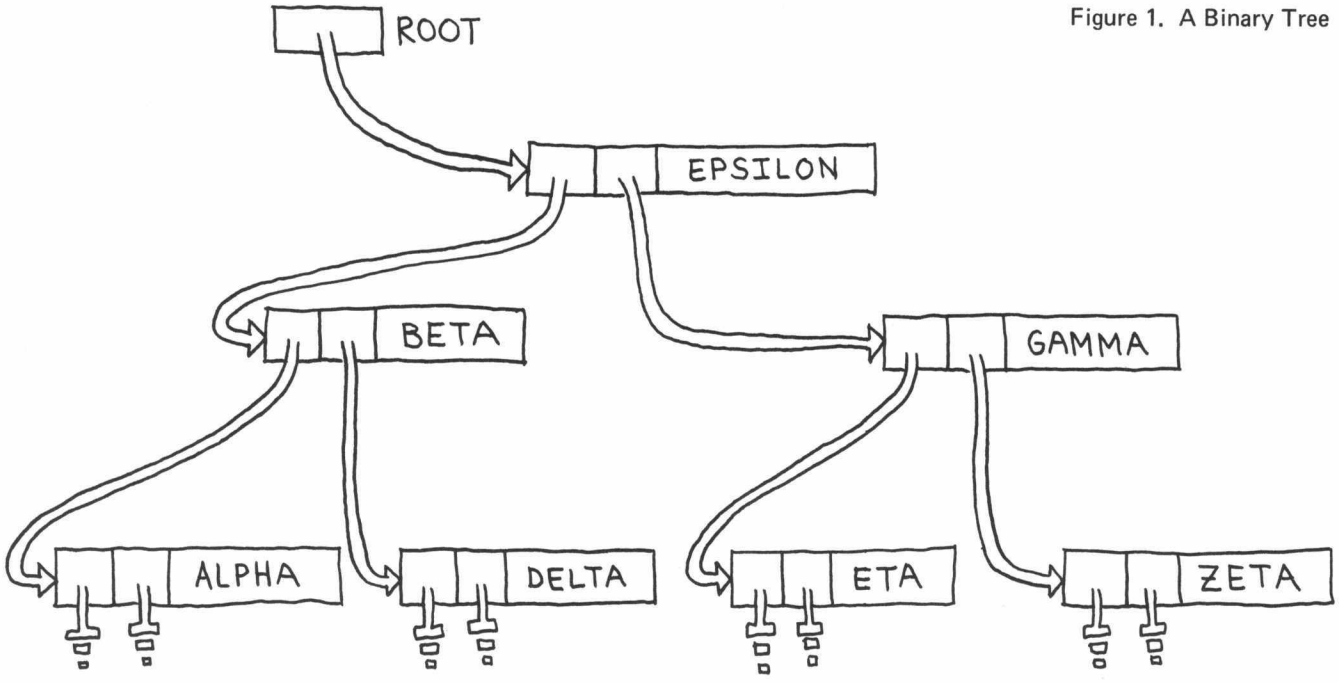
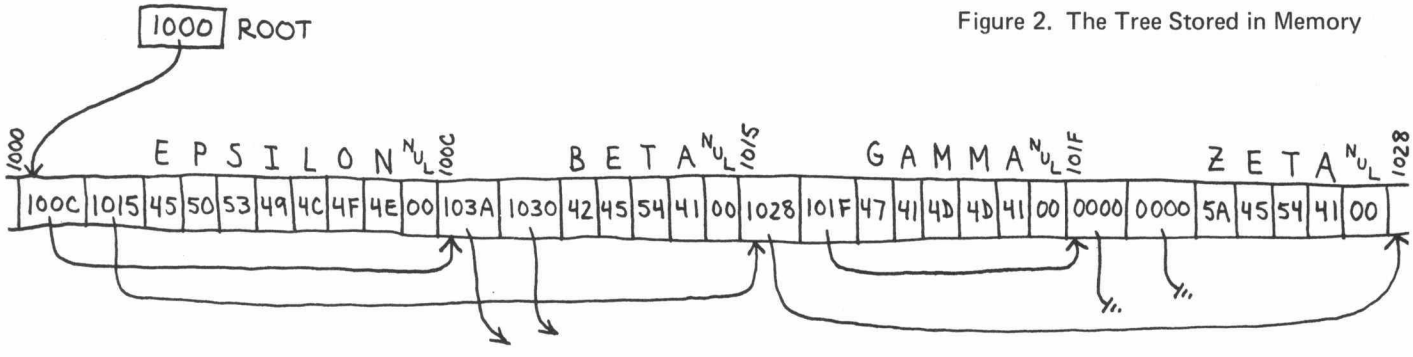


Figure 2. The Tree Stored in Memory



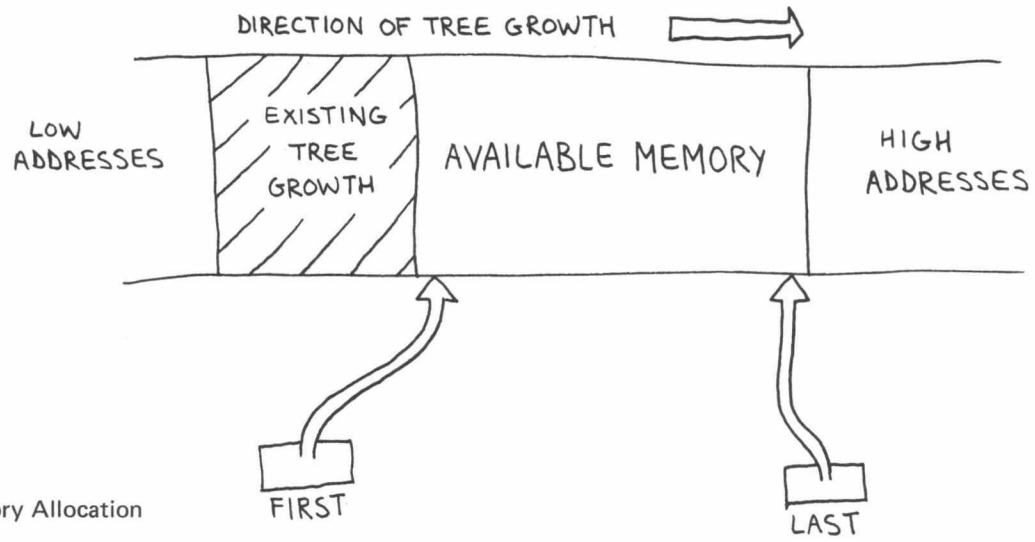


Figure 3. Memory Allocation

The symbols in our tree are arranged in a special way: if we pick any node, we find that the left pointer (if not grounded) points to a *subtree* containing symbols all alphabetically less than (preceding) the symbol in the node. Similarly, the right pointer points to a subtree containing symbols all alphabetically greater than (following) the symbol in the node. Maintaining the tree in this ordered manner allows us to design very simple but effective algorithms to manipulate the data in the tree. For example, suppose our compiler needs to look for a symbol in the tree. How should the tree be searched? Starting with the first node (pointed to by the root), we compare the node's symbol to the symbol we desire. If there's a match, we're done. If our symbol is less than the node's symbol, we follow the node's left pointer and repeat the test with that node. If our symbol is greater, we pick the right subtree, and so on down the branches until we find a match or until we reach a grounded pointer, which indicates the symbol is nowhere in the tree. The sample subroutine called SEARCH will search a binary tree like ours using this algorithm.

```
1                ; SEARCH - search symbol tree for an ASCII string
2                ; Mike Gabrielson   8/10/78
3                ; accepts: BC = address of string to look for, terminated by NUL
4                ;          DE = address of ROOT pointer
5                ;          HL = contents of ROOT pointer (equal to address of
6                ;               first node if not grounded)
7                ; returns: carry set if string not found in tree (DE = address
8                ;               of last grounded pointer)
9                ;          else carry cleared (string found, HL = address of
10               ;               node containing the matching string)
11               ;=============================================================
12
13                       NUL ='@      ;end of string character
14
15 0000 7C      SEARCH: MOV  A,H      ;is the pointer grounded,
16 0001 B5              ORA  L        ;indicating end of tree?
17 0002 37              STC           ;(assume "yes")
18 0003 C8              RZ            ;yes, end of tree, string not found
19 0004 C5              PUSH B        ;no, save address of caller's string
20 0005 E5              PUSH H        ;save address of node
21 0006 23              INX  H        ;skip past the node's
22 0007 23              INX  H        ;left and right subtree pointers
23 0008 23              INX  H        ;and stop at the first character
24 0009 23              INX  H        ;of the node's string
25 000A 0A      NEXTC:  LDAX B        ;get character from caller's string
26 000B BE              CMP  M        ;same as character in node's string?
27 000C C21600          JNZ  TESTED   ;no, wrong node
28 000F 03              INX  B        ;yes,
29 0010 23              INX  H        ;point to next character pair
30 0011 FE00            CPI  NUL      ;was that the end of the strings?
31 0013 C20A00          JNZ  NEXTC    ;no, test next pair
32 0016 E1      TESTED: POP  H        ;restore address of node
33 0017 C1              POP  B        ;restore address of caller's string
34 0018 C8              RZ            ;done if strings matched (carry's cleared)
35 0019 DA1E00          JC   LEFT     ;pick left subtree if caller's string is low
36 001C 23              INX  H        ;else get address of right subtree pointer
37 001D 23              INX  H
38 001E 56      LEFT:   MOV  D,M      ;get address of subtree into DE
39 001F 23              INX  H
40 0020 5E              MOV  E,M
41 0021 EB              XCHG          ;then into HL
42 0022 1B              DCX  D        ;leave DE pointing to last node accessed
43 0023 C30000          JMP  SEARCH   ;continue searching down the tree
```

How do we grow a tree in the first place? One necessary ingredient to grow a tree is memory. Let's assume we have access to two new pointers. One pointer is called FIRST and is the address of the first byte of a contiguous chunk of memory available for the tree. The last available byte in the chunk is pointed to by LAST. This arrangement is illustrated in Figure 3. Each time our tree needs a byte to grow, it can grab the byte pointed to by FIRST, and then increment FIRST. Memory is exhausted when FIRST passes LAST. Now we can modify our search so that when a symbol is not found in the tree, a new node is automatically grown for it. The next listing is a subroutine called LOOKUP which appends new symbols to the tree as necessary. (To simplify the code, this version of LOOKUP does not check if FIRST passes LAST, indicating memory overflow.) An example of a call to LOOKUP is

```
LXI  B,STRING    ;get address of string
LXI  H,ROOT      ;get address of ROOT
MOV  D,M         ;get contents of ROOT
INX  H
MOV  E,M
DCX  H           ;restore address of ROOT
XCHG             ;adjust registers for LOOKUP
CALL LOOKUP
```

The calling sequence can be simplified if the contents of the 16-bit root and node pointers are maintained in standard Intel byte-reversed format, instead of unreversed as the code currently requires. Changing the code so that all pointers are byte-reversed is an easy exercise left to the reader. Figure 4 shows how the tree has grown after a LOOKUP of the symbol THETA.

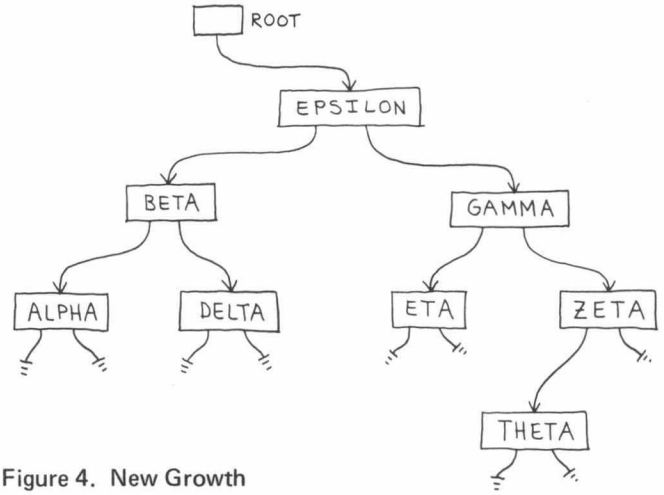


Figure 4. New Growth

```
1                ; LOOKUP - find ASCII string in symbol tree
2                ; Mike Gabrielson   8/10/78
3                ; accepts: BC = address of string to look for, terminated by NUL
4                ;          DE = address of ROOT pointer
5                ;          HL = contents of ROOT pointer (equals address of
6                ;               first node if not grounded)
7                ; returns: HL = address of tree node containing the string
8                ;          carry set if a new node had to be grown
9                ;          else carry cleared (symbol already in tree)
10               ; MEMORY OVERFLOW IS NOT TESTED FOR!
11               ;=============================================================
12
13                       NUL = 0        ;end of string character
14
15                       !EXTRN SEARCH  ;SEARCH is not defined in this assembly
16                       !EXTRN FIRST   ;external pointer containing address of
17                                      ;first byte of available memory for
18                                      ;growing new nodes
19
20 0000 CD---- LOOKUP: CALL SEARCH ;symbol already in tree?
21 0003 D0             RNC         ;yes, all done
22 0004 2A----         LHLD FIRST  ;no, get address of node about to sprout
23 0007 E5             PUSH H      ;save address of new node for return
24 0008 EB             XCHG        ;HL := address of last pointer SEARCHed
25 0009 72             MOV  M,D    ;replace grounded pointer with address
26 000A 23             INX  H      ;of new node
27 000B 73             MOV  M,E
28 000C AF             XRA  A      ;clear register A
29 000D 12             STAX D      ;and ground the two subtree pointers in the
30 000E 13             INX  D      ;new node
31 000F 12             STAX D
32 0010 13             INX  D
33 0011 12             STAX D
34 0012 13             INX  D
35 0013 12             STAX D
36 0014 13             INX  D
37 0015 0A     SAVNAM: LDAX B      ;get next character from caller's string
38 0016 12             STAX D      ;save in new node
39 0017 03             INX  B      ;adjust pointers for next character
40 0018 13             INX  D
41 0019 FE00           CPI  NUL    ;was that the end of the string?
42 001B C21500         JNZ  SAVNAM ;no
43 001E EB             XCHG        ;get address of next available byte into HL
44 001F 22----         SHLD FIRST  ;save new FIRST
45 0022 E1             POP  H      ;yes, restore address of new node
46 0023 37             STC         ;indicate new node was grown for caller
47 0024 C9             RET
```

A nice feature for our compiler would be the capability of listing the symbol table in alphabetical order. How can we list our tree? Because of the way the tree is ordered, one elegant algorithm we can use works as follows: before listing any node, first list the node (subtree, if any) pointed to on the left, then list the original node, then list the node (subtree) pointed to on the right. Start with the node pointed to by the root, but *always* try to first visit the left subtree at each node.

This is a recursive algorithm that is easy to implement on machines with a stack like the 8080. The final sample program is a recursive routine called PUTREE that will dump our tree using this algorithm. An example of a call to PUTREE is

```
LHLD ROOT      ;get contents of ROOT
MOV  A,H       ;swap H and L
MOV  H,L       ;to be consistent with
MOV  L,A       ;rest of tree
ORA  H         ;is root grounded?
CNZ  PUTREE    ;no, output tree
```

A way to visualize what's happening is to imagine an ant visiting each node by crawling around the edges of the tree (see Figure 5). If the sun is shining from the upper left, and the ant shouts out each symbol the *first* time it crawls along a *shaded* side of the symbol's node, then the symbols will be shouted out in alphabetical order.

```
1                ; PUTREE - output symbol tree in alphabetical order
2                ; Mike Gabrielson   8/10/78
3                ; accepts: HL = contents of ROOT pointer (must not be grounded)
4                ;=============================================================
5
6                        !EXTRN PUTNAM ;this is an external routine not
7                                      ;defined in this assembly.  PUTNAM
8                                      ;outputs the ASCII string addressed
9                                      ;by HL (which must be saved) and
10                                     ;terminated by NUL.
11
12 0000 D5     PUTREE: PUSH D      ;save pointer to the pointer when reentered
13 0001 E5             PUSH H      ;save address of node
14 0002 56             MOV  D,M    ;get pointer to left subtree into DE
15 0003 23             INX  H
16 0004 5E             MOV  E,M
17 0005 23             INX  H
18 0006 EB             XCHG        ;then into HL
19 0007 7C             MOV  A,H    ;no left subtree?
20 0008 B5             ORA  L      ;(pointer grounded?)
21 0009 C40000         CNZ  PUTREE ;subtree exists, traverse it now
22 000C EB             XCHG        ;restore address of original node to HL
23 000D 23             INX  H      ;skip right subtree pointer
24 000E 23             INX  H      ;and get address of symbol name
25 000F CD----         CALL PUTNAM ;output the ASCII string
26 0012 2B             DCX  H      ;get node's right subtree pointer into DE
27 0013 5E             MOV  E,M
28 0014 2B             DCX  H
29 0015 56             MOV  D,M
30 0016 EB             XCHG        ;then into HL
31 0017 7C             MOV  A,H    ;no right subtree?
32 0018 B5             ORA  L      ;(pointer grounded?)
33 0019 C40000         CNZ  PUTREE ;subtree exists, traverse it now
34 001C E1             POP  H      ;restore stack
35 001D D1             POP  D
36 001E C9             RET
```

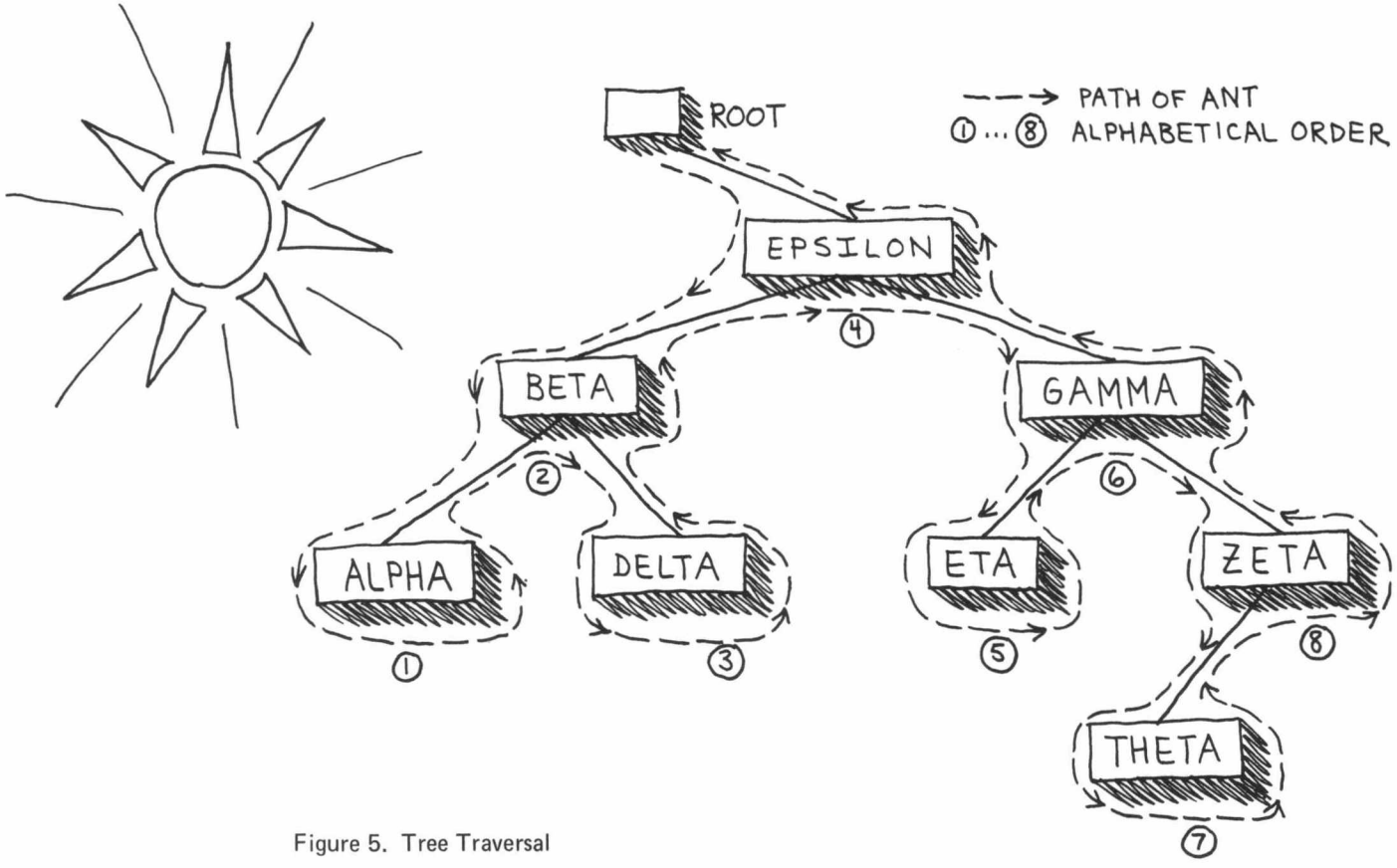


Figure 5. Tree Traversal

PROGRAMMING LANGUAGES AND STANDARDS

BY DAVID CHAPMAN
Western Reserve Academy
Hudson, Ohio 44236

Over the last few years, the tremendous proliferation of microcomputers has been accompanied by a growing reliance on BASIC as the "standard" high-level language. Nearly every microprocessor has at least one BASIC available for it; the 8080 must have at least a dozen. This is an undesirable state of affairs for a number of reasons, not the least of which is that BASIC is an atrocious language. Some people have therefore suggested that some other language should be made standard; but this is not the solution.

Flaws in BASIC have been written about elsewhere[1]. The language has only three datatypes (rel, array, and string), only the most elementary control structures (GOTO, GOSUB-RETURN, and FOR-NEXT), few built-in functions, and only the most primitive function definition facilities; it supports either no files at all or only sequential files, and so on. "But," you say, "surely all these deficiencies could be corrected by relatively minor modifications in the language?" Yes, some could be and have been. The result, however, is often something that is not quite BASIC, which is supposedly a standard. This is because most of the language's problems arise not from semantic difficulties (lack of random-access file capability, for example) but from fundamental syntactic problems: verbosity, verb richness, lack of structure, and a general lack of thinking through. BASIC was, after all, designed as an educational language and as an improved FORTRAN—not for applications programming.

The obvious (though mistaken) thing to do is to replace BASIC with some other language as a standard. Of course, this would mean some revamping in the microcomputer industry. This might not be difficult, as most of the manufacturers suffer from advanced, chronic bandwagonism: every time any one of them comes up with something new, it is immediately copied a dozen times over.

This bandwagonism hurts everyone: the innovators have their ideas stolen; the bandwagonists must compete against many other companies making an identical product, and the users get little freedom of choice in buying their computers and accessories. Bandwagonism has also tended to establish standards, most of them informal, incomplete, and harmful. The S-100 bus, established simply by its use in the Altair, the first computer to be produced in what were then large quantities, has an excessively large number of lines, many of them undefined or multiply defined, and uses expensive hundred-pin connectors.[2] The Kansas City standard cassette interface, by now more or less abandoned, was slow, inefficient, and error-prone.[3] Various other cassette "standards," for example the CUTS system sponsored by *Popular Electronics*, have faded even further into oblivion. It is also worth pointing out that variations between implementations of BASIC generally force one to re-write programs when transporting them from one machine to another. In general, in a field developing as fast as microcomputing, standards will merely hinder progress.

Besides these general considerations concerning the desirability of standards in the microcomputing field, there is the question of applicability when establishing a language as a standard. No language is good at everying; the so-called "general purpose languages" are good at nothing and merely mediocre at everything. BASIC has been discussed above; FORTRAN is very similar, only, if anything, worse. Most of the structured languages, while syntactically relatively clean, are very rigid, limiting, and poor at dealing with amorphous data or problems. PL/I, the language that was to be all things to all people, is a syntactic disaster, a language so huge that few people know all of its arcane abilities and disabilities, a partial combination of ideas and constructs stolen from many other languages.

This article was originally intended to extoll the virtues of my own favorite language, SNOBOL4. SNOBOL4, though designed as a string-processing language, is good at a wide variety of things, including, contrary to popular belief, numeric processing. However, I would not recommend it for all purposes to all people; it is poor at dealing with file I/O, for example, and has even less control structure than BASIC. There is no easy way to correct these deficiencies, as they are due to the basic syntactic structure of the language.

Several people have suggested designing a new language as a standard for microcomputing.[4] This has all of the arguments above against it as well as some others. There are already literally thousands of languages available, and there is no call to write a new one unless one has some radically new idea or application. Moreover, most of the well-known languages already have textbooks available for them, a group of people who are familiar with the language, and often a rigid standard. All of these things would have to be built up from scratch if a new language were made standard. At least one of these attempts is doomed to failure because the author is attempting to include "all of" PL/I, FORTRAN, BASIC, ALGOL, SNOBOL, COBOL, and bits of RPG and APL.[5] Assuming that this is possible in under a megabyte (many big-computer PL/Is run to hundreds of kilobytes), it would be a syntactic catastrophe. Sure, maybe such a language could do everything PL/I, FORTRAN, et al. can, but then so can machine language. Using all that power would be difficult, to say the least. The enormous power and elegance of such languages as APL and SNOBOL4 is due to their syntax, which could not possibly be made compatible with PL/I, for example.

(cont'd on page 21)

SAM76 Language Update No.3-July 1978

BY ANCELME ROICHEL

version <0023>

This new version is upward compatible with previous versions and is the result of user requests for means to facilitate access to major pointers. In addition linking of user programs to the processor is simplified; provisions are made for up to four overlay zones defined by the user. These enhancements were made without, however, increasing the size of the processor. The new functions are defined below:

- |@cn,current,new| Change function Name

This function allows user to rename any of the resident function "current" names to any desired "new" name; these new names must consist of two or three alphabetic characters; in changing names the user should be particularly careful in his command sequence. At this time "os" and "is" should not be changed.

- |qio| Query current IO assignments

The value of this function is in the current "X" base the value of "IOBYTE" as defined by the monitor - if such a facility is provided.

- |sio,iobyte| Set IO assignment

This null valued function is used to reassign the monitor "IOBYTE" if provided for. It should be noted that a "cold" start entry into the processor carries with it the current monitor assignment; if this function is used then a "control C" or "ex" exit causes restoration back to the original assignment. A "warm" reentry, or a "hot" continue causes reassignment to the setting in force at the time of exit.

- |xll,sØ| Xamine Label List

The value of this function is a list of internal system labels available for examination by name; these labels may be actual values, identify locations of program entry points or tables, or give addresses of memory locations in which system pointers may be found. Each label name is preceded by the string symbolized by "sØ". (Label names and their purpose will be described later).

- |xal,label,offset| Xamine Address of Label

The value of this function is the value of the "label" to which the value of the "offset" is added. Both the value returned and the "offset" are in the current "X" base.

- |nud,function,arguments| Null Display mode

The "nud" function serves to modify the scanner to the end that instead of a value being returned by the function whose name is symbolized by "function" (with its appropriate arguments), output of what would have been the value is sent to the currently assigned "console" device.

"ex" Exit function

The exit function in version <0023> has been enhanced to automatically reduce the work space to a suitable minimum prior to returning to the operating system. Reentry automatically restores the work space to its maximum allowable size. Next enhancement will be to implement the passing of a file name and other information back to the operating system.

"cws" Change Work Space

This function has now been enhanced so that execution of %cws/ without arguments serves to reduce the work space to a suitable minimum.

"ot" Output Text

The neutral form of this function may be used to "re-specify" size of inter record gaps and length of leader and trailer; the two forms of this function are:

%ot,n1,n2,...,nN/= which serves to "output texts" whose names are symbolized by "n1", "n2", "nN"; the overall output is bounded by 64 bytes of leader, and individual texts are separated by 16 bytes of gap.

&ot,leader,gap,n1,n2,...,nN/= first resets the specifications for leader and gaps to the new desired decimal values symbolized by "leader" and "gap", and then outputs the texts in the normal manner.

Generalized structure - labels and points of interest

The accompanying figure shows a typical arrangement of the user work space, as well as associated pointers and their names. This work space may be located either below or above the main system program, can be saved after having performed the "ex" exit function, and this saved file can be reloaded in another system with its workspace in a different area of the machine; a "hot" reentry permits the process terminated by the "ex" function to resume with the next and other subsequent unevaluated expressions.

System Labels

Typical system labels as determined by the "xll" Xamine Label List function are given below with their characterization.

```
"AST"  ;  Address System subroutine Table
"ASV"  ;  Address System subroutine Vectors
"ENC"  ;  Entry Continue "hot start"
"ENR"  ;  Entry Restart Level "warm start"
"ESL"  ;  Entry Subroutine Links
"OPV"  ;  Offset Permanent - Variable tables
"POA"  ;  Pointer overlay A - "extra"
"POB"  ;  Pointer overlay B - "disk"
"POC"  ;  Pointer overlay C - "plot"
"POD"  ;  Pointer overlay D - "user"
"PSE"  ;  Pointer service entry
"PSW"  ;  Pointer function "SW"
"PSY"  ;  Pointer function "SY"
"PUL"  ;  Pointer user link table
"PWA"  ;  Pointer Work Area start
"TAE"  ;  Text area end
"TAH"  ;  Text area high
"TAL"  ;  Text area low
"TAM"  ;  Text area max
```

Details of the mechanism of linking the overlay areas to the system function table, and details of user application of system subroutines will be described in a later write up.

Exercise for the adventurous

The inquisitive person may begin to investigate the system subroutine system by finding the code for the "xr", "xrp", "xw", and "xwp" functions. The entry point for functions can be found using the "xqf" function. The function ends with either a "jump" or "return" instruction. In the body of the code are some subroutine calls; the addresses called are also found in two other places: somewhere beyond the start of the table located at "AST" (said table locates some fifty of the system subroutines) and also somewhere - the same index distance (but modulo three) in the table created at address starting at "ASV".

The first table is used if an instruction in one of the two following forms is encountered in a user program:

RST 6	CALL ESL
index number	index number

Alternatively a "call" to the proper location in the table of jump vectors which starts at "ASV" will achieve the same task.

In like manner other subroutines may be identified through exploration of other functions.

"sw" and "sy" linking

The user may make use of the "sw" and "sy" functions by storing at locations determined by %xal,PSW,offset/ and %xal,PSY,offset/, respectively, the memory addresses which execution of these functions should reach. If the offset is Ø, then the temporary table address is given, if the offset is that value returned by executing %xal,OPV/, then the permanent table address is given. The "xrp" and "xwp" functions may be properly used to read and write in these locations.

The value of this function is the contents, in the current "X" base, of the identified sector. Each "byte" is preceded with the string symbolized by sØ.

- |xws,unit,track,sector,X| X Write Sector

This null valued function places the string symbolized by X in the designated sector. If that string is greater than one sector then only as much as will fit will be transferred.

special feature of bf and uf

The "bf" bring file, and "uf" update file functions operate as expected when used with the active function marker - namely: %bf,filename/, and %uf,filename/. If the neutral form is used then any file may be brought in regardless of its type (provided there is room) and may be put back on disk. Care must be used in doing this since strange files will not have the necessary pointers to be compatible with the SAM76 processor. This is a very effective means for moving files from disk to disk - particularly on one disk drive systems.

For example:

```
&bf,xdir.com/=

Now change diskettes

&uf,xdir.com/=
{nav-uf-aef}

Ho-hum an already existing file message - well no matter

%ef,xdir.com/&uf,xdir.com/=
```

Anomaly Indications

Whenever a non fatal anomaly occurs during the execution of file functions, a value is returned by these normally null valued functions. These values are in the form:

{nav-function-anomaly}

In the above indication, the function being executed is symbolized by "function", and the anomaly type may be one or more of the following:

"dsk" disk
"nrm" no room
"err" error
"fnf" file not found
"dir" directory
"clo" close
"ext" extension
"aef" already existing file
"rud" reading unwritten data

CP/M compatible overlay

This overlay is nominally 1K and provides at this time the following functions:

prim	Id No	Description
BF	220;	Bring File
EF	224;	Erase File
FB	226;	File Branch
LF	216;	List Files
QFA	QFA;	Query File Attributes
QFE	QFE;	Query File Extension
RF	243;	Read File
SDU	SDU;	Select Directory Unit
SFE	SFE;	Select File Extension
UF	218;	Update File
WF	244;	Write File
XRS	XRS;	X Read Sector
XWS	XWS;	X Write Sector

Those functions which in the preceding table have a numeric "Id No" have been described previously in Doctor Dobbs as well as in the SAM76 language manual; the other functions are still in the experimental evaluation phase and are described below.

- |qfa,filename,sØ| Query File Attributes

The value of this function is a list of file characteristics derived from the disk operating system directory; each attribute is preceded by the string symbolized by "sØ". One of the attributes is the file size expressed as a decimal number of 256 byte pages.

- |qfe| Query File Extension

The value of this function is the current setting of the file extension; note that there is a difference between "blanks" or "spaces" and "null" or zero.

- |sdu,d| Select Directory Unit

This null valued function allows the user to designate the Directory unit to be accessed by the various file functions; a Directory unit is any device which has a directory structure and could be cassette as well as disk.

- |sfe,extension| Select File Extension

This null valued function is used to preset a desired file extension so that the extension need not be used in any of the file functions. In effect this may be used as a classification means. The extension may be set to any desired three (or less) symbols, including spaces and ???, or to the null string (viz: %sfe/) with varying effects.

- |xrs,unit,track,sector,sØ| X Read Sector

Availability

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

Addresses in table below are all in HEX.

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

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

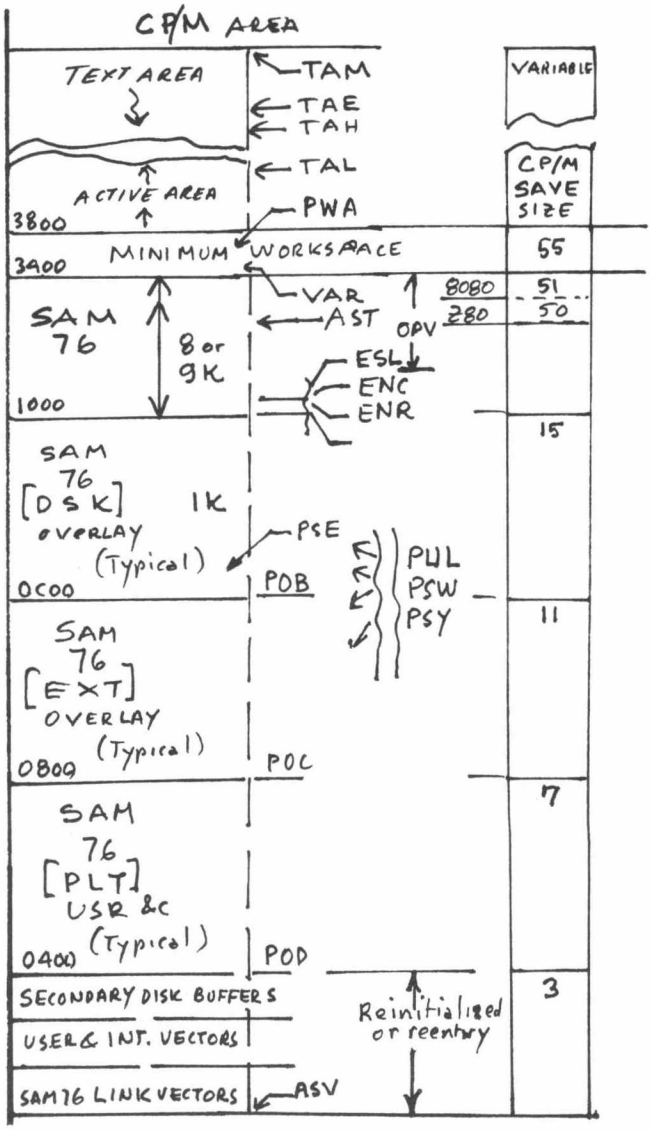
Text book describing the language in simple terms is available for a price of $12 - postpaid. (Specify bound or loose leaf). System loader booklet appropriate to object code format is $2.

| Source for CP/M Disk functions |

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

SAM76 Inc., Box 257, R.R.1, Pennington, N.J., 08534

Typical System Layout



(cont'd from page 18)

Recently, PASCAL has been hailed as the "next" standard language.[6] Besides the arguments above against standardization, I am not very impressed with PASCAL as a language. Some syntactic kluges have been pointed out already,[7] but the main problem with PASCAL is that it is one of the class known as "structured" languages; a misnomer, as SNOBOL4, for example, has a much neater, more elegant, and simpler syntax. That PASCAL is structured means essentially that the programmer is limited with respect to the control structure that he can use. This makes the source code easier to understand and debug, but it restricts the programmer and may discourage creativity. The controversy over structured versus nonstructured languages that raged in *Communications of the ACM* and similar publications for several years seems now to be over, structured programming techniques having emerged victorious; but this does not mean that you, or I, or anyone else has to agree.

Rather than making PASCAL or some other single language a standard, what is needed is a total lack of standards, allowing the programmer to choose which language to use for a given application based on the problem's nature and his personal preference. Each microcomputer manufacturer could write one or several languages that had not yet been implemented for their processor. Transportability between machines could be provided by device-independent I/O: the user, or, better, the maker of the user's mainframe, would write a set of I/O subroutines for the user's hardware configuration and I/O devices that would be called by the language interpreters and compilers, these being written with calls to user-provided routines only. Relocatability of code would be very helpful. Alternatively, the writer of the language might spend the small amount of extra work needed to modify the basic program for use with a variety of common systems. Some software is already being offered this way.

Some obvious first choices for languages to implement on microcomputers are PASCAL, ALGOL, LISP or one of its many variants, SNOBOL4, RPG II, GRASS, SKETCHPAD, APL, MUMPS, C (anyone for MULTICS on a micro?), GPSS, SIMSCRIPT, and a TRAC*-like language.

A lack of a high-level programming language standard would not only give the user the benefit of a wide variety of languages to choose among, it would also be to the advantage of the computer makers: there is certainly more money in being the first to introduce a SNOBOL4 than in being the hundred and first to produce a BASIC.

References

1. See, for example, Marc LeBrun, "Tilting at Windmills, Or, What's Wrong with BASIC," *People's Computer Company* vol. 2 no. 1 (December 1972) p. 5.
2. John McCallum, "The Altair (S-100) Bus Forum: PCC 77," *BYTE* vol. 3 no. 3 (March 1978) pp. 148-151.
3. Robert Suding, "Why Wait? Build a *FAST* Cassette Interface," *BYTE* vol. 1 no. 11 (July 1976) p. 46.
4. Glen A. Taylor, "Language Development: A Proposal," *BYTE* vo.2 no. 11 (November 1977) pp. 190-191.
5. Peter Skye, "The 8080 High Level Language Project of Peter Skye, Continued," *BYTE* vol. 2 no. 5 (May 1977) pp. 68-70.
6. Carl Helmers, "Is PASCAL the Next BASIC?" *BYTE* vol. 2 no. 12 (December 1977) pp. 6-8 and 184-185.
7. Leigh Janes, "Reactions to Previous Comments," *BYTE* vol. 3 no. 2 (February 1978) p. 159.

*Trademark of Rockford Research, Inc.

The Heath H-8 System

BY TOM WILLIAMS
Editor

The Heathkit H-8 computer system has been on the market for over a year now and since introducing the basic system, Heath has been gradually adding accessories to its product line. Somewhat more controversy than is usual for the introduction of "another 8080-based system" surrounded the coming of the H-8. Now that most of the dust has settled, however, and Heath has followed up with the introduction of a floppy diskette system, it seems a good time for a calmer look at the H-8 and its present and future peripherals.

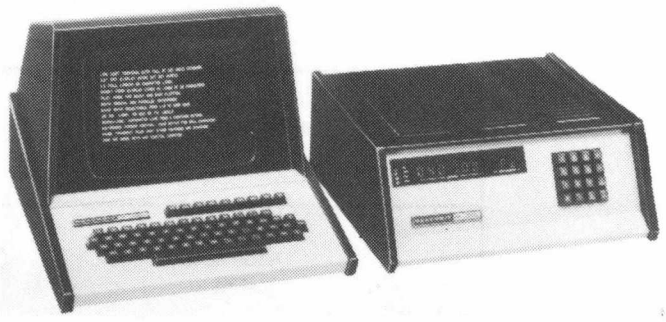
Since the microcomputer market is so vast and varied, any evaluation must, in addition to a critique of hardware and software, take into account the portion of the market a manufacturer appears to be aiming at and the apparent suitability of that product. Since this is the sort of exercise that makes or breaks executive careers, I will not suppose that my judgments are the "last word" on the subject.

The H-8 is an unconventional looking 8080 system and gives evidence of having been designed from the "ground up" without prejudice by previous designs. The two most striking features are the front panel display and keypad, the arrangement of the mother board with its Benton Harbor Bus . . . and the lack of a cooling fan.

The front panel display allows the user to examine (and alter by means of the front panel keypad) the contents of memory locations, and registers and to address I/O ports. The display is in split octal which is unfortunate since the majority of published microcomputer software is in hexadecimal. This is not so bad as long as one is working *within* octal, but refering to other material published in hex and then converting can be annoying.

The mother board, which is mounted in the right side of the cabinet instead of on the bottom as in many S-100 systems, incorporates Heath's own bus design, the Benton Harbor Bus. The introduction of this new bus design has been one of the more controversial aspects of the H-8. It is a 50 pin bus with all but seven pins permanently defined. And Heath has designated those pins it reserves the right to change in future products. Heath claims cost and engineering considerations as the main factors influencing the introduction of the new bus structure as well as the fact that the S–100 bus was created before the development of a system controller. One of the main considerations, according to Heath, was to insure that signals which could possibly interfere with each other were not on adjacent bus lines. A detailed discussion of Heath's new bus by Tom Yeager, chief computer engineer for Heath, can be found in the August 1977 issue of *Interface Age.*

Interference and impedance considerations also play a role in the discussion of an extender board. Heath recommends working on boards in a bus slot by moving the board in question to the rear of the cabinet and removing enough other boards to make room. This is not the most convenient arrangement and definitely awkward in the case of the CPU board which is connected to the front panel display board by a short three-wire harness. You certainly can't get a logic probe down between them and this leaves the scary alternative of probing about on the foil side keeping track of all the pins in reverse order. Nonetheless, Heath has discouraged use of an extender card for servicing because of the possibility of its changing the transmission characteristics of the bus. The one attempt by an independent manufacturer to make an extender card was an awkward affair and the instructions told the user to steady the board with one hand while working on it.



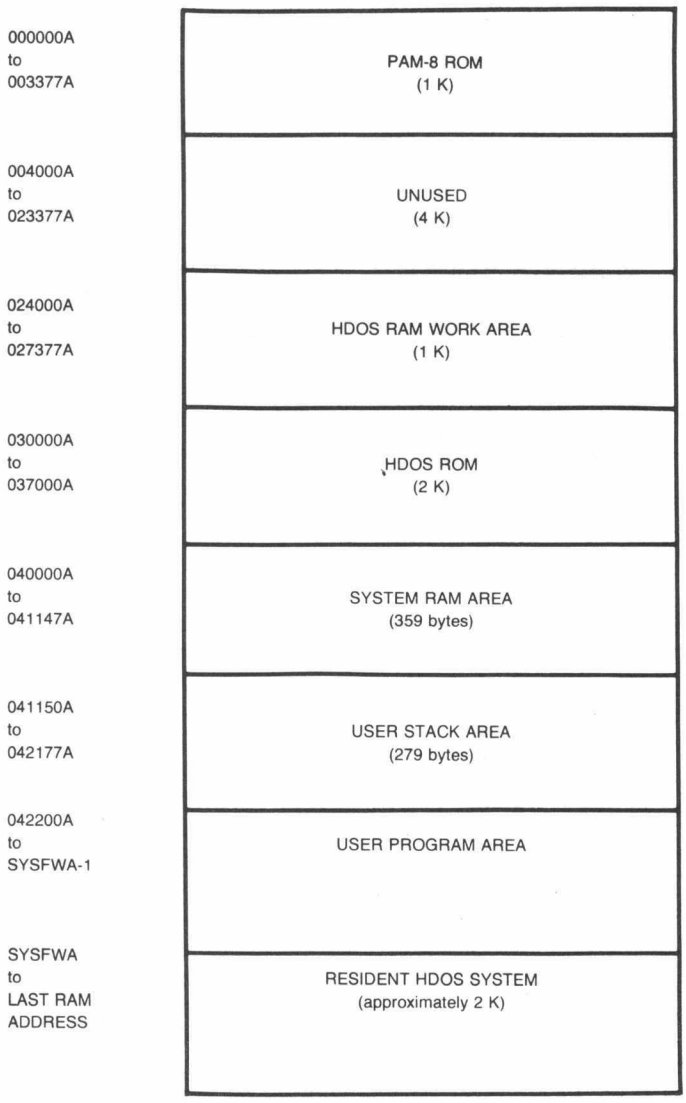
H8 System with H9 Video Terminal

All this aside, if we grant the fact that the Benton Harbor Bus is on balance an engineering and cost advance over the S-100 design (or lack of it), we must still recognize that there are powerful factors which would tend to deter a manufacturer from introducing an improved design. The most obvious of these is that there are no accessories being manufactured for that bus at the time it is first put on the market. No company can come out with everything at once. I have heard from Heath that they are planning many more goodies, but the fact is that at the moment you just can't get a modem or video graphics board for the H-8. In fact, the only serious independently produced product for the Heath bus to date is the Godbout 12K RAM board. The lack of such accessories can act as a deterrent to the potential customer, and the fact that Heath nonetheless decided to market a new bus design speaks for their confidence in their product.

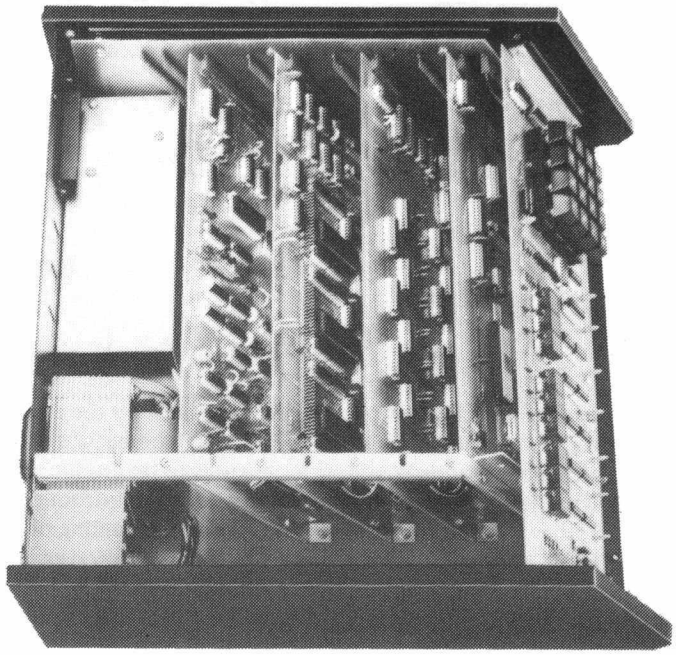
One of the most striking features of the mainframe, and one which attracted the immediate attention of the computer

music people here at People's Computer Company, is the absence of a cooling fan. Considerable design effort seems to have gone into producing a machine which could run without an AC motor in the immediate vicinity. The cabinet is vented on top and bottom and the voltage regulator ICs on the various boards are mounted on aluminum heat sink brackets at the end of each board. These brackets would be inadequate heat sinks in themselves, but they bolt to the bottom of the chassis and to a tie bracket at the top which in turn is attached to the metal cabinet. They thus provide for cooling by means of a combination of convection and conduction to the mass of the unit. This bracket on the unit operating with three 8K RAM boards, disk controller and serial I/O board was rather warm to the touch but still seemed equal to the task.

The mother board which is mounted in the right side of the cabinet, comes with connectors for seven boards in addition to the CPU and front panel with the rearmost connector reserved for external expansion. The power supply consists of a +8 Volt, 10 Amp supply (regulated to +5VDC on each board) and a ±12 volt supply at 500 mA to provide the +12 and -5 Volts required for the CPU – enough for ten boards averaging 750 to 800 mA each.



Memory Map Courtesy Heath Co.



Heathkit H8 Digital Computer Mainframe

Ideas for expansion have been incorporated into the original design. In addition to the rearmost mother board connector, memory addresses have been reserved for various expansion devices. The front panel ROM monitor occupies the first 1K of memory space. An additional 7K are reserved. The disk controller ROM and RAM, for instance, are mapped into this reserved area with plenty of room left over for other things. The fact that the monitor and this much reserved memory space are located at the bottom of memory means that a great deal of 8080 software, which requires RAM starting at the very bottom of memory cannot be used with the H-8 system. There are, of course, ways to get around this and it is possible to disable the front panel ROM.

Heath is currently marketing two memory boards – a 4K (expandable to 8K) and a 16K board – and two I/O boards. The I/O options currently consist of the serial/cassette I/O board with one serial port and cassette interface, and the three port parallel board. Heath has just begun selling a four port serial card now available in assembled form for $250. These I/O cards all have one interesting component in common: the 8251 USART.

All Heath software uses the same console driver and in the serial board, the 8251 is used to take the parallel data from the data buffer (or put it on as the case may be) and convert it to serial data. On the parallel card, the serial output of the USART is run into a UART and converted back into parallel data. This technique allows the use of the same console driver since in either case the CPU is "looking" at the 8251.

The Terminal

The H-9 terminal, while a good value for the price of $550, is the weakest part of the system. It provides a 12 inch CRT displaying 12 lines of 80 characters – upper case only. The display is crisp and solid, but users may find the 12 line limit somewhat inconvenient. A very useful feature, however, is the short form display mode. This switch-selectable option displays information as four 20 character columns, each 12

lines long, and is excellent for displaying long lists of numerical or similar data. In the short form mode, the display scrolls off to the left when the screen is full.

A third display mode, the plot mode, will display a dash at a proportional distance above the plot information code (an ASCII character) which is repeated four times on four lines (?). This is a rather bizarre way of presenting the information and I have yet to find a practical use for the plot feature. It may have its advantages, but it remains aesthetically displeasing.

The H-9 has a serial port (RS232), which is usually connected to the H-8, and also a parallel port. The parallel port might be a way of circumventing the need for a parallel I/O card in some cases or as a printer port. I have used it as a means of inputting lower case characters from a separate keyboard after converting the H-9 to display lower case (*DDJ* Vol. 3 No. 28, p. 30).

The keyboard of the H-9 is adequate but a trifle inconvenient. The keys are all the same size and color and the RETURN key is located right next to the SCROLL and LINE FEED keys. This has caused some aggravation with illegal character error messages and the like.

Software

Heath supplies a full set of software on cassette with the H-8 mainframe. This includes a console debugger (BUG-8), a text editor (TED-8), an assembler (HASL-8), and Benton Harbor BASIC. There is also a cassette-based extended version of BASIC available for $20 which includes string functions and file capabilities.

Benton Harbor BASIC is a good value for the price, but it does not have some of the added features of some other versions on the market. The extended version occupies about 9.2K and incorporates some rather novel features while omitting some of the editing and renumbering capabilities found in other BASICs.

BH BASIC represents numbers internally in 6 digit floating point. I did notice that raising a number to a power (even an integer) does not give a precise result because BASIC executes this command by multiplying the natural logarithm and then raising e to that power. Thus: (X↑Y) = EXP(Y*LOG(X)). Some of the unique features include using a Control B to execute a designated GOSUB from the keyboard while a program is running. Other control functions give access to the front panel display (for monitoring a register or memory location) and control of the real time clock.

Some of the extended BASIC formats are:

Internal line format:

2 bytes: line numbers
1 byte: carriage return (new line)
1 byte each for command and following space
1 byte each for character in the line which is not covered by the above, except that spaces, line number and start of text are ignored.

Number	=	6 bytes
Array	=	6 bytes + 2*(no. of dimensions) + 4*(no. of elements)
String	=	6 bytes + (2 bytes + no. of characters)
String Array	=	6 bytes + 2*(no. of dimensions) + 4*(no. of elements) + [2*(no. of elements) + no. of characters].

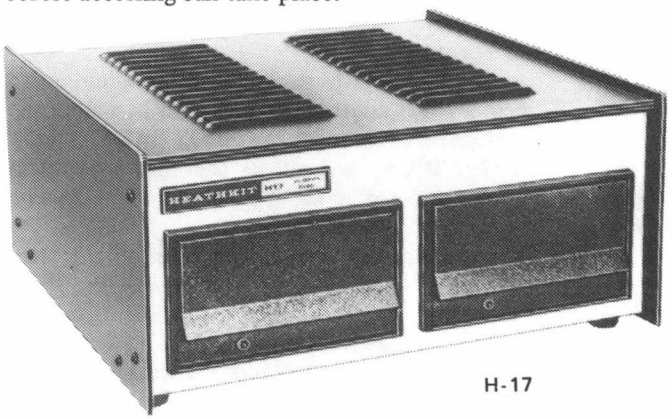
New Products

Heath people are reluctant to discuss new products until they are almost realities. Their reasoning is entirely commendable, given the sorry history of the industry on this point in the past, but it does run athwart my function as a journalist. Therefore, I shall report those things Heath has admitted to definitely having on the way and then the wild, unsubstantiated rumors. The real things include two software products: a diskette version of Microsoft BASIC and a disk version of UCSD Pascal, both to be available in March of 1979. Also, a dot matrix impact printer was shown at the Philadelphia show in August and should be available in late December or January. Projected price, $895 assembled, $595 in kit form. There are rumors that Lifeboat Associates of New York was interested in adapting CP/M to run on the H-8, and Tony Gold of Lifeboat Associates told me they were interested in just that, but had not yet gotten the necessary technical information from Heath. The author of SAM 76 (see *DDJ* No. 21, Jan, 1978) reports a similar problem.

HDOS/Diskette System

The Heath H-17 diskette system is based around the Wangco model 82 drive unit. Heath sells a cabinet which will accomodate two drive units and there appears to be no provision for expansion to additional drives. We have seen the wired version and the kit is due to be available in the near future. The kit assembly should be fairly simple, consisting of little more than assembling the controller board, power supply and cabinet.

The drive uses a stepper motor and worm gear to position the head. The disk is hard sectored and organized into 40 tracks, 10 sectors each for a total of 102.4K bytes of storage. Heath appears to be quite proud of the access time of its disk drives, but is careful to caution the user not to confuse track seek time with random sector access – the true time it takes to get to the data. Both drive units I used performed reliably at a track to track seek time of 8 msec. This means an *average* track seek time of 104 msec. Add to this a rotational latency of 100 msec. (the average time it take a sector on a given track to come under the head) and we have a random sector access time of 204 msec. The HDOS and controller make use of the motor on feature of the model 82 drive to extend motor life. If no commands are sent to the drive for a certain length of time, the spindle motor shuts off. When required, the spindle motor turns on and takes one second to come up to speed before accessing can take place.



H-17

HDOS appears to be a rather sophisticated diskette operating system and can best be described as consisting of 2 major portions of interest to the user: the system diagnostic and optimization portions and the general operation portions. These distinctions are my own. The main diagnostic programs in HDOS are contained in TEST17 which starts out by checking the drive's rotational speed and displaying it on the H-8's front panel LEDs. If the rotational speed does not fall within the parameters required, the user can make a small adjustment with a trimpot on the drive's circuit board.

The next test is a general read/write test on each sector to test the head and seek mechanism. This is an exhaustive test making three complete passes and requiring fully 40 minutes to complete. Any problems are displayed as either "hard" or "soft" errors and the user is referred to the media check or to the troubleshooting section of the manual. A media check then performs a test on a diskette to determine if there are any bad sectors. The results tell how many bad sectors were located and lists the individual sectors. During the initialization procedure for that diskette, the user can enter the bad sectors and HDOS will ignore them while using the remaining good ones.

The final test checks the track to track seek time of the individual drive units. HDOS makes its first attempt at 18 msec. It then displays the highest speed at which the unit will seek reliably. This information can then be used during system optimization to set the seek time for the drive.

The SYSGEN portion of HDOS allows the user to copy the minimal operating system onto another diskette to use as a system volume so that the original can be safely stored away. To this minimal system can be added whichever other programs from the distribution diskette the user wishes. Thus, several system volumes with various portions of the distribution software can be created, such as a diskette with BASIC, one with the editor and assembler, and so on. The minimal system contains among other things, SYSGEN itself, PIP, the peripheral interchange program that allows copying a file from one drive or peripheral device to another, and SET for optimizing the system. With this latter program the user can set the track seek time to the optimum determined in the diagnostic sequence, set the console or alternate terminal to backspace and set the proper number of stop bits. STATUS displays disk statistics including the number of reads and writes and, more importantly, the number of hard and soft errors encountered. An error that occurs 10 times in succession is classified as a hard error.

The distribution diskette sells for $100 and contains disk versions of all the software supplied with the H-8: BUG-8, HASL-8, TED-8 and BASIC. In addition, there are a number of programs specifically oriented toward the operation of the disk system. These include:

BASCON This allows the user to read a BASIC program into a buffer in memory and convert that program as a file onto the diskette which will run under the HDOS BASIC.

TXTCON Like BASCON, this allows the user to convert cassette text files created under TED-8 into text files on the disk.

ONECOPY This is a program for copying files in a system with only one drive unit. ONECOPY is one of the more fun utility programs in that it reads a portion of the file from the source disk into a buffer in RAM. The size of this buffer is dependent on the amount of memory in the H-8. It then displays the word "destination" on the console as well as the characters DES on the front panel LEDs. The user then switches diskettes, hits RETURN and that portion of the file is written onto the new diskette. The system then displays the word "SOURCE" on the terminal and Sou on the front panel, instructing the user to switch back to the source diskette for another portion of the file to be copied. This process continues until the entire file is copied.

Documentation and Service

If there is any one area where Heath outshines all comers, it is in documentation and customer service. Heath's assembly manuals are of a uniform quality—as they are in all Heath products—containing illustrated, detailed, step by step instructions with interim checks and voltage/resistance tests. But the documentation goes far beyond this.

The operation manuals contain full schematics, timing diagrams, options for configuring I/O, and detailed troubleshooting flowcharts. The one criticism is that the schematics contain only Heath part numbers which must be referenced to the manuals to find out what the actual IC number is. It seems that this is an inconvenience it would have been so easy to avoid. Also included in the operation manuals for the H-8 and H-9 are circuit descriptions and general theory of operation sections with timing diagrams and drawings which allow the interested user to study the details of his system to the depth of his interest.

The software reference manual contains a detailed description of all the software, with many examples of the unique features. These are not courses in programming, nor are they meant to be, but user reference material. Also included are listings of several BASIC utility routines, including the console driver, as well as listings of the front panel monitor and disk controller ROMS.

Customer service is available through the factory or through any one of 50 Heath Electronic Centers around the country. Each has factory-trained people specializing in different Heath product lines. For those who do not live near a Heathkit store, the company maintains a telephone number in Benton Harbor, Michigan with technical advisers on hand to answer desperate questions from the hinterlands. I have checked out this number several times and can report some very positive experiences.

In early January, while I was constructing the H-8 itself, the line would often be busy. If it was, the woman who answered when I finally got through asked how many times I had tried and found the line busy. By Spring they seemed to have gotten their traffic problems ironed out and there is little trouble getting ahold of a technician. The man at the other end has a system in front of him and his answers to my sometimes manufactured (and occasionally genuine) questions were always quick and clear. What you get is either a direct answer or a list of steps for isolating the problem such as swapping or substituting various ICs. To customers who have long since resigned themselves to futile letters and the general eral runaround, this kind of treatment is nothing less than heavenly and puts real substance behind the Heath motto, "We won't let you fail."

BASEX: A Fast, Compact Interactive Compiler for Microcomputers

BY PAUL K. WARME
423 Kemmerer Road
State College, PA 16801

BASEX is a new intermediate-level language for microcomputers that combines some of the best features of both BASIC and EXecutable machine language code. It is almost as easy to use as BASIC and yet, it is nearly as fast and versatile as assembler language. The BASEX compiler is *interactive*; that is, it permits you to enter, list, edit and run your program without the help of any auxiliary programs, such as editors or linkage editors. In this respect, BASEX is more like BASIC than FORTRAN, PASCAL and other compilers. Most BASEX commands resemble their counterparts in BASIC language, and this facilitates learning the language and translating programs from BASIC into BASEX.

The main advantages of BASEX are its speed (typically 7 to 20 times faster than BASIC) and the compact size of BASEX programs (typically at least 6K bytes smaller than similar programs written in BASIC). This difference in size is due to the fact that the BASEX compiler does not have to be resident during execution and the run-time subroutines occupy only 2K bytes of memory. However, since the BASEX compiler and run-time routines together use only 8K of memory, it is generally more convenient to leave the compiler in memory while debugging your BASEX programs. A microcomputer with 16K bytes of memory is adequate to make full use of the power of BASEX. Since the BASEX compiler is able to decompile its own object code very rapidly, you only need to save one copy of the program (the object code). This is a distinct advantage over FORTRAN and other compilers, which save separate source, object and executable versions of the same program.

The Origins of BASEX

About a year ago, I was having a lot of fun with my IMSAI 8080 microcomputer until I tried some BASIC programs to draw pictures on my video terminal and play electronic music. Unfortunately, BASIC was too slow to permit smooth motion of video images and accurate timing of musical notes. After buying a chess program written in BASIC, I found that it took about ten minutes to make a move. The Bulls and Cows game (similar to Mastermind) in *101 Basic Games* often took 10 to 30 minutes to make a move, much to my consternation. In short, I discovered that there is a rather large class of problems that demand greater execution speed than BASIC can deliver. Unfortunately, many of the things that I wanted to do, such as interactive graphics, electronic music and word processing, fall in this category.

At the time, it seemed that programming in assembler language was the logical alternative to BASIC, although I dreaded the thought of writing large programs (>8K) in assembler language. When I started translating one of these large programs into assembler language, it was quite discouraging. I had to reinvent the wheel at every turn, writing routines to input and output numbers and text, multiply, divide, etc. Then I noticed that certain operations were done repeatedly in different parts of the program and could be called as subroutines, thus avoiding repetitious coding. After thinking about this, it occurred to me that most of these same subroutines would be needed in just about any large assembler language program.

From this point, it was easy to conclude that assembler language programming would be much easier and faster if all of these useful subroutines could be made available in any program. A method was needed to pass arguments to the subroutines; the most straightforward way of doing this would be to insert the addresses of the arguments immediately after the subroutine call. This approach requires that a region of memory be set aside for a *symbol table*; that is, a table containing the names of variables and their present values.

Pulling on the Bootstraps

The next step was to write a set of subroutines that would be generally useful in almost any program and then write a compiler in BASIC that would translate three-letter mnemonic names of each subroutine into a CALL to the corresponding subroutine. The compiler also had to copy each new variable name included in the user program into the symbol table and leave a two-byte space for the value associated with that name. Once the compiler was working in BASIC, I used it to translate the compiler into BASEX (in other words, a series of CALL's to BASEX subroutines). If this sounds circular, well, it is.

I now had a compiler written in BASEX which worked about 20 times faster than the compiler written in BASIC. The speed advantage of BASEX was already paying off! In order to put some finishing touches on the BASEX compiler, the original version was copied to a different part of memory and the original compiler was used to modify the relocated copy. Incidentally, writing a compiler in a new language is a good test of its versatility and power. In fact, I decided to add a few more subroutines to the run-time package during the process of translating the compiler into BASEX.

An Overview of BASEX Program Commands

Table I summarizes the commands available to BASEX users. Notice that all of the command mnemonics are three letters long and the data types include 16-bit signed constants and variables (-32,767 to 32, 767), one-dimensioned arrays of up to 255 values, string variables of up to 255 characters and string constants of any length. One unusual feature of BASEX array names is that the "(" is treated as part of the name, much like the "$" at the end of a string name. Thus, the closing parenthesis,), is not needed after the element. For example, DATA(1 means the first element of the array called DATA(. Variable names may contain any number of charac-

ters (or numbers), which makes BASEX programs more descriptive than BASIC programs. Since variable names are saved only once (in the symbol table), the memory cost of using longer, more meaningful names is slight.

Pseudo-Operations:

The BASEX pseudo-operations are used only by the compiler; they are skipped over during execution of the program. The REM command is for comments, which are set off by exclamation marks. The DIM command tells the compiler how much space to save in the symbol table for arrays and strings. The *** command marks an entry point in the program which can be addressed by the GTO, JMP and CAL commands. This feature of using names instead of numbers for entry points adds to the readability of BASEX programs. The END command indicates the end of a procedure or program.

Arithmetic and Logical Commands:

All of the arithmetic and logical commands place their result in the accumulator, which is referred to by the letter A (this is the only reserved variable name). The accumulator, A, is a 16-bit variable that can be used just as any other BASEX variable. The sign of the most recent arithmetic or logical operation is retained for conditional branching via the JMP command. The arithmetic commands allow addition (ADD), subtraction (SBT), multiplication (MLT) or division (DIV) of 16-bit signed constants or variables. Commands are also available to determine the ABSolute value (ABS), INCrement a value by one (INC) or DECrement by one (DEC). The logical commands (AND, NOT, IOR, XOR) operate simultaneously on all bits of a 16-bit constant or variable.

Load and Store Commands:

BASEX provides several ways to load and store values in memory. The SET command is equivalent to the LET command in BASIC. The DEF (DEFine) command allows multiple SETs to be combined in one command. However, it is more convenient to use the DAT command to assign values to successive elements of an array. The BRD (Byte ReaD) command works like the 8-bit PEEK or EXAM functions in BASIC, while the WRD command gives the BASEX user the added capability of reading 16-bit words anywhere in memory. The complementary functions of storing bytes or words in memory are performed by the BRT (Byte wRiTe) and WRT commands. These commands are analogous to the POKE or FILL commands in BASIC, but it is easier to store multiple bytes or words using BASEX, since multiple values can be stored with a single command.

String Commands:

Basex needs only one very powerful string command, STR, to duplicate the ASC, CHR$, LEFT$, MID$ and RIGHT$ string functions used in some versions of BASIC. Characters can be transferred back and forth between strings and variables (or arrays). Substrings are specified in terms of their first and last character positions in a given string. The CMP command CoMPares the alphabetic order of two strings and allows conditional branching via the JMP command, based on the outcome of the comparison. Other useful string commands are the LEN command to determine the LENgth of a string or substring and the VAL command, which returns the numeric value of a string in the accumulator, A.

Program Control Commands:

One of the most useful program control commands is the JMP command, which branches to a labelled (***) entry point if a particular condition has resulted from the most recent arithmetic, logical or compare command. The conditions that can be tested for are EQual (EQ) to zero, Not Equal (NE), Less Than (LT), Greater Than (GT), Less than or Equal (LE) and Greater than or Equal (GE) to zero. The GTO command can be used for unconditional branching. Structured programming buffs may choose not to use the GTO command, but after all, if you know where you're going, the GTO is the quickest way to get there. However, structured programming techniques can be used with BASEX.

Instead of the FOR. . .NEXT loop structure of BASIC, BASEX uses the FOR and TIL commands. The FOR command sets the initial value of the loop variable, while the TIL command adds or subtracts any value from the loop variable and then checks to see whether the terminating value has been exceeded. If not, the program branches to the statement following the corresponding FOR command. FOR. . .TIL loops can be nested to any depth and there are no restrictions on jumping into or out of loops.

BASEX has a much better facility for calling subroutines than BASIC. For starters, subroutines are called by name, rather than by statement number, so the subroutine name can

```
                    BASEX PROGRAM COMMANDS

X,Y and Z are variables or constants; A( is an array; S$ is a string
"TEXT" is a string constant; variable names can be any length.

>> Pseudo-Operations <<
REM !BASEX! (comments)  DIM A( 1Ø S$ 8Ø (Dimension arrays & strings)
*** START (Program labels)  END (End of program)

>> Arithmetic Commands << (Results are in the Accumulator)
ADD X+Y (add) SBT X-Y (subtract) MLT X*Y (multiply) DIV X/Y (divide)
ABS X (absolute value) INC X (increment) DEC X (decrement)

>> Logical Commands << (Results are in the Accumulator)
AND X&Y (logical AND)   NOT X (complement)
IOR X#Y (Inclusive OR)  XOR X%Y (Exclusive OR)

>> Load and Store Commands <<
SET X=Y or SET A( 1=X or DEF X=Y A( 1=X ... (equate variables)
DAT A( X=Y Z ... (initialize an array)
BRD X (read byte at X)  BRT X=Y Z ... (write bytes from X onward)
WRD X (read word at X)  WRT X=Y Z ... (write words from X onward)

>> String Commands <<
STR X=S$ Y      or STR A( X=S$ Y (string to variable transfer)
STR S$ X X=Y    or STR S$ X X=A( Y (variable to string transfer)
STR S$ X Y=S$ Z or STR S$ X Y="TEXT" (string to string transfer)
CMP S$ X Y-S$ Z or CMP S$ X Y-"TEXT" (compare strings)
LEN S$ X (length of string)  VAL S$ X (numeric value of string)

>> Program Control Commands <<
JMP CC X (jump to X on condition CC = EQ, NE, LT, GT, LE, or GE)
FOR X=Y.......TIL X+Y>Z (loop)     GTO Y (unconditional branch)
CAL X Y Z ... (call subroutine X)  GET Y (pass Y to subroutine)
PUT Z (return Z to caller)         RET (return from subroutine)

>> Input/Output Commands <<
INP X A( Y S$ Z ... (input)        CHR X (print character)
PRT X A( Y S$ Z "TEXT"... (print)  TAB Y (tab to column Y)
XIN X (read input port)   WAT X Y (wait for condition Y on port X)
XOT X (write output port) DSI/ENI (disable/enable interrupts)

>> Memory Search and Move Commands <<
BSR X Y Z (search from X to Y for byte Z)
WSR X Y Z (search from X to Y for word Z)
MOV X Y Z (move memory from X through Y to Z)

                    BASEX COMPILER COMMANDS

LST or LSM (list all or part of the program or symbol table)
DMP or NTR (dump or enter one or more memory values)
INS or DLT (insert or delete one or more program statements)
SIZ (print program size)  LOC (set program pointer)
RUN (run the program)     MON (exit to monitor)
```

reflect its purpose. The first argument after a CAL command is the entry point name, defined elsewhere in your program by a *** command. The optional second and subsequent arguments specify variables to be passed to or returned from the

subroutine. Within the subroutine, you can use the GET command to receive a value from the calling program or use the PUT command to return a value. Since the variables named in the GET and PUT commands may be different from the names used in the CAL command, you can reserve "local" variable names for use in subroutines. However, any subroutines that are compiled at the same time and share the same symbol table may access any variable without using the GET and PUT commands. This allows "global" variables to be shared by subroutines. The RET command is used to RETurn from a subroutine.

Input/Ouput Commands:

The I/O commands in BASEX make it easy to communicate with terminals and instruments of all types. To customize BASEX for your terminal, you simply fill in the address of your routines to input and output one character, as described in the manual. The INP command INPuts one line of characters or numbers, ending with a carriage return, and stores them in the variables listed after the INP. The PRT command PRinTs a list of variables or constants, separated by spaces. The PRN command does the same, but omits the space between items. You may space over to a particular column by using the TAB command or print a single CHaRacter with the CHR command.

If you wish to communicate directly with I/O ports, you can INput from port X with the XIN command or OuTput to port X with the XOT command. In both cases, the byte to be communicated is placed in the accumulator, A. The WAT command WAiTs until the input port specified by its first argument gives a non-zero value when ANDed with its second argument. You can also ENable or DiSable Interrupts with the ENI and DSI commands.

Memory Search and Move Commands:

Since the block memory search and move commands are extremely fast, they are indispensible for writing compilers, editors and other programs where speed is important. The BSR command SeaRches for a particular Byte (8 bits) within a specified region of memory, while the WSR command SeaRches for Words (16 bits). If the specified byte or word is found, its address is placed in the accumulator, A, and the condition flags are set to the Less Than (LT) condition so that a JMP command can test whether the search was successful. If the requested byte or word is not found, the Greater Than (GT) condition is flagged. The MOV command MOVes a group of bytes from one region of memory to another. It gives correct results even when the new region of memory overlaps the old region.

Using the BASEX Compiler

The interactive BASEX compiler makes it easy to write and debug programs. Each time BASEX is restarted, it prints "RANGE?" and expects you to enter four numbers to designate the first and last addresses of your program and the first and last addresses of its symbol table. Let's call these addresses PRG1, PRG2, SYM1 and SYM2, respectively. BASEX works with decimal numbers, since most people think most readily in base 10. If your BASEX program is already loaded in memory and you have previously declared its memory space, you can type a 0 after the "RANGE?" prompt, and then a carriage return (here denoted <CR>). In this case, BASEX will print the current values of PRG1, PRG2, SYM1 and SYM2, just as a reminder.

If you want to write a new BASEX program, you should respond to the "RANGE?" prompt by typing the starting address of the program (PRG1), a space, and then type the same value for PRG2, followed by a space. The space is the preferred separator between numeric arguments, although any non-numeric character will do. Now, you should type the address of the last byte in memory that you want to set aside for the new program, a space, and then type the same number once more, followed by <CR>. This sets SYM1 and SYM2 to the top of memory because the symbol table builds downward from the top of memory. The BASEX compiler will now erase any program that previously existed between PRG2 and SYM1.

Whether you are starting a new program or revising an old one, the value of PRG2 is next printed, followed by a question mark. The "?" always signals that a BASEX program (the compiler, in this case) is waiting for user input. You can now type any three-letter BASEX command mnemonic, a list of arguments and then <CR>. The BASEX compiler checks to see whether your command line begins with a compiler command (discussed below) and if so, performs the requested action and then retypes the value of PRG2 and "?" once more.

If your command line contains a *program* command, the appropriate CALL to a BASEX run-time subroutine is inserted in your program, followed by the symbol table addresses of any arguments specified in the command line. If any new variable names occur in the argument list, they are copied into the symbol table. Next, the compiler updates PRG2 to the address of the next available program location, prints it and waits for another command line after the "?" prompt. Notice how the automatic line number feature of BASEX facilitiates program entry.

BASEX Compiler Commands

The BASEX compiler commands allow the user to list and modify programs. The SIZ command prints the values of PRG1, PRG2, SYM1 and SYM2 in the following format:

```
PROGRAM ##### #####  SYMBOLS #####
    #####
```

The LOC command allows you to set PRG2 to any position in your program, so that the next program command will be inserted there. If you want to LiST your entire program, you can type LST; if you type one number after LST, the single command at that location will be listed or, if you type two numbers after the LST command, all program commands between those limits will be listed. The LSM command is used to LiSt the entire SyMbol table and one or two arguments may follow the LSM to list selected parts of the symbol table. As each variable name is listed, its location in the symbol table and the current value of that variable are also printed. One useful debugging technique is to insert an END command at some point in your program and then, when your program halts, list all or part of the symbol table to determine the current values of your variables. You can now alter the values of certain variables, list the program or modify the program. In most cases, you can still resume your program after the END command without harmful side-effects.

The INS and DLT compiler commands are used to INSert or DeLeTe program lines. Both require two arguments, which specify the first and last memory locations to be inserted or

deleted. These locations can be determined from the program listing. The INS command moves the last part of the program to a higher place in memory, prints the range of the relocated program segment, sets PRG2 to the first location to be inserted, prints PRG2 and then waits after the "?" prompt for one or more new program commands to be entered. When you finish entering new commands, you should use the DLT command to delete the unused locations up to the reloacted program segment. In cases where the altered program will be the same size or shorter than the original one, you can use the LOC command (instead of the INS command) to set PRG2 to any location in your program, enter the changes, and then delete any unwanted commands with the DLT command.

The DMP command is available for printing the decimal values stored between any two memory addresses that you specify. A carriage return and line feed are printed automatically whenever the address is evenly divisible by ten. The NTR command can be used to eNTeR one or more decimal values into sequential memory bytes, starting at the memory address specified by its first argument. When you want to RUN your program, you merely type the RUN command, followed by the desired starting address. Since your program has already been compiled as you entered it, there is no delay before your program begins. The address given after RUN can point to the first byte of any command in your program or any other program. You can restart BASEX at any time by typing RUN 0 or just RUN. After your BASEX program finishes, it will jump to location 0 and restart BASEX.

The MON command allows you to exit conveniently to your system monitor. In view of the great diversity of mass storage devices now available, BASEX contains no mass storage handlers. However, you can use your system monitor to save and load BASEX programs or you can even use BASEX to write a tape handler, as will be shown by one of the examples coming up.

BASEX Error Messages

In order to maintain fast program execution and small program size, BASEX is somewhat weak on error checking. Nevertheless, it is a rare thing for a BASEX program to run wild, because after each BASEX command is completed, a check is made to be sure that the next byte in the program is a CALL instruction. Any other instruction will cause the program to halt, and error dump will be printed and then the BASEX compiler will be restarted. Other run-time errors (of 18 different types) will also halt and print and error dump, consisting of the values of all the registers, the program location where the error was detected, the error type number and the top three words on the stack. In addition, when the BASEX compiler detects any of nine different types of program errors, it prints a two letter error code and ignores that command line.

Relocating LOADER for BASEX Programs

It is frequently desireable to be able to move a BASEX program to a position different from that at which it was compiled. A special program called the LOADER (written in BASEX) is provided for this purpose. Although the standard version of the LOADER overlays the BASEX compiler, it can be used to relocate the compiler or the LOADER itself. The LOADER commands include MOV, to MOVe a program or its symbol table to a new location, PRG to modify the PRoGram and SYM to modify the SYMbol table to take into account its new location in memory. The DMP, NTR, RUN and MON commands described above are also available in the LOADER.

The FIX command is the most powerful LOADER command; it moves the program to a new location, removes all pseudo-operations (REM, DIM, and *** labels) and moves the symbol table to directly follow the program. As each variable name is moved, its new location is printed and the name is abbreviated to its last letter to conserve space. Comments are also omitted to make the program smaller. After the FIX operation, the program can no longer be listed or changed by the BASEX compiler, nor can it be relocated by the LOADER. This provides a measure of protection for proprietary software.

EXAMPLE: Memory Test Program

Let's write a simple BASEX program to write every possible 8-bit byte in every memory location within a selected region of memory and then read it back to verify its correctness. Of course, any errors should be reported by giving the erroneous value, its location and the correct value. In order to compare BASEX with BASIC, we'll do it first in Altair BASIC, version 3.2.

```
10 REM MEMORY TEST IN BASIC
100 INPUT "FIRST AND LAST BYTES";FIRST,LAST
110 FOR I=0 TO 255
120 FOR J=FIRST TO LAST
130 POKE J,I
140 IF PEEK(J)=I GOTO 160
150 PRINT "BYTE",J,"WAS",PEEK(J),
    ";SHOULD BE",I
160 NEXT J
170 NEXT I
180 PRINT "DONE"
OK
RUN
FIRST AND LAST BYTES? 16501,16600
DONE

OK
```

In BASIC, this program took 250 seconds to test 100 memory bytes (25,600 write, read and compare operations). Now, let's translate this program into BASEX and observe how similar the two programs are.

```
*GO BASEX
RANGE ? 11000 11000 11500 11500
11000 ? REM !MEMORY TEST IN BASEX!
11006 ? PRT "FIRST AND LAST BYTES"
11012 ? INP FIRST LAST
11020 ? FOR I=0
11028 ? FOR J=FIRST
11036 ? BRT J=I
11044 ? BRD J
11050 ? SBT A-I
11058 ? JMP EQ OK
11066 ? BRD J
11072 ? PRT "BYTE" J "WAS" A ";SHOULD BE" I
11088 ? CHR 13
11094 ? *** OK
11100 ? TIL J+1>LAST
11112 ? TIL I+1>255
11124 ? PRT "DONE"
11130 ? CHR 13
```

```
11136 ? SIZ
PROGRAM 11000 11136     SYMBOLS 11387 11500
11136 ? LSM
11387 DONE" 11387
11395 ;SHOULD BE" 11395
11409 WAS" 11409
11416 BYTE" 11416
11424 OK 11100
11429 J 0
11433 I 0
11437 LAST 0
11444 FIRST 0
11452 FIRST AND LAST BYTES" 11452
11476 MEMORY TEST IN BASEX! 11476
11136 ? RUN 11000
FIRST AND LAST BYTES ? 16501 16600
DONE
RANGE ? 0
PROGRAM 11000 11136     SYMBOLS 11387 11500
```

The execution time in BASEX is 29 seconds for 100 bytes. This is 8.6 times faster than BASIC, even though this version of BASIC is the fastest one tested by Tom Rugg and Phil Feldman in their article, "Basic Timing Comparisons . . . Information for Speed Freaks," published in *Kilobaud* (June, 1977, p. 66). When I coded their longest program, Benchmark Program 7, in BASEX, it ran in 7.1 seconds, whereas Altair BASIC took 51.8 seconds and other versions of BASIC took up to 235.6 seconds. Apple's 6K integer BASIC took 28.0 seconds; this may be a better comparison with BASEX, since neither of these use floating point arithmetic. However, like all benchmark programs, there is a certain amount of variation in performance, depending on the nature of the program. In all of the cases that I have tested, BASEX runs 7 to 20 times faster than Altair BASIC, Version 3.2.

EXAMPLE: Intel Format Tape Routines with Checksum

The following program reads, writes and verifies tape files in Intel format. It is fast enough to work with my National Multiplex CC-7 data recorder running at 2400 baud (try that with BASIC!). It will also work with paper tape or other flavors of cassette tape. The use of descriptive names for entry points and subroutines makes the program fairly easy to understand.

In Intel format, the first byte of a record is a colon and all subsequent bytes are in ASCII hexadecimal form. Bytes 2 and 3 are the record length, bytes 4 through 7 are the load address and bytes 8 and 9 are the record type (0 here). Next comes the data, two ASCII digits per byte, and then the checksum, which is the negated 8-bit sum of all the digits since the colon. At the end of each record, there is a carriage return and line feed.

The verify operation is the same as the read operation, except that the values read from the tape are just compared to the contents of memory. Subroutine INCHAR reads one ASCII digit and OUTCHAR writes one digit on the tape. Both of these routines must be customized for the user's I/O ports. Subroutines READCHK and WRTCHK read and write memory bytes as two hex digits and update the checksum, while READHEX and WRTHEX do the same thing without contributing to the checksum. Subroutine HEX converts ASCII hexadecimal digits to binary, whereas HEXOUT does the reverse conversion and goes on to output the result.

The program and symbol table occupy 1,712 bytes and they can be compressed to only 1,097 bytes by means of the FIX command in the LOADER program.

```
PROGRAM 12288 13406     SYMBOLS 13453 14000
13406 ? LST
12288 REM !INTEL FORMAT TAPE ROUTINES!
12294 DIM ANSWER$ 6
12302 DEF TAPE 2 STATUS 3
12314 *** BEGIN
12320 CHR 13
12326 PRT "READ/WRITE/VERIFY/STOP"
12332 INP ANSWER$ 1
12340 CMP ANSWER$ 1 1 "R"
12352 JMP EQ READ
12360 CMP ANSWER$ 1 1 "W"
12372 JMP EQ WRITE
12380 CMP ANSWER$ 1 1 "V"
12392 JMP EQ VERIFY
12400 GTO 0
12406 *** VERIFY
12412 SET VRFLAG=1
12420 GTO READ1
12426 *** READ
12432 SET VRFLAG=0
12440 REM !READ NEXT BLOCK!
12446 *** READ1
12452 SET CHKSUM=0
12460 CAL INCHAR
12466 SBT A-58
12474 JMP NE READ1
12482 CAL READCHK
12488 SBT C-0
12496 JMP EQ BEGIN
12504 SET NBYTES=C
12512 CAL READCHK
12518 MLT C*256
12526 SET ADDRESS=A
12534 CAL READCHK
12540 ADD C+ADDRESS
12548 SET ADDRESS=A
12556 CAL READCHK
12562 FOR BYTE=1
12570 CAL READCHK
12576 SBT VRFLAG-0
12584 JMP EQ READ2
12592 BRD ADDRESS
12598 SBT C-A
12606 JMP EQ READ3
12614 PRT "V" "ERROR AT" ADDRESS
12624 GTO BEGIN
12630 *** READ2
12636 BRT ADDRESS=C
12644 *** READ3
12650 INC ADDRESS
12656 TIL BYTE+1 NBYTES
12668 CAL READHEX
12674 ADD A+CHKSUM
12682 AND A&255
12690 JMP EQ READ1
12698 PRT "R" "ERROR AT" ADDRESS
12708 GTO BEGIN
12714 REM !READ WITH CHECKSUM!
12720 *** READCHK
12726 CAL READHEX
12732 SET C=A
12740 ADD C+CHKSUM
12748 SET CHKSUM=A
```

```
12756 RET
12760 REM !READ 2 HEX DIGITS!
12766 *** READHEX
12772 CAL INCHAR
12778 CAL HEX
12784 MLT A*16
12792 SET C=A
12800 CAL INCHAR
12806 CAL HEX
12812 ADD A+C
12820 RET
12824 REM !CONVERT ASCII TO HEX!
12830 *** HEX
12836 SBT A-58
12844 JMP LT HEX1
12852 SBT A-7
12860 *** HEX1
12866 ADD A+10
12874 RET
12878 *** INCHAR
12884 WAT STATUS 2
12892 XIN TAPE
12898 AND A&127
12906 RET
12910 REM !WRITE TAPE!
12916 *** WRITE
12922 PRT "WRITE RANGE"
12928 INP FIRST LAST
12936 SET A=0
12944 FOR BYTE=1
12952 CAL OUTCHAR
12958 TIL BYTE+1 10
12970 REM !WRITE NEXT BLOCK!
12976 *** WRITE1
12982 SET NBYTES=16
12990 SBT LAST-FIRST
12998 JMP GE WRITE2
13006 REM !TERMINATOR!
13012 SET A=58
13020 CAL OUTCHAR
13026 SET A=48
13034 CAL OUTCHAR
13040 CAL OUTCHAR
13046 GTO BEGIN
13052 *** WRITE2
13058 SBT A-15
13066 JMP GE WRITE3
13074 ADD A+16
13082 SET NBYTES=A
13090 *** WRITE3
13096 SET CHKSUM=0
13104 SET A=13
13112 CAL OUTCHAR
13118 SET A=10
13126 CAL OUTCHAR
13132 SET A=58
13140 CAL OUTCHAR
13146 CAL WRTCHK NBYTES
13154 DIV FIRST/256
13162 CAL WRTCHK A
13170 CAL WRTCHK FIRST
13178 CAL WRTCHK 0
13186 FOR BYTE=1
13194 BRD FIRST
13200 CAL WRTCHK A
13208 INC FIRST
13214 TIL BYTE+1 NBYTES
13226 SBT 0-CHKSUM
13234 AND A&255
13242 SET C=A
13250 CAL WRTHEX
13256 GTO WRITE1
13262 REM !WRITE WITH CHECKSUM!
13268 *** WRTCHK
13274 GET C
13280 ADD C+CHKSUM
13288 SET CHKSUM=A
13296 REM !WRITE 2 HEX DIGITS!
13302 *** WRTHEX
13308 DIV C/16
13316 CAL HEXOUT
13322 SET A=C
13330 *** HEXOUT
13336 AND A&15
13344 SBT A-10
13352 JMP LT ASCII
13360 ADD A+7
13368 *** ASCII
13374 ADD A+58
13382 *** OUTCHAR
13388 WAT STATUS 1
13396 XOT TAPE
13402 RET
13406 END 13406 ? LSM
13453 ASCII 13374
13461 HEXOUT 13336
13470 WRITE 2 HEX DIGITS! 13470
13492 WRITE WITH CHECKSUM! 13492
13515 WRTHEX 13308
13524 WRTCHK 13274
13533 WRITE3 13096
13542 TERMINATOR! 13542
13556 WRITE2 13058
13565 WRITE1 12982
13574 WRITE NEXT BLOCK! 13574
13594 OUTCHAR 13388
13604 LAST 16100
13611 FIRST 16101
13619 WRITE RANGE" 13619
13634 WRITE TAPE! 13634
13648 HEX1 12866
13655 CONVERT ASCII TO HEX! 13655
13679 HEX 12836
13685 READ 2 HEX DIGITS! 13685
13706 READ WITH CHECKSUM! 13706
13728 READHEX 12772
13738 ERROR AT" 13738
13750 READ3 12650
13758 READ2 12636
13766 BYTE 6
13773 ADDRESS 16101
13783 256 256
13789 NBYTES 5
13798 C 0
13802 READCHK 12726
13812 INCHAR 12884
13821 CHKSUM 0
13830 READ NEXT BLOCK! 13830
13849 READ1 12452
13857 VRFLAG 1
13866 VERIFY 12412
13875 V" 13875
13880 WRITE 12922
13888 W" 13888
13893 READ 12432
13900 R" 13900
13905 READ/WRITE/VERIFY/STOP" 13905
13931 BEGIN 12320
```

(cont'd on page 39)

TINY GRAFIX... FOR TINY BASIC

BY JOEL SWANK
4655 SW 142nd No. 186
Beaverton, OR 97005

Tiny GRAFIX is a system for graphic display on small micro computer systems. It is composed of a SWTPC GT-6144 TV graphics board, Tom Pittman's Tiny BASIC, and some machine language subroutines. It runs on a KIM-1 with 4k of memory.

The GT-6144 divides the TV screen into 6144 rectangles in a configuration of 64 horizontally (X) and 96 vertically (Y). It has its own 6144 bit memory used to maintain the display. This memory is altered by data from the computer. The data consists of X coordinates, Y coordinates and control codes. Data is received via one eight-bit output port with data ready strobe. The GT-6144 has no facility to send data to the CPU. The GT-6144 is connected to KIM by the B port of a 6820 PIA.

Communication with the GT-6144 is handled by a set of machine langauge subroutines located at $200. They are designed to be used with the USR function of Tiny BASIC. The following functions are provided: BLANK – Initialize the PIA and the GT-6144 and clear the screen; ON – Turn on spot at coordinates X,Y; OFF – Turn off spot at X,Y; FILL – Turn on all spots on the screen; REVERSE – Reverse the screen; CNTL – Send 1 byte of data to the GT-6144.

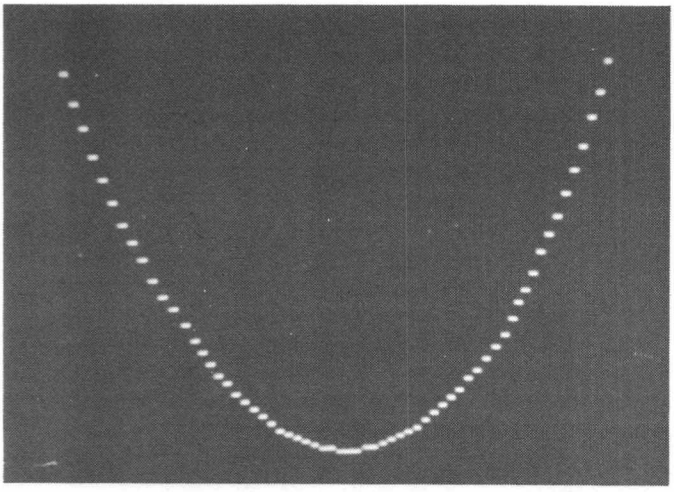
The USR function passes data from Tiny BASIC's 16-bit signed integer variables to these subroutines. The calling syntax is shown in the comments of the assembly listing. Only the low order byte of each parameter is used by the subroutines. Subroutines ON and OFF do minimal checking of the data for validity. With the CNTL subroutine a BASIC program can send any value (0-255) to the GT-6144.

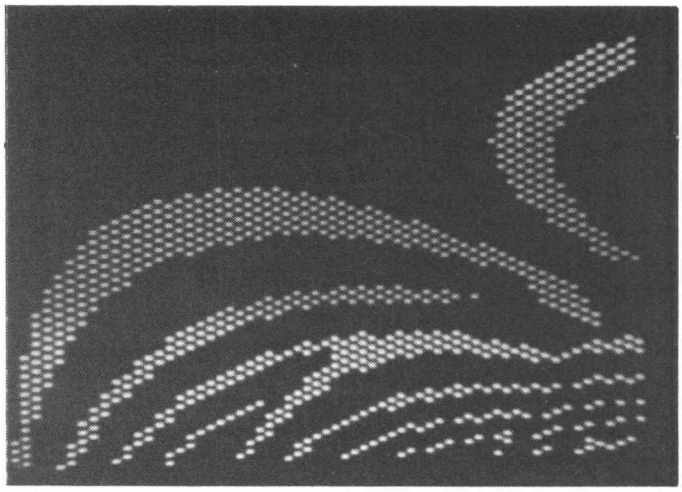
Tiny Graffer is a simple BASIC program that will graph an equation of the form Y=f(X), where f(X) is any valid Tiny BASIC operation. This equation is entered between lines 8000 and 8200. One value of Y is computed for each value of X. Tiny Graffer requires 4 input parameters to scale the graph on the screen. They are start value for X, increment for X, low of Y range, and high of Y range. A good value for the equation Y=X*X is –64, 2, –100, 4000. This results in picture 1. There is also the facility to do negative graphs (black line on white background). This is done by changing line 1020 to P=579. Usually in 3 or 4 attempts input values can be found that will produce a centered graph.

Tiny Pattern Generator takes advantage of the wraparound nature of Tiny BASIC's 16-bit signed integer variables to do computer art. An equation of the form U=f(X, Y) is used to control the pattern. The program computes U for every other spot, and then based on present criteria either turns it on or off. The variables H and L define the range for turning on spots. A good range for most functions is L=500 and H=3000. Very small numbers also turn on spots. H & L should be set before the program is run. Every other spot is used instead of every spot to speed up generation (Tiny BASIC is pretty slow), and to provide a more pleasing display. Many interesting patterns can be developed with this routine. Pictures on this page and the next are examples.

One problem with the GT-6144 is hard copy. About the only way to get hard copy is to take a picture. To take a picture the shutter of the camera must stay open for at least 1 raster scan of the TV (1/60 second). Instead of shelling out for a camera with variable shutter speed it is possible to use a Kodak instamatic camera (the cheapest one). The shutter speed of this camera is not directly adjustable and the normal speed is less than 1/60 second, but when a flash stick is inserted the shutter speed slows to allow for less light. So I cut the plug-in out of a used flash stick and plugged it into the camera. This slowed the shutter speed enough to allow the camera to capture a full raster scan.

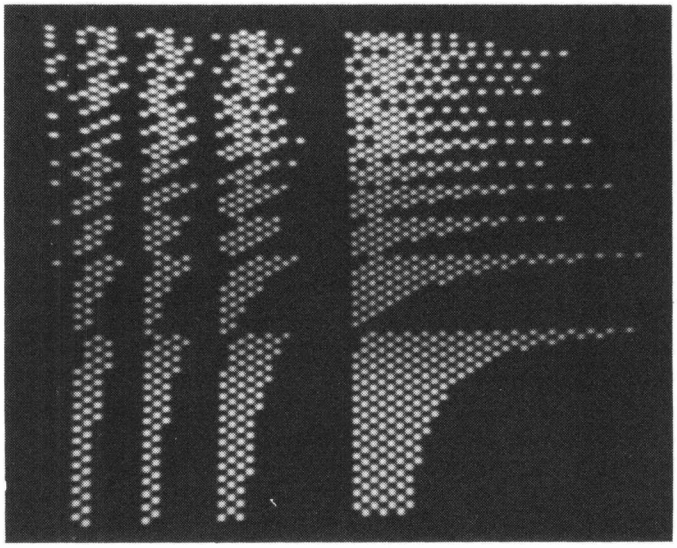
Because of its speed, Tiny BASIC can only do static graphics. These two programs only scratch the surface of what can be done with static graphics. With a few line or figure drawing subroutines (in BASIC) and some imagination, sophisticated graphic displays can be produced by Tiny BASIC.





```
0010:                    CSP : GRAFIC SUBROUTINE PACKAGE
0020:                          TINY BASIC INTERFACE VERSION
0030:
0040:                    FUNCTION :
0050:                        TO PROVIDE AN INTERFACE FOR THE SWTPC
0060:                        6144 TV GRAPHICS BOARD. THE BOARD IS
0070:                        CONNECTED VIA THE B PORT OF A 6820 PIA
0080:                        LOCATED AT MEMORY LOCATIONS D802
0090:                        (DATA REGISTER) AND D803 (CONTROL
0100:                        REGISTER). IT ALSO USES CONTROL BIT
0110:                        2 FOR DATA OUT PULSE.
0120:
0130:
0140:                        TINY BASIC CALLS
0150:
0160:                    CLNK : INIT PIA AND CLEAR SCREEN
0170:                        P=USR(535)
0180:
0190:                    ON   : TURN ON SPOT X,Y
0200:                        P=USR(577,Y,X)
0210:
0220:                    OFF  : TURN OFF SPOT X,Y
0230:                        P=USR(579,Y,X)
0240:
0250:                    FILL : TURN ON ENTIRE SCREEN
0260:                        P=USR(609)
0270:
0280:                    RVRS : REVERSE THE SCREEN
0290:                        P=USR(596)
0300:
0310:                    CNTL : SEND X TO 6144
0320:                        P=USR(518,X)
0330:
0340: 0200                    ORG   $0200
0350:
0360:                    ZERO PAGE STORAGE
0370:
0380: 0200               REVR  *     $00E0
0390:
0400:
0410:
0420:                    EXTERNAL LABELS
0430:
0440: 0200               DATB  *     $D802
0450: 0200               CTLB  *     $D803
0460: 0200               GETK  *     $1F6A
0470:
0480:
0490:
0500:                    TVOUT : SUBROUTINE TO SEND DATA FROM
0510:                            X & Y REGISTERS.
0520:
0530: 0200 8E 02 D8 TVOT  STX   DATB   SEND X
0540: 0203 20 09 02       JSR   TOGG   PULSE
0550:                     SEND 1 BYTE
0560: 0206 8C 02 D8 CNTL  STY   DATB   SEND Y
0570:                     TOGGLE : SEND PULSE ON CONTROL BIT 2
0580: 0209 A9 34    TOGG  LDAIM $34
0590: 020B 8D 03 D8       STA   CTLB   CONTROL LINE LOW
0600: 020E EA             NOP
0610: 020F EA             NOP          DELAY SOME
0620: 0210 EA             NOP
0630: 0211 A9 3C          LDAIM $3C
0640: 0213 8D 03 D8       STA   CTLB   CONTROL LINE HI
0650: 0216 60             RTS
0660:
0670:                    CLANK : CLEAR ENTIRE SCREEN
0680:
0690: 0217 A0 80    CLNK  LDYIM $80    INITIALIZE PORT TO OUTPUT
0700: 0219 A2 00          LDXIM $00
0710: 021B 8E 03 D8       STX   CTLB
0720: 021E A9 FF          LDAIM $FF
0730: 0220 8D 02 D8       STA   DATB
0740: 0223 A9 3C          LDAIM $3C
0750: 0225 8D 03 D8       STA   CTLB
0760: 0228 20 00 02 WRIT  JSR   TVOT   CLEAR A BYTE
0770: 022B C8             INY          NEXT Y
0780: 022C C0 E0          CPYIM $E0    END OF LINE?
0790: 022E D0 F8          BNE   WRIT   NO
0800: 0230 A0 80          LDYIM $80    RE-INIT
0810: 0232 E8             INX          NEXT X
0820: 0233 E0 40          CPXIM $40    END OF LINE?
0830: 0235 D0 F1          BNE   WRIT   NO
0840: 0237 A2 E1          LDXIM $E1    NORMAL SCREEN
0850: 0239 86 E0          STXZ  REVR   SAVE STATUS
0860: 023B A0 E4          LDYIM $E4    ENABLE GRAPHICS
0870: 023D 20 00 02       JSR   TVOT   SEND
0880: 0240 60             RTS
0890:
0900:
0910:                    TURN ON SPOT
0920:
0930: 0241 09 40    ON    ORAIM $40    ON BIT
0940:                     TURN OFF SPOT
0950: 0243 29 7F    OFF   ANDIM $7F    LIMIT
0960: 0245 AA             TAX
0970: 0246 C0 60          CPYIM $60    Y IN RANGE?
0980: 0248 90 02          BCC   YOK    YES
0990: 024A A0 5F          LDYIM $5F    CLIP
1000: 024C 98       YOK   TYA
1010: 024D 09 80          ORAIM $80    INDICATE Y
1020: 024F A8             TAY
1030: 0250 20 00 02       JSR   TVOT   SEND
1040: 0253 60             RTS
1050:
1060:                    REVERSE : SUBROUTINE TO REVERSE THE SCREEN
1070:
1080: 0254 A5 E0    RVRS  LDAZ  REVR   GET CTL BYTE
1090: 0256 49 01          EORIM $01    TOGGLE BIT 0
1100: 0258 85 E0          STAZ  REVR   SAVE
1110: 025A 8D 02 D8       STA   DATB   SEND
1120: 025D 20 09 02       JSR   TOGG   PULSE
1130: 0260 60             RTS
1140:
```

```
1150:                    FILL : SUBROUTINE TO TURN ON ALL
1160:
1170: 0261 A2 40    FILL  LDXIM $40    INIT COORDS
1180: 0263 A0 80    NEWY  LDYIM $80
1190: 0265 20 00 02 WRT   JSR   TVOT   SEND THEM
1200: 0268 C8             INY          NEXT Y
1210: 0269 C0 E0          CPYIM $E0    END OF COLUMN?
1220: 026B D0 F8          BNE   WRT    NO
1230: 026D E8             INX          NEXT X
1240: 026E E0 80          CPXIM $80    DUN?
1250: 0270 D0 F1          BNE   NEWY
1260: 0272 60             RTS
TS-

-T

    SYMBOL TABLE 3200 3260
    CLNK   0217    CNTL   0206    CTLB   D803    DATB   D802
    FILL   0261    GETK   1F6A    NEWY   0263    OFF    0243
    ON     0241    REVR   00E0    RVRS   0254    TOGG   0209
    TVOT   0200    WRIT   0228    WRT    0265    YOK    024C

:LIST

1000 REM   TINY PATTERN GENERATOR
1010 A=USR(535)
1020 Y=0
1030 X=Y-(Y/2*2)
1040 GOSUB 8000
1050 IF U<-12000 A=USR(577,Y,X)
1060 IF U>L IF U<H A=USR(577,Y,X)
1070 X=X+2
1080 IF X<64 GOTO 1040
1085 Y=Y+1
1090 IF Y<96 GOTO 1030
1100 PR "BYE"
1110 END
8000 REM THE PATTERN GENERATION FUNCTION GOES HERE
8100 U=X*X*X/7-Y*Y/9+X*Y-X*X/8+Y*Y*Y/13
8200 RETURN

:LIST

1000 REM       TINY   GRAFFER
1010 REM
1015 REM SET P=577 FOR POSATIVE GRAPH
1016 REM SET P=579 FOR NEGATIVE GRAPH
1020 P=577
1025 REM INPUT  X-START, X-INC, Y-LOW, Y-HIGH
1030 INPUT S,T,U,V
1035 REM CLEAR SCREEN
1040 A=USR(535)
1045 REM FILL SCREEN IF NEGATIVE GRAPH
1050 IF P=579 A=USR(609)
1055 REM COMPUTE Y  INCREMENT
1060 N=(V-U)/95
1070 X=S
1080 L=0
1085 REM COMPUTE VALUE
1090 GOSUB 8000
1095 REM SCALE RESULT
1100 Y=(N/2-U+Y)/N-1
1105 REM CLIP AT EDGE OF DISPLAY
1110 IF Y<0 GOTO 1150
1120 IF Y>95 GOTO 1150
1125 REM DISPLAY VALUE
1130 Y=95-Y
1140 A=USR(P,Y,L)
1150 X=X+T
1160 L=L+1
1165 REM NEXT X
1170 IF L<64 GOTO 1090
1180 PR "BYE"
1190 END
8000 REM  THE EQUATION TO BE GRAFFED GOES HERE
8100 Y=X*X
8200 RETURN
```



The STD Bus:

A Critical Look at a Proposed New Standard

BY KEITH BRITTON
P.O. Box 6214
Hayward, CA

Recently Mostek and Prolog jointly announced a new bus. Such announcements appear from time to time, generating close scrutiny from a handful of intensely interested parties, mild curiosity from a somewhat larger and mostly hardware oriented group and, from most of the rest of us, no more than dull resentment at the need to increment our files of dubious mnemonics and ill chosen acronyms. The STD bus may well repay somewhat closer attention, for a surprisingly wide variety of reasons. STD is an abbreviation of the word standard and the principal importance of the bus lies in its proponents' attempt to create a widely adopted standard rather than to merely launch a product line. Few observers will fail to find something of interest in the resulting interplay of politics, economics, history and tradition (distinction intended), mechanics, engineering decisions and tradeoffs, software/firmware/hardware interactions, logic, individual and corporate personalities. And, for those addicted to private or professional meditation, there is not only room for consideration of the immediate and future technological impact of the new bus but also of some subtle sociological implications as well.

The Bus – General

A bus design is probably best considered top down. That is to say from the general logical content through the specific logical meaning of signals to their electrical and mechanical expression. As proposed, the STD bus is a simple synchronous bus with an eight bit data path separate from a sixteen bit address path. The necessary control signals are also simple and distributed one per bus line. Thus the overall philosophy of the bus is to accept limitations while gaining simplicity, but there is also an attempt to fit the bus to a variety of eight bit processors and to make token provision for memory and I/O expansion. The read/write controversy is resolved on the separate line side, an exception to the general air of frugality.

The economy of lines (56) used by the STD bus is not only a distinguishing feature but also a significant selling point. Every line which must be supported exacts its economic toll in bus drivers and buffers and in printed circuit board area for both motherboard and cards. This may not be a disadvantage for a complex high unit value computer, nor for highly flexible hobby/development systems as in the S-100 bus. However, for the more price-conscious industrial control/light data processing market at which the STD is aimed, the fewer pins the better. Perhaps even more important, for equipment aimed at industrial environments, is the reduced difficulty experienced in maintaining reliable bus operation and reduced power requirements, especially where line termination is employed.

As a practical matter, the number of lines also determines the pin spacing and contact areas, which also impact economics and reliability. Contact pins on .156 inch centers, as in the common 44 pin standard sockets, allow the design of cheap but excellent motherboards. Signal traces may be heavy, and therefore of low impedance, and there is plenty of room between signal traces for ground line shielding. Power supply is unlikely to be a problem thanks to the substantial contact area of the power pins. A motherboard capable of reliably handling state-of-the-art data rates can thus be produced as a simple double sided PC board and manufactured by a marginal board house. In contrast, a comparable motherboard for connectors in .1 inch spacing is a work of art and expensive. The .125 inch spacing used for the STD bus represents an engineering compromise between these extremes. As such it offers some rewards for good design but threatens significant penalties for bad.

Card Size and Shape

STD bus advertising stresses the card size, and for good reason. A system using many small cards, as used by Control Logic Corp., incurs significant expense in connectors and buffers. Increase in system size and complexity leads to awkwardly shaped card cages, and motherboards so long as to present formidable difficulties for high speed data. Systems with comparatively large cards, such as the SBC 80 and LSI 11, economize on connectors, buffers and bus lengths but at the cost of increased card complexity and expense. In addition, enclosures for large cards are generally expensive and may be mechanically weak. Large cards are both inherently more difficult to support and more difficulty to thermally control. STD cards are 4½ inches wide and, as standard, 6½ inches long. Supported along their long sides, they are well adapted for use in rigorous environments. They also represent a compromise in terms of card area, which make them well suited to their market. System maintainability is critically dependent upon module size, as is first cost, though to a lesser extent. Large, complex multi- function cards significantly impact first cost unless all functions are used, and impact maintainability in that failure of any card function is likely to result in complete card replacement. STD cards will be typically single logical function, encouraging board swapping trouble shooting, reducing spares inventory, and keeping initial system cost aligned to functional requirements.

The small work area of the STD card produces some problems however. While it can certainly accept as many as 48 chips for a dense arrayed card such as a memory board,

even excellent printed circuit layout will not allow densities equivalent to much more than 30 chips for more randomly arranged circuits. For this reason, skill in printed circuit board design will be at a premium for this bus. Note, however, that this applies only to the standard STD card size. Unlike other buses, the STD has the potential for being comparatively flexible in its card area. The 4½ inch width of the card fits a very wide variety of standard enclosures, and while the most popular is certainly the 6½ inch length, other lengths are supported most notably the 9.6 inch. This latter offers roughly double the effective work area, that is the total area less buffer area. Even longer cards, though possible, are unlikely as the resulting card cages become mechanically poor and circuit board layouts become increasingly inefficient due to the increased aspect ratio.

An interesting comparison may be made between the STD standard card and its probable extension and the Eurocard system. The small Eurocard is roughly 4 inches by 6 inches (100 x 160 mm). Unlike the STD card, it is fractionally too narrow for two 40 pin dip chips to be aligned crosswise, a surprisingly significant layout disadvantage. Its effective work area is only some two-thirds that of the STD cards which, combined with its less advantageous aspect ratio, will in practice nearly double the number of small Eurocards compared to STD cards required for an equivalent system. Since increasing the area of the Eurocard by increasing its length would incur very substantial aspect ratio layout inefficiencies, the Eurocard area expansion is done by increasing the width, resulting in a disastrous economic disadvantage for small systems. The larger cards require two connectors rather than one, use different motherboards, different supporting chassis, and large and small boards cannnot be intermixed in a single system. If the STD bus does in fact become the standard for industrial electronics within the United States, this comparative flexibility may prove to be a significant, perhaps even crushing, advantage over its European competition. And with the advance of the metric system within the US, existing cards will fit a 115 millimeter card guide spacing with 175 millmeter and 250 millimeter card lengths.

Power

The principal logic power supply is 5 volts supplied with off-card regulation to avoid local heat concentration. Two bussed pin pairs are provided at one card edge to allow both ample contact area and room for motherboard traces of unlimited area. At the opposite end of the card single opposed pins supply a nominally analog plus and minus 12 volts. This supplies, though with limited contact area, the 24 volts likely to be needed for industrial controls. Power is more likely to be contact area limited than trace area limited since the pins concerned are the outboard pins on the connector, allowing unlimited trace room. The ground return for these pins, which need not be logic ground, is more limited physically, but this is partly compensated for by use of two bussed pins. Both sides of the board may therefore be used with heavy traces and, if desired, extra area may be used between sockets at the expenses of the power lines to reduce impedance.

Up to this point, the power supply and distribution can be considered beyond reproach. The remaining power supply blows two more precious pins on minus 5 volts. This eyebrow raiser becomes even more curious when one realizes that the pins are specifically not bussed and that at least one of the bus proponents does not propose to supply this power level for standard systems. This curiosity comes about for partly historical and partly corporate policy reasons. A draft of the bus dated in May shows the two pins concerned as undefined. Corporate policy then required that no pins be left unassigned and it seemed possible that minus 5 volts could prove a useful substrate voltage, particularly for use with dynamic memories. Although this possibility later receded, no one could think of anything useful to do with the pins so both were assigned to an unwanted power level, the primary function of which is to discourage ultimate use of the pins concerned, leading to the pointed question—What *is* the STD bus standard? Either an STD bus designer has a right to expect an STD compatible supply to supply this power level, or the STD bus may well be headed down the road pioneered so well by the S-100.

Bus Lines—Address/Data

As proposed, the STD bus has eight data lines and 16 address lines. Data lines are bidirectional and all lines are three state. This is the classic arrangement for a synchronous eight bit processor and is unsurprising since the Prolog customer base buys four bit and eight bit processors and Mostek, which was unable to second source the Zilog Z8000, offers no processor larger than eight bit. The absence of any system for multiplex use of the bus and/or wider data paths is highly regrettable and the biggest reason why this bus should not, as it stands, become a national standard.

Multiplex operation is becoming increasingly common and for good reason. For memory parts, multiplexing roughly halves the number of pins required, significantly reducing direct costs and dramatically reducing the printed circuit board area required and this is of great importance to small cards used in the STD bus. Similar advantages, though less marked, are realized for CPU and other support devices, and this has led to multiplexing, originally considered only applicable to 16 bits and up, to extend downwards into the 8 bit field, notably with the support line for Intels 8085. There is, therefore, a good argument for multiplex capability on the STD bus even if the bus is to be considered restricted to 8 bit operations.

To allow synchronous and multiplex cards to coexist within a single machine would require only two lines to be used for multiplex operation control. Obvious candidates would be the minus 5 power lines. Adding this capability would greatly increase the flexibility of the bus since multiplex processors could still use synchronous cards by demultiplexing at the processor card buffers. In addition, the addition of multiplex capability would allow the use of wider data paths—up to 24 bit. At first glance this may seem unnecessary or even undesirable for small systems and systems particularly designed for industrial control use. It must, however, be realized that PDP 11s, Novas and even more powerful processors have long been commonly used for industrial applications for which 8 bit machines prove inadequate. A national standard based on the STD bus could perfectly well support the Motorola 68000, the Intel 8086 and Zilog Z8000 as well as a wide variety of 8 bit processors. This would allow the user to maintain a high degree of compatibility amongst his most common cards, which will always be predominantly 8 bit I/O cards.

Memory I/O Control 31-36

Four of the six lines are quite straightforward, separate read and write lines plus two lines for memory or I/O select. The remaining two lines are designated for memory and I/O expansion, one line for each. Enquiry, however, reveals that although Prolog and Mostek are in accord as to the designation of these lines, neither has as yet formulated a scheme for actually implementing expansion. This provides another can of worms, since the implementation chosen will impact software, firmware and hardware. Expansion capability may be used in two ways. The most obvious is to simply extend the number of I/O devices or memory locations which may be addressed. Less obvious, but for engineering applications possibly more significant, is the ability to use alternate memory locations or I/O devices. This is best illustrated by considering the latter.

A subroutine using absolute I/O addresses may, given alternate bank capability, be applied to more than one set of I/O devices by simply swapping banks, with implications for both hardware and software. Memory management, for processors not designed for page mode operation, is more of a problem. One means of implementation would be to treat the memory expansion line as the 17th bit to be decoded on each memory card. This would required each memory card to be burdened with the necessary decoding logic, a small but significant expense and penalty in precious PCB real estate. A second system might be to AND with memory request or the enable from the address decode, which has similar disadvantages. Another system is to allow optional jumpering on each memory card for memory request and to issue memory request upon the appropriate line at the processor. Similar considerations apply to I/O cards. This latter scheme involves negligible expense or real estate penalty, other than on the processor card, and has the advantage that since all four lines are conveniently grouped, a simple jumper or switch arrangement allows any desired mix of memory I/O expansion. This latter flexibility would have one disadvantage. Fewer devices could be controlled in this manner than could be controlled using the two expansion lines as address extensions. But memory and I/O cards are not the only ones to be considered. Extended memory or I/O could be managed from the processor card or from a separate management card addressed by the processor via I/O instructions.

Actual implementation for memory management is not as simple as it might appear at first glance. The CPUs to be used were not designed for such duty, and it may entail some curious and amusing oddities in bit hardware and software. Consider the case of a CPU which is swapping memory pages by issuing an I/O instruction. The next instruction fetch will be from the "new" memory page, and from whatever byte the preceding I/O instruction left the program counter pointing at. Transfer between pages of ROM can thus require some care in coding while transfer between pages of writable memory, assuming one has succeeded in achieving an initial beachhead, could add a novel dimension to self-modifying code. Equally amusing, although unattractive, is the thought of dribbling data back and forth across boundaries, hand carried in registers.

This nonsense can be alleviated by processor card logic which arbitrates read, write and instruction fetch between designated memory banks. Other methods are also tenable, but again, in the absence of a defined standard covering logical approach, addressed board hardware and software techniques, and management device address, it is hardly probable that any two manufacturers for the new bus will achieve mutual system compatability.

The Control Group 37-46

Six of the ten signals are straightforward implementations of logical requirements—bus request and acknowledge to allow multiprocessing or DMA, maskable and nonmaskable interrupt request with interrupt acknowledge allowing one level of interrupt priority but requiring vectoring for further interrupt expansion, and wait request for slow devices. The remaining four lines are more questionable from the point of view of a national standard.

Refresh is a frankly Z80 signal, though Prolog documentation states that it may be supplied from a specific purpose card. Standardization upon a processor dependent signal, or mimicry thereof, is at best questionable, but there is worse in store. Pin 38 turns out not only to be defined in terms of specific hardware, but also in terms of the manufacturer's specific product line, and mutually incompatible products to boot. Prolog defines this pin as machine cycle sync for the convenience of the bus analyzer it sells (note that this is NOT the sync associated with an instruction fetch). Mostek calls the line by a different name and uses it as a hardware single step controller. In justice, it has to be admitted that this is not inconsistent with the general tone of this proposed standard. Paraphrased: "Manufacturers shall adhere to a universal standard except that they shall note in their literature where they deviate therefrom." In this instance it is unclear whose definition is *the* STD bus definition, but there is a good argument that neither should be. Nor is this the only signpost pointing to the road pioneered by the S-100.

At Wescon Prolog showed its 7000 series STD cards while Mostek showed its STD-Z80 cards. The Z80 has no separate pin for interrupt acknowledge, the logical signal being produced by an AND of the instruction fetch sync signal, M1, and I/O request. Both companies issued this logical signal from the CPU card to the appropriate bus line. The I/O card usage differed, however. Z80 peripheral support chips requiring an interrupt acknowledge are designed to decode directly from M1 and I/O request. Mostek cards reportedly did just that, ignoring the bus line intended for the purpose, which was the line usd by Prolog I/O cards. Prolog I/O cards will therefore work with Mostek processors and vise-versa, but this is only true for Z80 processor boards and Prolog proposes to offer 6800 and 8085 boards almost immedately. Caveat emptor.

The remaining two signals of this group are titled CPU status 0 and 1. CPU status 1 is used for any signal identifying an instruction fetch, M1 for the Z80. The remaining line, unused by Mostek, is used by Prolog for S0 for the 8085 and VMA for the 6800. In general the clarity, precision and completeness of definition for these lines—again bearing in mind that the companies concerned are proposing the STD bus as a national standard—is perhaps best illustrated by these words from the Prolog draft of the bus manual. "The status control lines are used to provide timing information related to special machine cycle operations. Where it is specifically available status 1 is considered to be a signal to identify instruction fetch."

Reset

All computer systems need a reset line. The STD bus has two. One is a short, clean, debounced signal for use by TTL logic. The other blows one of the few precious lines available merely to communicate an external push button closure to whichever bus resident card supplies the former signal. Power-on reset and debounced external push button reset can perfectly well be handled on the motherboard, and/or at the switch, saving a line and/or a slot and/or CPU card real estate. If a second reset line is to be provided, for engineering purposes it may be better employed as an addressed reset, which would allow the processor to reset a card by issuing the reset and the card's I/O address. This would avoid the necessity of issuing a general system reset, and, since it would reuse the address decode already available, avoid issuing multiple reset commands or committing additional I/O addresses for the sole purpose of implementing local reset functions.

Clock 49 and 50

The STD bus has tow lines for clock signals. One is the processor clock and the other is available for alternate system use. This will allow slow speed devices to use a common clock, a simple and effective arrangement.

God Is On My Right 51 and 52

The STD bus is unusual in defining two pins for daisy chain operation. These pins are not bussed, but connected 51 to 52 along the motherboard. Implementation of daisy chains is therefore greatly simplified since adjacent cards may control each other and priority is simply established by physically positioning the highest priority card to the right of the next highest and so on. This will prove a great convenience for I/O operations, interrupt driven applications, and possibly for implementation of multiprocessors.

An Opportunity

The STD bus, as presently proposed, is a competently designed synchronous bus for eight bit processors unusually well suited to its putative market—engineering and light data processing. But incompatibilities exist even amongst the earliest cards, bus definitions are often fuzzy (and at least one case mutually incompatible) and no apparent consideration has been given to a support or policing system capable of establishing discipline amongst nominal bus suppliers. Based on the above, a reasonable prognosis for the bus would appear to be islands of very good engineering in a shambles comparable to the S-100. Nevertheless, the potential exists for the STD bus to become a well controlled system of complex building blocks of truly extraordinary versatility—the 7400 series of the microcomputer world. This can reasonably be achieved by a hardware change, a policy change and an organizational development.

The hardware change is the definition of two bus lines for control of multiplex operations, preferably by junking the minus 5 volt power. This would open the bus to the new 16 and 32 bit processors and would allow use of the new generation of 8 bit support devices.

It has been the policy of the original STD bus proponents to define all lines of the bus in what is already exposed as a futile attempt to avoid anarchy. Fortunately, this untenable policy is capable of simplification and rationalization. If the bus be considered from a top down design stance, it will be immediately obvious that certain signals should be expressed within the bus architecture because they are logically necessary or extremely desirable for computer operation. In this category fall the address and data lines, read, write, memory and I/O select, bus request maskable and nonmaskable interrupt (with acknowledgements), wait and reset. These signals should be supplied and used, to the exclusion of alternatives, for their logical purpose by all suppliers to the bus. Of lesser importance is a second category of contenders for bus space, commonly used and highly desirable conveniences. In this category fall the processor clock, instruction fetch identification and daisy chain system.

The remaining single lines on the bus, having failed to justify their inclusion in either general category, should be considered as utility lines and defined as such. This is no bar, of course, to multiple definition. Line 37 could, for instance, be defined as refresh/utility. Thus refresh, if implemented, must appear on that pin and no other. The physical implications for this approach are trivial, merely that to accommodate the dual or multiple use potential of such lines, no permanent connection be made to the pins concerned. This requirement may be simply met by use of a jumper or a prepared trace separation point, a necked down portion between two pads usable for repair. STD bus designers, and more importantly users of same, could thus enjoy a high degree of flexibility while maintaining, for the bus lines of primary importance, standards of such rigidity as to preclude basis system incompatibilities.

No standards wll be observed without discipline. To apply it, the STD bus needs a watchdog. Industry watchdogs tend to be toothless, but in this case the situation is unusual. The teeth are obvious but the dog is not. For the market at which the STD bus is aimed, adequate commercial coercion can be simply applied by withholding use of a trademarked logo for products which do not meet established standards. Hobbyists will buy an off brand article but industrial purchasing agents will not. Proof—when did you last see a non-UL approved appliance in an office? There are a number of obvious possibilities for a regulatory body, JEDEC, ANSI, and IEEE to name but a few. None are noted for snappy footwork under pressure, blinding speed in their committee work or evident desire to expose themselves to the muck raking and flak density which may be reasonably expected. The organizations named might, however, be persuaded to nominate board members to an association of STD bus manufacturers set up for standards development and maintenance.

Such organizations are not without precedent and much might be adapted from the commissioner oriented forms of organization developed to police national sports. In addition to its regulatory function, such an organization could perform an invaluable service to manufacturer and customer alike as a central information source for the bus. Available information could include second sources, plug compatible sources, functional equivalents and near equivalent alternatives. Incompatibilities and cautions could be flagged, as could existence of application notes or other useful references. In addition, cross

referencing could be done to sources of related items such as STD width enclosures and solid state relays and transducers. And this information could be easily made available on a 24-hour basis for the price of a microprocessor, a disk and an answer modem.

Futurebus

The IEEE has committees working to straighten out the S-100, rationalize microprocessor bus interfacing and, significantly, to study, anticipate (and hopefully in some measure ure to direct) future developments. Since we may be absolutely certain that the STD bus is not the ultimate bus, it may be well to consider its probable lifespan and the changes which may occur to it within the near future. A number of technological trends are already sufficiently well defined to allow extrapolation. At the risk of belaboring the point, an obvious trend is towards multiplexing. It seems reasonable to suppose that the STD bus will have limited scope and/or limited future if it remains synchronous and, since a trend is also fairly obvious toward wider data paths, if it remains 8 bit.

Interface and I/O support chips become ever more complex while simultaneously single chip microcomputers gain in power and richness of built in I/O devices. To some degree these trends are convergent and are likely to impact the STD bus in two ways. Increased chip complexity will reduce demands from printed circuit board real estate, a definite plus for the STD compared to its competitors. But the trend will also be to distributed intelligence and array processing. This will place an increasing premium on bus architecture factors favoring multiprocessor or distributed intelligence use. Since the STD bus was not designed with this in mind, its present architecture could prove to be a blind alley. Presently, however, distributed intelligence and its problems remain but poorly understood.

Trends in memory toward increased speed and density at reduced cost appear unlikely to significantly affect STD machines used for engineering purposes as such applications tend to require little memory. For data processing applications of STD machines, unless multiplexing opens up all 24 address/data bits for address use, the present single designated memory expansion pin will prove totally inadequate to prevent the bus from reaching rapid obsolesence.

The remaining major question mark for the future of the STD bus results from a quite evident trend which has, to date, been relatively unremarked. In the past processors have radiated a wide variety of status and timing signals usually using a pin for each. A move away from this practice is a recurrent theme for the most recently developed CPUs particularly the Intel line. The signals may still be available redundantly at individual pins, but all necessary signals are likely to be recoverable from a small group of encoded status lines. This makes so much sense in terms of reduced pin count, board and bus complexity, as to be clearly the way of the future. Consider the STD bus. Every status or control line emanating from the processor could be replaced with a block of four encoded lines, and there would still be space in the encoding for multiplex control and I/O and memory expansion. True, each card would then be burdened with decoding, but this would merely involve a simple mask ROM with four inputs configured as bus buffers, hardly more expensive than the buffers required for the present arrangement. Presently, the bus proponents intend that it should be a living thing and adapt to technological change. Production lines have a certain inertia as do customers. Probably significant changes could be made within the next two or three months, present commitments being still limited, and hopefully this opportunity will be taken to immediately make changes which cannot be long postponed. Flexibility and change are essential to avoid ossification, but one should bear in mind Börge Langefors' fifth principle—"If the time it takes to implement a change in a system is longer, on the average,than the mean time between changes, then the system is not viable."

Who Will Salute?

Reference to the Oxford English Dictionary reveals three senses for the word "standard." Firstly, a standard is a rallying point. The present burning question in several board rooms is simply—who will climb aboard? Without wide support, the STD bus will not become a standard, nor will it become a significant market place. It may be reasonably expected to attract a substantial number of very small manufacturers because it is the only game in town for the commonly used card size. Many cards, thanks to the bus's philosophy of competent solid simplicity, will be fitted to the bus by mere pinout reassignment. It is possible, however, that Mostek and Prolog may have more difficulty in attracting to the bus comparably substantial full line competition. Implementing the relatively minor modification necessary to allow multiplex and multibyte bus operation could significantly alter this position, particulary since there is no evidence as yet that two giants and one bigee, Motorola, Intel and Zylog, have yet committed to a bus for their wide processors.

A Good Measure?

The second sense of the word standard is that of a measure. Would, or could, the STD bus become a good or bad standard? and further, would the wider implications of such a standard prove generally beneficial or detrimental? As proposed, the STD bus seems far too limited to have the significant impact necessary to generate a market segment standard, let alone a national or international one. On the other hand, as proposed it already contains the bulk of the features likely to be needed for at least the next decode and has no obvious bar to adopting the missing ones.

The IEEE owes a particular debt for its future bus studies to Hermann Schmid, who recently kindly supplied this author with a paper titled "A Top-Down Approach for Monolithic Processor Standardization," containing his current thinking on the subject. His focus has been particularly upon the mid-80's, but the paper exhibits an almost eerie correspondence with the opportunities open to the STD bus and its proponents. For the STD bus, and by extension the segments of industry and humanity directly or indirectly impacted, the future could be now.

Microprocessor building block standardization could profoundly affect the widespread application of artificial intelligence. But this is a more serious business than merely the drastic cost and development timesaving which might be expected to parallel that shown, on smaller scale, by the standardiation of the 7400 TTL series. Microcomputer standardization has to be seen in historical perspective, and a striking

analogy can be made with the British of 200 years ago. America is leading the way in a revolution far more profound than the Industrial Revolution. Where the British, during the Industrial Revolution, extended man's muscle with inanimate power, Americans are today extending man's mind with artificual intelligence.

From the engineering anarchy of the early Industrial Revolution certain standards emerged, some to become, for economic reasons, immutable. For instance, the British bequeathed the world a standard railway gauge of 4' 8½" for no better reason than that a few miles of existing haulage road being adapted for railway use had timbers convenient for that spacing. Pressures for standardization are increasing so rapidly that it seems hardly likely the next decade will not irretrievably commit our descendants. The STD bus will not be the ultimate bus, but it deserves careful consideration and perhaps support if only because of its potential as a prototype or test bed capable of at least enlightening possibly irrevocable technological decisions which we cannot long evade.

A Thing Erect

The third sense of standard is that of a thing erect—which may lead cynics and those familiar with the traditions of the industry to suspect a digit. Neither of the original STD bus proponents claims to be other than a commercial organization and out to make a buck. It is seldom reasonable to criticize manufacturers for introduction of a product line containing features peculiarly adapted to their advantage, but by proposing the STD bus as a standard, Mostek and Prolog have exposed themselves to judgment in more than simply commercial terms. The usage of pin 38 is clearly self-serving and hardly in accord with a spirit of standardization. There is also evidence, particularly related to the minus 5 power supply pins, that the companies concerned have been willing to accept at least questionable engineering decisions, in an unseemly haste to lock up all pins, and create a defacto standard. Interestingly, the smaller company, Prolog, has adopted the more intransigent take-it-or-leave-it stance. Development of a new bus is, or should be, a problem orders of magnitude greater than can be indicated by this somewhat cursory article, and is a subject which has engaged increasing attention from many fine minds. In particular, the IEEE committees studying bus related matters are an obvious resource and reference point. Presently, this author has been unable to trace any attempt by Prolog and Mostek to consult such resources or to apprise them, in a timely manner, of their intentions. A prima facie case thus exists to indict both companies for at least irresponsibility, in the context, it must be emphasized, of the STD bus proposal as a standard.

Judgment on this issue would, however, be premature, since the companies clearly do not yet have their joint act together and their proposed bus may turn out to be more of an asking bid than a fait accompli. If this proves to be the case, history may record substantial indebtedness to them for short-circuiting years of tedious committee maneuvering by developing the small system standard for the 80's to the point where adoption is swiftly possible with but minor changes. And the sense of erectness may well be associated with the names of Mostek and Prolog—for stature that has hitherto been often lacking amongst the industry's leaders.

```
13939 STATUS 3
13948 TAPE 2
13955 ANSWER$ 6 S
13970 INTEL FORMAT TAPE ROUTINES! 13970
13406 ?
```

(cont'd from page 31)

Other Applications of BASEX

The BASEX manual includes a very useful sample program that allows you to convert numbers from any number base between 2 and 36 to any other base and do arithmetic (+, −, *, /) or logical operations (&, #, %) in any number base. This 1K-byte program will do everything the $60 TI Programmer calculator will do and more! I have also converted that chess program and the Bulls and Cows game into BASEX; both run about 10 times faster than they did in BASIC. That factor of 10 is the difference between fun and boredom. BASEX was also ideal for writing a powerful word processor that I call PRO-TYPE (see below). In order to use my North Star disk and Meca tape drives with BASEX programs, I have written some utility routines that permit random or sequential access to disk or tape files, etc. These things have a way of eating up time, so the interactive graphics and electronic music programs haven't yet been translated to BASEX, but someday . . .

Hardware Requirements and Availability

The current version of BASEX was written for the 8080, but it also runs on Z80 or 8085 microcomputers. However, it is fairly straightforward to translate the 2K bytes of run-time subroutines to run on other microcomputers. Since the BASEX compiler and LOADER are written in BASEX, they should work with minor modifications on any other microprocessor, once the run-time subroutines are available. It should also be very easy to translate any user application programs from one dialect of BASEX to another. The conversion for the 6502 is already underway and if there is sufficient interest, BASEX may be made available for other microprocessors, as well. A second limitation of the current version of BASEX is that it resides in the lowest 8K of memory (locations 0 to 5 and 64 to 1FFF), which presently precludes its use with some computers that have ROM down there (e.g., the Radio Shack TRS-80, Heath H-8).

The BASEX manual is being published by Byte Publications, Inc. and will be available from them or from your local microcomputer store. It includes a more detailed explanation of each command, sample programs and fully commented listings of the assembler language run-time subroutines, the compiler and the relocating LOADER program. A full explanation of how the BASEX user can add his own customized commands to BASEX is included in the manual. It also contains the object code for these programs in bar code format. For those of you who want to be the first on your block to try BASEX, I have a limited number of copies of the preliminary, unedited version of the manual available for $10. North Star disks or Meca tapes containing all of the programs are available for $25 from Interactive Microware, Inc., 116 South Pugh Street, State College, PA 16801. Other disk and tape formats will also be available as soon as possible. This company also markets the program that allows BASEX programs to access North Star disk or Meca tape drives and the PRO-TYPE word processor, which is written in BASEX.

COPYING A DISKETTE ON INTEL DOUBLE DENSITY DOS

BY PAUL M. JULICH
Dept. of Electrical Engineering
Louisiana State University
Baton Rouge, LA 70803

The program presented allows one to copy track by track all blocks of diskette regardless of attribute. The program runs considerably faster than the INTEL supplied program using ISIS Commands for copying files. A complete diskette can be copied in 2 minutes, 45 seconds.

An added benefit is that the copied disk is identical with the original. In other words corresponding blocks will contain identical data. This may not be true for disks copied file by file using the ISIS Commands.

The attached listing of DCPY is the output of the INTEL Relocatable assembler. After the program is loaded the disk to be copied is placed in Drive 0 and a *formatted* disk which is to be written on is placed in Drive 1. The program is executed by directing the program counter to COPDK. During the copy operation the system will indicate to the operator that each track is copied by a message on the CRT. When the copy operation is complete the program will return to the Monitor.

As the program is written it requires a buffer of 6656 bytes beginning at location 6000H. The buffer address may be changed however in line 7.

A similar program can be written for the INTEL Single Density DOS. The major difference is the size of the parameter blocks.

```
.SM80 DCPY.SRC DEBUG

'ISIS-II 8080/8085 MACRO ASSEMBLER, V2.0

LOC  OBJ            SEQ     SOURCE STATEMENT

                      1 ;   PROGRAM TO COPY A DISKETTE TRACK BY TRACK
                      2 ;   INTEL DOUBLE DENSITY FLOPPY DISK SYSTEM
                      3 ;
                      4 ;
                      5       ASEG
FF0F                  6       MONIT EQU 0FF0FH
                      7       CSEG
                      8 ;
                      9 ;
                     10 ;   MAIN ROUTINE TO COPY A DISK
                     11 ;
0000 CD3800      C   12 COPDK: CALL DCPY
0003 C30FFF          13       JMP MONIT
                     14 ;
                     15 ;
                     16 ;   ROUTINE TO CHECK STATUS OF DRIVE 0
                     17 ;
                     18 STAT0:
0006 DB78            19       IN 78H          ;INPUT CONTROLLER STATUS
0008 E619            20       ANI 19H         ; MASK STATUS
000A FE19            21       CPI 19H         ;CHECK DRIVE 0 STATUS
000C C20600      C   22       JNZ STAT0
000F C9              23       RET
                     24 ;
                     25 ;   ROUTINE TO CHECK STATUS OF DRIVE 1
                     26 ;
                     27 STAT1:
0010 DB78            28       IN 78H
0012 E61A            29       ANI 1AH
0014 FE1A            30       CPI 1AH
0016 C21000      C   31       JNZ STAT1
0019 C9              32       RET
                     33 ;
                     34 ;   LOAD ADDRESS OF PARAMETER BLOCK AND EXECUTE COMMAND
                     35 ;
                     36 PADDR:
001A 7B              37       MOV A,E
001B D379            38       OUT 79H         ;LOAD LSB
001D 7A              39       MOV A,D
001E D37A            40       OUT 7AH
0020 C9              41       RET
                     42 ;
                     43 ;CHECK COMPLLETION STATUS
                     44 ;
                     45 CMPLT:
0021 DB78            46       IN 78H
0023 E604            47       ANI 4           ;MASK INTERRUPT BIT
0025 CA2100      C   48       JZ CMPLT
0028 DB79            49       IN 79H          ;READ RESULT TYPE
002A 57              50       MOV D,A
002B DB78            51       IN 78H          ;READ RESULT BYTE
002D 5F              52       MOV E,A
```

```
002E C9             53        RET
                    54 ;
                    55 ;CHECK FOR ERROR
                    56 ;
002F 7B             57 ERCHK:   MOV A,E
0030 E6FF           58          ANI 0FFH
0032 00             59          NOP
0033 00             60          NOP
0034 C20FFF         61          JNZ MONIT
0037 C9             62          RET
                    63 ;
                    64 ;COPY A DISK IN DRIVE 0 TO DRIVE 1
                    65 ;
0038 0600           66 DCPY: MVI B,0
                    67 ;
003A CD0600    C    68       CALL STAT0
003D 111B00    D    69         LXI D,RECO       ;RECALIBRATE DRIVE 0
0040 CD1A00    C    70         CALL PADDR
0043 CD2100    C    71         CALL CMPLT
0046 CD2F00    C    72         CALL ERCHK
0049 CD1000    C    73         CALL STAT1
004C 112200    D    74         LXI D,REC1       ;RECALIBRATE DRIVE 1
004F CD1A00    C    75         CALL PADDR
0052 CD2100    C    76         CALL CMPLT
0055 CD2F00    C    77         CALL ERCHK
0058 210060         78 L2   : LXI H,6000H       ;SET BUFFER ADDRESS
005B CD6600    C    79       CALL CTRAK         ;COPY A TRACK
005E 04             80       INR B
005F 78             81       MOV A,B
0060 FE4D           82       CPI 4DH
0062 C25800    C    83       JNZ L2
0065 C9             84       RET
                    85 ;
                    86 ;  COPY 1 TRACK SPECIFIED BY THE B REG FROM DRIVE 0
                    87 ;                                          TO DRIVE
                    88 ;  READ THE TRACK
                    89 ;
0066 78             90 CTRAK: MOV A,B
0067 320300    D    91        STA TRK        ;STORE TRACK NUMBER IN PARM BLKS
006A 320A00    D    92        STA TRKW
006D 3E01           93        MVI A,1
006F 320400    D    94        STA SECT       ;STORE SECTOR IN PARM BLK
0072 320B00    D    95        STA SECW
0075 220500    D    96        SHLD BFAD      ;IDENTIFY BUFFER ADDR TO PARM BLKS
0078 220C00    D    97        SHLD WBFAD
                    98   ;PERFORM READ OF A TRACK
007B CD0600    C    99 RT:     CALL STAT0     ;DRIVE 0 READY
007E 110000    D   100        LXI D, R0
0081 CD1A00    C   101        CALL PADDR     ;READ TRACK
0084 CD2100    C   102        CALL CMPLT
0087 CD2F00    C   103        CALL ERCHK
008A 00            104        NOP
008B 00            105        NOP
008C 00            106        NOP
008D 00            107        NOP
008E 00            108        NOP
008F 110700    D   109        LXI D, W1
                   110 WT:
0092 CD1A00    C   111        CALL PADDR     ;WRITE TRACK ON DRIVE 1
0095 CD2100    C   112        CALL CMPLT
0098 CD2F00    C   113        CALL ERCHK
009B E5            114        PUSH H
009C 210E00    D   115        LXI H ,CM1
009F CDA400    C   116        CALL PSTRG
00A2 E1            117        POP H
00A3 C9            118        RET
                   119 ;
                   120 ;PRINT MESSAGE
                   121 ;
                   122 PSTRG:
00A4 7E            123        MOV A,M
00A5 B7            124        ORA A
00A6 C8            125        RZ
00A7 CDAE00    C   126        CALL CHOUT
00AA 23            127        INX H
00AB C3A400    C   128        JMP PSTRG
                   129 ;
                   130 ;  CHARACTER PRINT TO CRT
                   131 ;
                   132 CHOUT:
00AE F5            133        PUSH PSW
00AF DBF7          134        IN 0F7H
00B1 1F            135        RAR
00B2 D2AF00    C   136        JNC $-3
00B5 F1            137        POP PSW
00B6 D3F6          138        OUT 0F6H
00B8 C9            139        RET
                   140 ;
                   141 ;
                   142 ;
                   143         DSEG
0000 00            144 R0:     DB 0H           ;PARM BLOCK FOR READ OF DRIVE 0
0001 04            145         DB 04
0002 34            146         DB 34H          ;NR. OF SECTORS TO READ
0003 00            147 TRK:    DB 0
0004 00            148 SECT:   DB 0
0005 00            149 BFAD:   DB 0
0006 60            150         DB 60H
                   151 ;
                   152 ;
0007 00            153 W1:     DB 0H
0008 16            154         DB 16H          ;PARM BLOCK FOR WRITE ON DRIVE 1
0009 34            155         DB 34H
000A 00            156 TRKW:   DB 0
000B 00            157 SECW:   DB 0
000C 00            158 WBFAD:  DB 0
000D 60            159         DB 60H
                   160 ;
                   161 CM1: ;                          MESSAGE
000E 54524143      162          DB 'TRACK COPIED',0
0012 4B20434F
0016 50494544
001A 00
                   163 ;
001B 00            164 RECO:   DB 0H
001C 03            165         DB 03H
001D 01            166         DB 1
001E 00            167         DB 0
001F 00            168         DB 0
0020 00            169         DB 0
0021 60            170         DB 60H
0022 00            171 REC1:   DB 0H
0023 13            172         DB 13H
0024 01            173         DB 1
0025 00            174         DB 0
0026 00            175         DB 0
0027 00            176         DB 0
0028 60            177         DB 60H
                   178     END
```

PUBLIC SYMBOLS

EXTERNAL SYMBOLS

USER SYMBOLS

RFAD	D 0005	CHOUT	C 00AE	CM1	D 000E	CMPLT	C 0021	COPDK	C 0000		
ERCHK	C 002F	L2	C 0058	MONIT	A FF0F	PADDR	C 001A	PSTRG	C 00A4		
REC1	D 0022	RT	C 007B	SECT	D 0004	SECW	D 000B	STAT0	C 0006		
TRKW	D 000A	W1	D 0007	WBFAD	D 000C	WT	C 0092				

ASSEMBLY COMPLETE, NO ERRORS

CTRAK	C 0066	DCPY	C 0038		
R0	D 0000	RECO	D 001B		
STAT1	C 0010	TRK	D 0003		

Letters

COMPRESSED SUBROUTINE CALLS FOR SC/MP

Dear Sirs, *Received: 78 Aug 24*

With reference to your SC/MP subroutine manager in the March 1977 issue of Dr. Dobbs, please find enclosed a compressed version of the same.

Yours faithfully,
Phil Hughes
Microprocessor Marketing Manager

National Semiconductor
2900 Semiconductor Dr.
Santa Clara, CA 95051

7.2.6 Compressed Subroutine Calls.

This example permits the user to impliment nested subroutines while using only 3 bytes per call.

```
Initialization    LDI     H(HANDL)    ; Load address of
                  XPAH    P3          ; Subroutine handler
                  LDI     L(HANDL)
                  XPAL    P3
                  .
                  .
                  .
Call              XPPC    P3
                  .ADDR   SR1         ; Address of Subroutine 1
                  .
                  .
                  .
                  XPPC    P3
                  .ADDR   SR2         ; Address of Subroutine 2
                  .
                  .
                  .
                  etc.

RETURN:           JMP     RET (P3)
                  JP      RET (P3)    ; Any one of these
                  JNZ     RET (P3)
                  JZ      RET (P3)

Organization Routine

RETURN:           ST      -1(P2)      ; Exit Subroutine
                  LD      @1(P2)      ; Get us byte of RET ADDR
                  XPAL    P3
                  LD      @1(P2)
                  XPAH    P3
                  LD      -3(P2)      ; Restore A  U

HANDL:            XPPC    P3
                  ST      @-3(P2)     ; Save ALLU
                  LD      @2(P3)      ; Adjust P3
                  LD      (P3)        ; Load lower S/R ADDR
                  ST      1(P3)       ; Save temp
                  LD      @-1(P3)     ; Get upper part
                  XPAH    P3
                  ST      2(P3)
                  LD      (P3)
                  XPAL    (P3)
                  LD      @1(P3)      ; Restore ACCU
                  JMP     HANDL
```

RET = RETURN - HANDL.

SAM76-FORTH-STRUBAL

Received: 78 Sept 4

In Volume 3, Number 8 of *Doctor Dobbs'* Mr. R. D. Grappel, a co-devisor of STRUBAL criticized "FORTH" on a variety of points. Since the SAM76 language might be subjected to virtually identical criticism I find it appropriate to submit the following in defense of FORTH and SAM76.

Mr. Grappel criticizes FORTH on the basis that the published representations are difficult to read. He by implication suggests that "pseudo English" is the way to make things legible; for some people that may be true—particularly if they use a latin character set. Many people in the world use other alphabets and linguistic representations, and for them STRUBAL would be quite foreign, whereas the symbology of FORTH and SAM76 is readily adaptable to their needs and desires.

Furthermore it would appear that Mr. Grappel fails to recognize that FORTH, and SAM76 also, is a "core langauge" capable of self redefinition to the extreme of looking just like STRUBAL—or APL—or BASIC—etc. There is a speed price to pay for such conversion, but that is one way of documenting a program for the benefit of the reader—and yet execute the procedures as written in their highest form.

I am not familiar with the internal details of FORTH, but I would hope that the FORTH interest group maintains a registry of well defined resident primitive functions and that means are provided within FORTH as they are in SAM76 for dynamic change of resident function names to permit international users to select function names or mnemonics best suited for their native languages. Portability of programs on an international basis is thus facilitated without imposing "anglicisms" on the rest of the world. Note that conversion of programs by editing is usually not a viable method because in languages such as SAM76, and probably FORTH, the function names might not exist until computed at run time, and consequently the only safe way is to dynamically change the state of the machine to match the program being run.

Mr. Grappel merely mentions in passing the fact that STRUBAL requires an overhead of multiple passes and needs intermediate files. This renders interactivity and immediate write—test—run mode of operation impractical if not impossible. Consequently "one shot" programs written in STRUBAL require considerably more time than those wirtten in FORTH or SAM76 before the user sees the result sought.

The issue of speed is becoming rapidly a non-issue, and if one accepts the forecast of infinite speed—zero cost computation by the year 1985, then that aspect of STRUBAL becomes totally irrelevant.

Mr. Grappel also glosses over the fact that STRUBAL runs in a minimum of 16K bytes (possibly 15.7 or 15.8 with

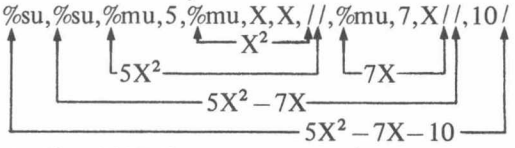
skilled reprogramming) and that the object code for just the one particular program is 3.9K bytes as opposed to the 5 to 6K bytes for the FORTH interpreter. I hardly consider it a saving to have a run time program be 4K bytes versus 5 or 6K bytes if all it can do is one thing. It is even less of a saving if you need a prior overhead of 16K just to achieve that saving.

Again the issue of space is becoming rapidly a non-issue and the fact that the run time program is 4K or 6K becomes totally irrelevant.

It is interesting to note that the number of symbols that the user has to key in to produce the polynomial illustrated "5X **2−7X−10" is approximately the same in both FORTH and SAM76 as illustrated below:

```
FORTH:POLY DUP DUP * 5 *SWAP 7 * −10−;
SAM76 %dt,POLY,!%su,%su,%mu,5,%mu,X,X,//,
   %mu,7,X//,10///
```

For the purpose of clarity the content of the SAM76 language text named POLY is analyzed as follows:



Admittedly SAM76 is not primarily intended to be a scientific or mathematical language, but it is powerful enough to simulate a substantial portion of APL in about 600 bytes of user defined procedures. The interpreter requires approximately 8K bytes of Z80 code and this can be reduced, without reprogramming, by about 300 bytes at a cost of 75 microseconds per subroutine call (2 megahertz clock system).

It should be noted that the philosophy underlying the implementation of the SAM76 language required the provision of a very rich resident function set−approximately 150−explaining the fact that the interpreter is significantly larger than required by FORTH.

Cordially,
Ancelme Roichel,

Box 257 RR1,
Pennington, NJ 08534

Descriptions and updates of the SAM76 language may be found in IEE Computer Society Repository, *Dr. Dobbs'* Volume 3, Issue 1, 3, 6 and 8. A good beginner's tutorial appears in *Creative Computing* (May-June) and (Sept-Oct) 1978.

FORTH/STRUBAL REBUTTAL

Greetings: *Received: 78 Sept 22*

I enjoyed Robert Grappel's response to my FORTH article. His language STRUBAL looks like a good piece of work, a strong candidate to replace BASIC or FORTRAN on small machines. But I question his comparisons of STRUBAL and FORTH.

Let's start where STRUBAL is strongest−clarity and understandability of programs. I admit that for small programs like the LIFE example which Mr. Grappel re-wrote, STRUBAL is clearer than FORTH−to programmers oriented to BASIC or FORTRAN. But since STRUBAL evidently isn't extensible, somebody reading STRUBAL sees the primitive statements−FOR, NEXT, STRING, PRINT, XTRACT, etc.−no matter how large the application. In contrast, a large FORTH program defines an application-oriented vocabulary −and if it's documented, the user seldom or never has to look at the source code of those definitions. A brief description of what each one does, plus a list of arguments in and results out, is enough. A STRUBAL programmer always goes back to the statements of the original language, while a FORTH programmer can use large or small chunks of previous work, interchangeably and with no separate subroutine protocols, linking loaders, or trouble with name conflicts. And FORTH is structured, with no GOTO and no statement labels in the system.

Of course any extensible language puts more responsibility on the application communities to develop reasonable standards and to avoid useless idiosyncracy. There is more incentive for communication. Is this a problem, or an opportunity?

Mr. Grappel pointed out that his LIFE program in STRUBAL took less time to develop than Boulton's FORTH version of LIFE which appeared in my article. But most of Boulton's time was spent on the word "ARRAY," which is a special kind of operation (a "defining word") which creates a new data type. Most FORTH systems come with ARRAY already defined (and now Boulton's has it too). STRUBAL also has an array type available, but if a new data type were needed, it would take some time indeed to define it (by changing the compiler).

STRUBAL is a little faster than FORTH when speed *doesn't* count. When it does, critical sections such as inner loops can be re-written using FORTH's assembler, to run at full machine speed. FORTH interfaces so naturally to machine code that the user doesn't need to know which operations are code and which are higher-level; except for speed it's transparent. And the higher-level is fast enough that code is seldom necessary; higher-level FORTH overhead is commonly 30 percent over machine code (on 16-bit stack-type machines and more like 100 percent on 8-bit micros). FORTH executes an indirect threaded code which should not be confused with usual interpreters which are ten or more times slower than code.

And now for compactness. Indeed it's true as Mr. Grappel showed that both systems take about the same space to run that little LIFE example (1888 bytes for STRUBAL vs. 475 for FORTH, but 3.9K for the STRUBAL run-time package, and 5 to 6K for the entire FORTH system, which usually stays in memory at run time). However if you multiply the size of the project by 50, you get 97K for STRUBAL vs. 30K for FORTH. And FORTH would probably compare even better than this, because programming is in increasingly larger chunks as the application grows, and each invocation of any chunk (regardless of its size) takes two bytes.

FORTH (like STRUBAL) is supposed to use a floppy, but recently I played with a stripped-down version which uses no external storage except an ordinary cassette. This bare-bones FORTH has no assembler and no editor−it resides in 3.8K bytes, running stand-alone except for a ROM monitor which handles reading and writing of the cassette, and I/O for single characters (which FORTH could do, but it uses the monitor since it is available). While I wouldn't want to develop a major project on this little system, it's remarkable how conveniently it does work. You load the cassette (about 20 seconds), then type in and run FORTH source statements. As soon as each statement is typed, it is ready to run, as it compiles while being typed. With no disk or editor, if you make a mistake and don't discover it until after a carriage

return, you have to re-type the whole statement (which is short, seldom over three lines). FORTH debugging is exceptionally thorough, because the statements are short and modular. Without the editor, when bugs are discovered the statement must be typed over; and worse, on those rare occasions when a statement way back in the program must be changed, all later statements which refer to it must also be retyped, or they will continue to refer to the old definition. This is why you want a floppy and an editor.

After developing some definitions, you can 'FORGET' early scratch work back as far as desired, re-type what is necessary (e.g. your new utilities, color graphic operations, or other applications), then save the whole system on a new cassette. Next time it will take a couple extra seconds to load the tape, with perhaps a few hundred more bytes, and your new language is ready. There is no line where the FORTH system ends and your applications begin—the whole language (including your programs) is homogeneous.

Later when I do get a floppy for this machine, this little bit of FORTH will grow into a full professional development system, just by having three or four pages of FORTH source code (for an assembler, floppy I/O, and a text editor) typed in.

Sincerely yours,
John S. James.

1090 Miller Ave.
Berkeley, CA 94708

For more information on FORTH, see *Forth Dimensions*, the newsletter of the Forth Interest Group, 787 Old County Rd., San Carlos, CA 94070. Subscriptions are $5. for issues one through six.

ASTRONOMY APPLICATION FOR PET FORTH

Dear Tom:

Received: 78 Sept 11

After reading Robert Grappel's letter (No. 28, page 28) comparing FORTH and STRUBAL, I felt that my experience with FORTH might be of interest to some of your readers. Although I had been vaguely aware of FORTH for some years, I had never had the opportunity of trying it out until Programma Consultants (3400 Wilshire Blvd., Los Angeles, CA 90010) made it available for the PET 2001. I've had their PET-FORTH version 1.0 for two weeks now and I find it to be an ideal language for my needs.

I am currently building microprocessor systems for astronomical data acquisition and analysis. For this application I must be able to control custom I/O interfaces and other hardware. My machines must be physically small and inexpensive, so large memory and mass storage devices are undesirable. It must be possible to design code that results in simple and yet flexible user interfacing. Easy program modification and testing are essential since much of the program expansion takes place during (hopefully short) periods of cloudiness at the telescope. FORTH, with its resident interpreter, compiler, and assembler, meets my needs perfectly. I can write a line of code to operate an external device and then immediately execute that code to see what happens. Since my debugging involves hardware problems as well as software problems, separate passes to compile, then execute would be inconvenient.

Since FORTH code runs at almost full machine speed (the 100% penalty generally quoted for microprocessors refers to general FORTH code, not tightly coded inner loops), I have had no trouble coding clock routines etc. where timing is important. Several comparisons of FORTH's speed versus BASIC's speed in the PET have yielded ratios of x 9 to x 18 (but BASIC was working to ten decimal digits accuracy, FORTH only to five digits).

The real advantage of FORTH, however, lies in its essential function of defining the vocabulary of a special purpose dictionary. I can define dictionary entries which carry out the various basic functions of the telescope and data system. The user can then use the FORTH machine as a programmable calculator with the equivalent of many special function commands. As an example, the following code might point the telescope at a specific position (right ascention -RA- and declination -DEC-), select a blue filter, take data for 10 seconds, and record that data.

```
12 33 26 RA= -21 33 06 DEC = POINT
BLUE FILTER 10 SECS TAKE DUP DATA = RECORD
```

In this example I use previous FORTH definitions to store the coordinates, activate the telescope positioning motors, push the filter number of the BLUE filter on the stack, output that filter number, set up a data integration time, digitize the data, store it at the next location in the array DATA, and finally output it to the recorder. Although the definitions of these functions might not be "readable," I think most would agree that the final code used by the observer is very readable. For my purposes definition readability is much less important than this ability to define a logical and simple vocabulary for the user. Even the best observer tends to become a bit muddled at 3AM on a cold, partly cloudy night and I want to give him as much help as possible. Although STRUBAL or other languages could have been used to program a series of prompts ("RA?", "DEC?", "FILTER?"), this would "lock in" the exact sequence of operations. FORTH gives me the flexibility to set up a variety of operations, repeat them, *and change them* as I choose.

I don't think it is necessary to comment at any length on specific points of Mr. Grappel's letter, but I would like to make the following points:

1. His first example of the difficulty of reading FORTH (the POLY definition) essentially comes down to the old argument of postfix versus infix operators. Some like one (and buy HP calculators), some like the other (and buy TI calculators).
2. The comparison of a 3.9K run-time STRUBAL package with the full 5–6KFORTH package is unfair. Using the cross-compile option of FORTH would result in a run-time FORTH package of less than 1K bytes for this program.
3. I doubt that STRUBAL would run "much faster" than FORTH since optimum machine language code could only run twice as fast as FORTH.

To close, STRUBAL looks like an excellent choice for the BASIC programmer who wants to move up to a compiled language, but for me FORTH looks much more useful.

Sincerely,
John P. Oliver
Associate Professor of Astronomy.

Department of Physics & Astronomy
University of Florida
Gainesville, Florida 32611

QUESTIONS – AND ANSWERS – ON DATA TRANSFER

Dear Mr. Pugh: *Received: 78 Aug 21*

First let me say how much I appreciated your article in *Dr. Dobb's Journal*, No. 26. Your article provided a great deal of information in a field in which I am just now getting started. Toward this end, I have subscribed to the PCNET Committee publications recently. I also plan to start attending their meetings in this area.

Right now I am putting together a disc system using a Tarbell board and a couple of drives. As soon as I have this system operating, I plan to obtain your program from the CP/M User's Group.

All the above gives some background for the questions I am about to ask: What is the reason for having to operate the system at a baud rate half of that of the terminal console? How can this limitation be overcome? I would like to operate at a baud rate at least of 300.

In connection with your discussion of losing data during file transfers, I anticipate even more trouble than you have had because from the documentation I have, my head will unload after only 2 revolutions (I am using Shugart 800 drives). I do not know yet how I will solve this difficulty.

Very truly yours,
Robert L. Edens

1096 W. Remington Dr.
Sunnyvale, Calif. 94087

Dear Mr. Edens:

I am glad to see you plan to join the PCNET as I believe the more participation it receives the more effective it will become.

As to the questions you raised in your letter, I am afraid I did a very superficial job in explaining system timing. The following discussion may be helpful in clarifying this.

1. If communication is taking place between two personal computers with each operating at 300 Baud (Modem rate) and each transmitting via the keyboard, there should be no data lost even if a 110 Baud terminal is being used as console device for one of the computers. This is so because it is only the character burst rate that occurs at 300 Baud. In order to cause the true data rate to be 300 Baud, the operator with the fast console device would have to type at a rate of 300 words per minute—where a word consists of 5 letters and a space on average.
2. If a large time sharing computer is communicating with a personal computer at 300 Baud and the console device of the personal computer (PC) uses a 300 Baud printing terminal as console, some data will probably be lost. This is because the TSC outputs a line at a time and usually transfers it at the maximum rate allowed by its modem device. At the PC end of the communications, some delay is inserted between the receipt of data by the computer and the relay of this data—on a character by character basis—to the console device. This is caused by the time taken to execute the instructions necessary to input/output the data and, perhaps more time consuming, check the status of each I/O device (the modem and console respectively) to see if they are ready for a data transfer. This time delay builds up with each character transmitted if the console is only capable of 300 Baud operation. If the data string is sufficiently long, data loss will occur. If the console device were capable of transfering data at 600 Baud then both the computer and the console would be "twiddling their thumbs" while waiting for receipt of the next character.

   Perhaps this answers the question as to why the system (modem) must operate at a baud rate half that of the terminal console. Of course there is no problem if the console is operating at a baud rate significantly higher than the modem rate, e.g., modem.rate=300 Baud; console. rate=9600 Baud. Further, choosing a ratio of 2 to 1 is really an oversimplification. The real value is somewhere between 1 to 1 and 2 to 1. I can only say that my printing terminal successfully copied data at a rate of 150 Baud while it was operating at 300 Baud. However, it would not copy data reliably if its rate were set equal to the data rate.

   As to the question of data loss during file transfer. No data loss will occur when your PC is transmitting files since it does so "in its own good time." However, when you are receiving a file, it is the guy at the other end who determines the rate, i.e., whether the data is passed through the modem at its maximum rate or whether there is a certain amount of "dead time" between characters or groups of characters. The following approaches can be used to overcome this problem:
3. At 300 Baud each character requires a minimum of 30.3 msec time, i.e., (30.3 msec/Character ==> (30.3 sec)/(1K characters). Thus, it takes approximately 1 minute to transfer 2K characters. If, instead of writing each block of 128 characters out to disk as soon as the data block is accumulated, one were to buffer (save) the data in memory, one could buffer 15 minutes of data with 30K of memory. The data could then be recorded on disk. Thus, by breaking up the data to be transferred into manageable "time slices" one could transfer a large file without data loss.
4. If communication is occuring between computers which adhere to the PCNET protocol (or some similar protocol) then a file can be received without data loss. As in (3) above, memory buffering in excess of 128 characters (bytes) is required. However, the amount of buffered data is small compared to the above approach. Under the PCNET protocol, data is transmitted in block format with the data content of each block being 0-253 bytes. To keep things simple, assume block size=256 bytes. The receipt of each block must be acknowledged although up to four blocks may be transferred before receipt of the first block is acknowledged. If no acknowledgement (ACK) is sent, the sender goes idle and awaits ACK before transferring more data. Thus, a data buffer of 1K BYTES (4*256 BYTES) is sufficient to assure no data loss. The data thus buffered can be written to disk before the sender "times out" and breaks off communications. Following the transfer of data to disk, the ACK for each of the four data blocks can be sent to the sender resulting in the restart of data transmission of the remaining data, etc.
5. If one is communicating with a TSC which does not provide for PCNET protocol and one can not afford to set aside a large buffer as in (3) above, then he may achieve the desired result by using interrupt processing and double

buffering. Thus if one had an IMSAI PIC-8 board or similar interrupt handling board and a DMA floppy disk controller he could force an interrupt each time data was available from the modem and store it in one of the two buffers allocated for data storage. When the first buffer became full the incoming data would be stored in the second buffer while the contents of the first buffer were written to disk. Each buffer could have a size equal to one physical disk record (128 bytes). Thus, only 256 bytes of memory is needed to implement double buffering.

It may be possible to perform interrupt processing without a DMA controller for the floppy disk but I have not yet figured out how to do it. Consider that whereas it takes only 4.1 msec to transfer one physical record (128 Bytes) to disk—assuming a 250K bit data transfer rate; the total time required may be much longer. This is because the time between blocks is sufficiently long to assure that the head has "unloaded." Thus, it takes 35 msec to load the head and an average of 83 msec to reach the desired sector—one revolution of the disk takes 166.6 msec. Thus, 118 msec may be required before writing can begin. Since the microprocessor must be ready to control the writing of each data byte, it must wait for the instant that the sector has been reached and immediately begin data transfer. This assumes that the read/write head is already positioned at the proper track. Since at 300 Baud each character takes 30.3 msec one will "drop" 3 to 4 characters on average while attempting to write a record to disk.

Sincerely,
Tim J. Pugh, Jr.

9655-M Homestead Court
Laurel, Md. 20810
Phone: (301) 776-5253

LAST CHANCE ON DG TV READOUT CARD

Dear Dr. Dobbs: *Received: 78 Sept 13*

It has come to my attention that The Digital Group has discontinued production of Dr. Suding's 16 by 32 TV Readout Card. They still have on hand approximately fifty printed circuit cards which they are selling for $15.00 a piece.

This video card has some very nice features: parallel interface to the computer for fast update, on board memory, use of the Motorola MCM 6571 pre-programmed character generator (there are ten different ones to choose from) and the card connects easily to a television set via an RF modulator (Pixe-Verter #PXV-2A) and a 72 ohm to 300 ohm transformer (Radio Shack). I use a wire-wrapped version which is connected to a 12 inch Sony and I get an excellent picture. If you decide to purchase the card be sure to get the 5.990 MHz crystal at $4.95 and the schematic and circuit description DOC TVC-1-R1 for $1.50. Altogether you shouldn't have to spend over $80.00 for the card and parts, not counting the RF modulator and the 72 ohm to 300 ohm transformer. A good buy, considering you also get the printed circuitry for the Suding 1100 Baud Casette Interface on the same card. I would recommend a couple of modifications. Capacitors C1, C2, C3 and C4 should be changed from Mylar to Mica. My board works better without capacitor C4. If you wish to use the white rectangle for a cursor rather than the underline, the least significant data line must be inverted by inserting a 7400 NAND gate between pin connector 8 and IC 22. IC 22 must be changed from a 7430 to 74H30. This makes the card home up on octal 376 and frees the white rectangle for use as a cursor.

The enclosed program is a general keyboard input program with video echo and utilizes a software-controlled cursor. If you do not invert the LSB, byte 075 must be changed from 177 to 137 and byte 174 from 376 to 377. The various subroutines contained within the program are screen clear with cursor at home-up position, back space, carriage return, up one line and down one line. If you use a status bit with your keyboard, a call to your own input routine should be inserted at 011.

Sincerely,
Don Woods.

12012 Pebblebrook Lane
Carmel, Indiana 46032

P.S.

1. There are two obvious errors in the schematic. Line 4 of IC 29 and line 4 of IC 9 do not connect to the five plus line. Oddly enough the same mistake appears in the schematic for DG's 1024 Character TV Readout, IC 13 and IC 3.
2. Prime parts can be obtained from Jameco Electronics. Good prices and fast service except for their 22/44 printed circuit edge card connector. Stick with Vector, Cinch or TEKA.

```
100  000  061 111 hhh   LXI SP          SET STACK
     003  315 152 100   CALL           CALL CLRSC
     006  315 140 100   CALL           CALL TVDSP1
     011  000           NOP
     012  333 004       IN       KYBRD IN PORT 4*
     014  346 177       ANI            MSB EQU 0
     016  303 362 100   JMP            JUMP TO KYBRD PATCH
     021  376 015       CPI            COMPARE RETURN KEY
     023  314 217 100   CZ             CALL CRLF
     026  376 137       CPI            COMPARE BACK SPACE KEY
     030  314 241 100   CZ             CALL BKSP
     033  376 012       CPI            COMPARE LINE FEED KEY
     035  314 152 100   CZ             CALL CLRSC
     040  376 025       CPI            COMPARE CONTROL U
     042  314 271 100   CZ             CALL UP
     045  376 004       CPI            COMPARE CONTROL D
     047  314 303 100   CZ             CALL DWN
     052  376 000       CPI            OTHER DATA THAN CONTROL ?
     054  304 140 100   CNZ            CALL TVDSP1
     057  303 011 100   JMP            JMP BACK TO KYBRD
     062  076 040       MVI A          LOAD A WITH BLANK
     064  366 200       ORI      TVNC  SET MSB HIGH
     066  323 003       OUT            OUT PORT 3*
     070  257           XRA A          CLEAR A
     071  323 003       OUT            TURN OFF STROBE*
     073  311           RET            RETURN
     074  076 177       MVI A          LOAD A WITH WHT RECTANGLE
     076  315 064 100   CALL           CALL TVNC
     101  311           RET            RETURN
     102  257           XRA A    TRIGR CLEAR A
     103  107           MOV B,A        CLEAR B
     104  074           INR A          LSB EQU 1
     105  315 066 100   CALL           CALL TVNC PLUS 2
     110  005           DCR B          DECREMENT B
     111  302 104 100   JNZ            JUMP BACK IF NOT DONE
     114  005           DCR B          DECREMENT B
     115  074           INR A          LSB EQU 1
     116  315 066 100   CALL           CALL TVNC PLUS 2
     122  005           DCR B          DECREMENT B
     123  302 115 100   JNZ            JUMP BACK IF NOT DONE
     125  311           RET            RETURN
     126  315 102 100   CALL     CURSR CALL TRIGR
     131  315 074 100   CALL           CALL WHT RECTANGLE
     134  315 102 100   CALL           CALL TRIGR
     137  311           RET            RETURN
     140  315 176 100   CALL     TVDSP1 CALL TVDSP2
     143  315 062 100   CALL           CALL BLANK OUT
     146  315 126 100   CALL           CALL CURSR
     151  311           RET            RET
     152  001 002 000   LXI B    CLRSC LOAD BC WITH COUNT
100  155  315 062 100   CALL           CALL BLANK OUT
     160  005           DCR B          DECREMENT B
     161  302 155 100   JNZ            JUMP BACK IF NOT DONE
     164  015           DCR C          DECREMENT C
```

```
      165   302 155 100   JNZ               JUMP BACK IF NOT DONE
      170   076 040       MVI A             LOAD A WITH CRLF COUNT
      172   127           MOVD,A            SAVE COUNT
      173   076 376       MVI A             LOAD A WITH HOME BYTE
      175   311           RET               RETURN
      176   366 200       ORI       TVDSP2  SET MSB HIGH
      200   323 003       OUT               OUT PORT 3*
      202   257           XRA A             CLEAR A
      203   323 003       OUT               TURN OFF STROBE*
      205   315 212 100   CALL              CALL CNTCHK
      210   025           DCR D             DECREMENT COUNT
      211   311           RET               RETURN
      212   272           CMP D     CNTCHK  COMPARE COUNT
      213   300           RNZ               RETURN IF NOT ZERO
      214   026 040       MVI D             LOAD D WITH COUNT
      216   311           RET               RETURN
      217   315 062 100   CALL      CRLF    CALL BLANK OUT
      222   000           NOP
      223   074           INR A             LSB EQU 1
      224   315 066 100   CALL              CALL TVNC PLUS 2
      227   025           DCR D             DECREMENT COUNT
      230   302 223 100   JNZ               JUMP BACK IF NOT DONE
      233   026 037       MVI D             LOAD D WITH COUNT
      235   315 143 100   CALL              CALL TVDSP1 PLUS 1
      240   311           RET               RETURN
      241   315 062 100   CALL      BKSP    CALL BLANK OUT
      244   315 102 100   CALL              CALL TRIGR
      247   315 126 100   CALL              CALL CURSR
      252   024           INR D             INCREMENT COUNT
      253   076 041       MVI A             LOAD A WITH D CNT PLUS 1
      255   272           CMP D             COMPARE A WITH D
      256   302 264 100   JNZ               JUMP IF SAME
      261   076 001       MVI A             LOAD A WITH 1
      263   127           MOV D,A           SET D COUNT TO 1
      264   257           XRA A             CLEAR A
      265   311           RET               RETURN
      266   000           NOP
      267   000           NOP
      270   000           NOP
      271   315 062 100   CALL      UP      CALL BLANK OUT
      274   315 315 100   CALL              CALL UP 32
      277   315 126 100   CALL              CALL CURSR
      302   311           RET               RETURN
      303   315 062 100   CALL      DWN     CALL BLANK OUT
      306   315 344 100   CALL              CALL DWN 32
      311   315 126 100   CALL              CALL CURSR
      314   311           RET               RETURN
100   315   325           PUSH D    UP 32   SAVE COUNT
      316   257           XRA A             CLEAR A
      317   001 340 000   LXI B             LOAD BC WITH COUNT
      322   074           INR A             LSB EQU 1
      323   315 066 100   CALL              CALL TVNC PLUS 2
      326   005           DCR B             DECREMENT B
      327   302 322 100   JNZ               JUMP BACK IF NOT DONE
      332   074           INR A             LSB EQU 1
      333   315 066 100   CALL              CALL TVNC PLUS 2
      336   015           DCR C             DECREMENT C
      337   302 332 100   JNZ               JUMP BACK IF NOT DONE
      342   321           POP D             RESTORE COUNT
      343   311           RET               RETURN
      344   325           PUSH D    DWN 32  SAVE COUNT
      345   257           XRA A             CLEAR A
      346   006 040       MVI B             LOAD B WITH COUNT
      350   074           INR A             LSB EQU 1
      351   315 066 100   CALL              CALL TVNC PLUS 2
      354   005           DCR B             DECREMENT B
      355   302 350 100   JNZ               JUMP BACK IF NOT DONE
      360   321           POP D             RESTORE COUNT
      361   311           RET               RETURN
      362   376 033       CPI       PATCH   COMPARE
      364   312 XXX XXX   JZ                JUMP TO MONITER
      367   303 021 100   JMP               JUMP BACK TO KYBRD
      372   000           NOP
      373   000           NOP
      374   000           NOP
      376   000           NOP
      377   000           NOP

*MUST BE CHANGED TO CONFORM TO USERS OWN PORTS
```

LEDIP AVAILABILITY

Dear Tom: *Received: 78 Sept 6*

Your readers might be interested in the following additional information regarding LEDIP, the KIM/6502 text editor.

Version K6 of LEDIP is available from The 6502 Program Exchange. Version K6 occupies less memory (116 bytes) than the published version (Version K4). The reduction in code was achieved by deleting the LINE REPLACE/ADJUST routines and using the LINE DELETE and LINE INSERT routines to achieve the same result. The penalty for this code reduction is a loss in speed which becomes apparent when editing large amounts of text. Apart from this difference, the two versions are completely compatible. In fact, the LEDIP user need not be aware of this difference.

Version K6 source listing is available for $4.50. LEDIP manual with sample runs and hex dump is available for $3.00. LEDIP hex paper tape or KIM hypertape cassette (on user's cassette) is available for $2.50. The 6502 Program Exchange, 2920 Moana, Reno, NV 89509 can be contacted for more information. Thanks.

Sincerely yours,

Kiumi Akingbehin, Computer Science Department
Wayne State University, Detroit, Michigan 48202

· Announcements ·

SUMMER 1979 COMPUTER SCIENCE SPECTACULAR

Received: 78 Sept 14

The Summer 1979 Institute in Computer Science at the University of California, Santa Cruz, U.S.A., will feature two major programs—an advanced course on *Computing Systems Reliability* to be offered July 9-20, and a symposium on *Programming Methodology* to be held in August.

Programming Methodology features 25 of the world's leading figures in computer science including Edsger W. Dijkstra, Burroughs Corporation, The Netherlands; David Gries, Cornell University, U.S.A.; Donald Knuth, Stanford University, U.S.A.; C.A.R. Hoare, Oxford University, England; O.J. Dahl, University of Oslo, Norway; J.J. Horning, Xerox PARC, U.S.A.; Niklaus Wirth, Institut fur Informatik, ETH, Zurich, Switzerland; W. M. Turski, Warsaw University, Poland; and William McKeeman, University of California, Santa Cruz. The symposium is three weeks in length. Participants may attend one, two or three weeks of the program.

Computing Systems Reliability will be presented by Brian Randell, University of Newcastle upon Tyne, England, and a dozen renowned lecturers.

For information, write to Institute in Computer Science, University of California Extension, Santa Cruz, California, U.S.A. 95064.

COMPUTERS IN THE HUMANITIES

Received: 78 Sept 10

The Fourth International Conference on Computing in the Humanities will meet August 19-22, 1979, at Dartmouth

College. Papers on any topic involving computing in the humanities will be considered for presentation. Languages of the conference will be English, German, and French. ABSTRACTS of 500 to 1000 words should be submitted by December 1, 1978; acceptances will be announced by April 1, 1979. For further information, contact Stephen V. F. Waite, Chairman, ICCH/4, Kiewit Computation Center, Dartmouth College, Hanover, New Hampshire 03755.

· Club News ·

Z USERS GROUP FORMED

The formation of a Users Group for users of the Z-8, Z-80, Z-8000, Mostek 3880, and other microprocessors in the same family, to be called the "Z Users Group," was announced by Jon D. Roland, its organizer.

Roland called for contributions to the newsletter that the Z Users Group will publish, including systems and applications software, interfacing problems and solutions, evaluations, extensions, and modifications to products based on these chips, criteria for selecting chips for various applications, proposed Z-8000 bus standards, benchmark programs, ideas for features in future chips in this family, and information on new products, suppliers, software available, and interesting applications. The Z Users Group will be independent of Zilog, Inc., Mostek, or any other producer or user of Z-chips.

One project Roland hopes to get support for is a group purchase of Z-8000 chips, so that members can get some of the first chips at reasonable prices for evaluation and development. The address of the Z Users Group is 1015 Navarro, San Antonio, Texas 78205. (512) 222-1427. The dues are $5.00 per year.

CLEVELAND DIGITAL GROUP

Received: 78 Sept 1

The Cleveland Digital Group meets the third Sunday of every month at 2 p.m. The CDG clubhouse is located 100 feet east of Broadway Ave. at 8700 Harvard Ave., Cleveland, Ohio 44105.

TRACE

The Toronto Region Association of Computer Enthusiasts, Box 545, Streetsville, Ontario L5M 2C1 Canada, are assembling a list of certain of their members who are willing to assist other members with hardware and software questions in exchange for whatever the two parties can agree on. Their newsletter, TRACE also contains a helpful question and answer column.

BAY AREA

A users group for Heath owners will be starting in the San Francisco Bay Area. Contact Bob Bance, 218 Roble Ave., Redwood City, CA 94061.

PAPER STRIKE HITS HOME

Due to a long-term strike in the paper industry which has been going on since Spring, we have not been able to obtain our usual quality of paper and are forced to print this issue of *Dr. Dobb's* on lower quality paper. It is very embarrassing to have to raise the price of the magazine and then not be able to maintain the physical quality due to an industry-wide shortage. We understand that there are good prospects for a settlement soon, but that does not mean an immediate end to shortages. We will, of course, go back to our regular paper as soon as humanly possible, but in the meantime we must ask our readers to bear with us.

—Tom Williams

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 1 – 1976	Vol 2 – 1977	Vol 3 – 1978
January	1	11	21
February	2	12	22
March	3	13	23
April	4	14	24
May	5	15	25
June/July	6	16	26
August	7	17	27
September	8	18	28
October	9	19	29
November/December	10	20	30

1802. *see* COSMAC

6502
- arithmetic
  - floating-point 7/17-19, errata 10/57
  - trig functions (extensions to KIMATH) 20/15-21
- article
  - programming tricks *28/29*
- cassette interface 27/31-33
- disassembler 8/22-25
  - (BASIC) 21/6, errata 23/3, 27/39
- games
  - lunar lander 7/22-25
  - mastermind 8/26-27
- text editor 29/4-12
- utilities
  - breakpoint routine 3/17-19, errata *8/33*
  - debugger 19/42-44
  - memory test 17/23; 21/4-5
  - monitor 7/20-21
  - opcode decoder 17/20-22, 19/42-44; 22/5-7
  - relocator 14/30-31
  - string output 8/33, 10/50, 12/8
- *see also* Apple-II; KIM-1; BASIC, tiny: Pittman 6502

6800
- articles
  - LISP *28/24-25*
  - SWTPC I/O control interface *17/35-37*
  - Resident Assembler *8/19-21*
  - programming tricks *6/19-10;* 29/3
- disassembler 13/30-33
- 8080 emulator 28/4-18
- GPM (macroprocessor) 20/22-29
- graphics 16/19-20
- interface to IEEE 488 bus 16/31-55
- utilities
  - display test 26/30
  - memory clear 29/43
  - memory test 5/9; 25/28-33
  - timer routines 9/21
- *see also* SWTPC; BASIC, tiny: Pittman 6800

8080
- arithmetic
  - multi-precision 13/15-23
  - expression evaluator 29/34-40
- articles
  - BASEX (language) 30/26-31+
  - 8080 emulator (on 6800) 28/4-18
  - Intel-NEC chip differences *10/14-15*
  - programming tricks *6/9-10; 28/29*; 29/3
- disassembler 12/30-43, *17/15*; 21/38-46; 27/4-18
  - (BASIC) 13/25-29
- games
  - button button 6/38-39
  - chase (a dot) 10/58-59
  - life 10/60; 24/10-11
  - oops (line drawing) 9/30
  - unizap (teaser) 5/10-11
  - *see also* games: chase (by robots)
- graphics. *see* graphics: 8080

languages
ACT (macroprocessor) 24/27-46, (PL/M) 23/20-45
BASIC. *see* BASIC
CASUAL 10/19-32, *12/4, 12/44-45*
KAPIAR (macroprocessor) 20/31-59, errata *23/3*
LISP 30/4-11
MINOL 4/9-17 errata 6/36, 7/32
PILOT. *see* PILOT: 8080
miscellaneous *4/5; 8/12; 8/18*
music program 2/6-7, 3/6, 8/32-33, 16/21
operating system
Kernel 18/20-40
text editor 6/13-26
utilities
backspace routine for display 19/18
binary tree routines 30/15-17
bootstrap 6/8
compactor (for BASIC programs) 9/31
debugger 9/26-29
diskette copier (ISIS) 30/40-41
disk file dumper (Northstar) 28/38-41
drivers
disk 12/12-17
display 2/34; 30/48-49
modem 26/4-16
loader 3/16, 7/31
memory
clear 8/32-33
dump 26/18-23; 27/4-18; (vocal) 28/42-46
test 8/32-33
monitor 4/18-32
reformator (for assembler programs) 6/10
relocator 13/8-9, 22/14-20
sort (quicksort) 24/20
status push/pop routines 29/3
word processor 29/18-29
see also H-8; Microsoft; Mits; Polymorphics; Processor Technology; SOL
8086
description of 27/19-21

Abshire, Jim 10/52-53
accounting program (BASIC) 19/36-41
acoustic coupler. *see* modem
ACT (macroprocessor) (PL/M) 23/20-45, (8080) 24/27-46
Agoston, Max K. 18/20-40
Akingbehin, Kiumi 29/4-12
Alevy, Gary *12/28-29*
Alexander, Mark 10/34-50
Allen, Dave 16/7
Allison, Dennis *1/1-12; 2/4-5; 13/34-35;* 29/13-15
Anderson, James 29/30-33
Anderson, Ken 25/17-20
APL. *see also* EMPL
APL Core Project *17/18-19*
Apple-II (6502)
utilities for BASIC
renumber 23/4-9
append 23/4-9
erased program recoverer 26/31-33
see also 6502, KIM-1
arithmetic
binary calculator (KIM-1) 23/13-15
expression evaluator (8080) 29/34-40 (BASIC) 6/11-12; 10/52-53
floating-point (6502) 7/17-19, errata 10/57
miscellaneous functions (BASIC) 29/13-15
multi-presicision (8080) 13/15-23
square root routine (algorithm) 13/23
trig functions (extension to KIMATH) (6502) 20/15-21
see also number systems: conversions
Armstrong, Arthur 19/28-29
Arnold, John 1/13-17, 2/13-31
Arnold, Mark *2/8-12*
Assembler, 6800 Resident
article on *8/19-21*

Barber, Jerry 11/8-62
bar codes *14/36-43*
BASEX (language) *30/26-31+*
BASIC
criticisms of *2/4-5; 30/18+*
games in. *see* games
LLL (Lawrence Livermore Labs) (8080) 11/8-62
miscellaneous functions (BASIC) 29/13-15
tiny
design and history *1/1-12*
Denver (F.J. Greeb) (8080) 3/20-30, errata 8/34
Palo Alto (Lichen Wang) (8080) 5/12-15, 6/35
extensions 12/6-17; 15/6; 19/18; 26/34-141; 29/3
Pittman 6502
extensions 30/32-33
Pittman 6800
extensions 9/23; 12/5; 12/7
Texas (Whipple/Arnold) (8080) 1/14-17; 2/13-31
extensions 2/34; 6/34; 8/30-31
see also CASUAL; FOCAL; MINOL; NIBL; SCELBAL *and under manufacturer's names*
Baum, Allen 8/22-25
Baunach, S.C. 16/31-55
benchmarks (BASIC) 15/8
binary tree routines (8080) 30/15-17
black box, computer as *14/34-35*
Boaz, Ray 5/9
book reviews: *9/36; 11/15; 11/62; 14/46*
bootstrap (8080) 6/8
Borgerson, Mark J. 7/29-30
Boute, R.T. *22/46*
Bowles, Kenneth L. *23/16-19*
Bradshaw, Jack 7/20-21
breakpoint routine (6502) 3/17-19, errata 8/33
(KIM-1) 16/25-27
Bridge, Theodore E. 20/12-13; 26/24-27; 30/12-14
Britton, Keith *30/34-39*
Broom, J. *7/8-9*
Broucke, R. 16/12+; *16/13-14*; 16/17-19
Buchanan, Martin *9/32-35, 10/54-55, 13/37*
bus
IEEE 488 16/31-55
S-100 *7/8-9; 25/40-46*
STD *30/34-39*
business software
accounting program (BASIC) 19/36-41

calculator. *see* arithmetic
calculator chip, interface to micro *1/18*, errata *7/16*
cassette
interface, modem as *10/7, 14/6-7, 19/5*
interface (6502) 27/31-33
I/O (6502) 26/24-27, *29-41*
CASUAL (8080) 10/19-32, *12/4, 12/44-45*
Cecil, Alex 23/20-45, 24/27-46
CHAOS (operating system) *26/3, 27/3*
Chapman, David *30/18+*
Christensen, Ward 12/30-43
Cline, Arthur W. 23/11-12
Coats, Jack O. 3/16

codes & cyphers *7/12-15; 8/35-36; 9/16-18; 12/9-10;* (BASIC) 15/9
Cohen, Harvey A. 13/12-13; 24/10-11
Comer, Edward I. *12/11*
compactor (for BASIC programs) (8080) 9/31
compiling, article on *25/4-16*
computer control of tape deck *8/3-6*
computer crime bill *28/36-37*
COSMAC (1802)
article on *15/34-44*
utilities
I/O, monitor, music, timer 19/30-35
CP1600 *11/3+*
CP/M (operating system) *4/5*
CRT. *see* display
curve-fitting routine (FOCAL) 30/12-14

Dahl, Aubrey *28/20-23*
Data-Boy™ (speech digitizer) 29/30-33
data communication. *see* networks
Dazzler, TV. *see* graphics: 8080: dazzler
DEAFNET *17/10*
debugger
for 6502 19/42-44
for 8080 9/26-29
for Z-80 24/12-18
delivery delays, report on *19/6-9*
Denney, D. *7/8-9*
Dickenson, John 11/8-62
Digicast™ Project *29/16-17*
Dinely, M.G. 13/15-23
disassembler
for 6502 8/22-25; (BASIC) 21/6, errata 23/3, 27/39
for 6800 13/30-33
for 8080 12/30-43, *17/15*; 21/38-46; 27/4-18
(BASIC) 13/25-29
for Z-80 (BASIC) 26/42-44
disk, floppy
diskette copier (8080/ISIS) 30/40-41
file dump (8080/Northstar) 28/38-41
interface and driver (8080) 12/12-17
displays
plans for display board *6/30-33*
ideas for *6/28; 10/8; 12/4; 15/17; 16/11*
utilities
backspacer (8080)19/18
driver (8080) 2/34; 30/48-49
tester (6800) 26/30
Dompier, Steve 2/6-7

Eaton, John 20/15-21; 23/13-15
Eckard, Royce 11/8-62
EDITHA (machine-code editor) (6502) 25/34-39
editor, machine-code 25/34-39
editor, text. *see* text editor
Elizondo, Ed *27/41*
Elliott, Dan 12/24
Emerson, Richard B. *19/15-17;* 26/18-23; 27/27-30
EMPL *14/12-16; 16/11*
emulator (of 8080 on 6800) 28/4-18
Espinosa, Chris 8/33
expression evaluator. *see* arithmetic

F8
FD8 disk system, modifications to 18/44
Ferguson, Mickey E. 12/7
Finkel, LeRoy *7/36-37*
Fisher, Eugene R. 9/26-29; 11/8-62; 12/12-17
floating-point. *see* arithmetic

FOCAL (FCL65E version) *30/12-14*
Formaniak, Peter G. *13/10-11*
FORTH *25/21-27; 28/26-27; 28/28, 30/44-46*
French, Gordon 22/22-39
Fullmer, Howard *25/40-46*
Fylstra, David Jon 17/10

Gabrielson, Mike 24/20; *27/19-21*; 29/34-40; 30/15-17
games
acey-ducey (BASIC) 7/26-27
button button (8080) 6/38-39
chase
(a dot) (8080) 10/58-59
(by robots) for (8080/VDM) 15/10-17
errata 19/20
modifications
impatient robots 16/30
real time 17/26
to Poly VTI 17/27-29
proper credit *18/41*
rewrite for (8080/Dazzler) 22/30-41
life
(BASIC) 1/13; 16/12+
(8080) 10/60; 24/10-11
(FORTH) 25/24
(STRUBAL) 28/28
further references *17/24*
lunar lander (6502) 7/22-25
mastermind (6502) 8/26-27
number guess (STRUBAL) 23/10
oops (8080) 9/30
slots (BASIC) 7/26-27
stars (BASIC) 7/26-27
teaser (brain teaser, unizap, shooting stars)
(8080) 5/10-11; (BASIC) 7/29-30; *12/11*
tiny trek (MINOL) 6/37, errata 7/31
trap (BASIC) 7/26-27
zapp the moon man (CASUAL) 10/24; (BASIC) 12/24
Garetz, Mark 7/22-25
Gaugler, Gary 28/4-18
Gerhold, George *27/46-47*
Gordon, H.T. 12/8; 17/20-22, 19/42-44; 22/5-7; 25/34-39; 28/29
GPM (macroprocessor) (8080) 20/22-29
graphics
6502
GT6144 14/32; 30/32-33
6800
oscilloscope 16/19-20
8080
dazzler 19/21-27; 22/22-29; 22/30-41
P.T. VDM 8/28-29; 16/28-30; 18/8-19
Poly VTI 13/12-13; 19/28-29; (BASIC) 16/17-19
hardware *20/6-8; 26/41*
Graves, Rod *13/41-44*
Greeb, F.J. 3/20-30; 6/13-26
Greenlaw, Richard 27/4-18
Gruhn, David *12/9-10*

H-8
bad review *19/10-13*, retraction *21/3*, parody *21/5*
reviews *20/9-11*; *30/22-25*
Hahn, Harvey E. *3/15*
Hancock, Les 24/12-18; 27/22-26
handicapped *16/4*; *17/10*; *18/7*
Happy Lady *1/5-10*
hardware
new
8080 floppy disk interface 12/12-17
6502 programmable IC tester 28/33-35

hardware, new (cont.)
homebrew TV display board *6/30-33*
modifications
H-9, lower case *28/30-32*
IMSAI
RAM-4A fix *3/6, 13/7*
memory clobberer fix: *8/17; 16/60*
VDM-1, fix *17/33-34*
TVT-11
64 char/line & scrolling *12/25-27,*
proper credit *15/45,*
errata *16/56,*
extension *22/46*
64 char/line *12/28-29*
lower case *16/57*
bell *26/47*
proposed
segmented memory *12/11*
extended addressing *14/35*
Hayden, Russ 12/6-7
Heathkit. *see* H-8
Hertzfeld, Andy 26/31-33
hexadecimal (or hex). *see* number systems
HI, tiny *9/32-35, 10/54-55, 13/37*
Hicks, Bruce 16/21
Hudson, David 16/19-20
humor *8/17, 10/32; 14/44; 22/3*
Hutchinson, Perry C. *21/10-17*

IC tester, programmable (6502) 28/33-35
IM6100 7/10
IMSAI
hardware fixes
to RAM-4A *3/6; 13/7*
for memory clobberer *8/17; 16/60*
Intel-NEC chip differences *10/14-15*
index
to *The Computer Hobbyist 5/26-27*
interface
cassette (6502) 27/31-33
floppy disk (8080) 12/12-17
IEEE 488 bus (6800) 16/31-55
ISIS (operating system), analysis of *21/10-17*

James, John S. *25/21-27; 28/26-27; 28/36-37*
Julich, Paul M. 30/40-41

KAPIAR (macroprocessor) (8080) 20/31-59, errata *23/3*
Kee, Henry L. 9/23; 12/5
Kelb, Carl 7/26-27
Kendall, Bruce R. 16/22
Kernel (operating system) (8080) 18/20-40
Kheriaty, Larry *27/46-47*
Kildall, Gary A. 22/10-13
KIM-1 (6502)
breakpoint routine 16/25-27
calculator, binary 23/13-15
cassette I/O 25/24-27, comments on *29/41*
curve fitter (FOCAL) 30/12-14
disassembler 20/12-13
editor (for machine code) 25/34-39
graphics routine 14/32
programmable IC tester 28/33-35
see also 6502; Apple-II
KIMATH (arithmetic routines)
extension to (6502) 20/15-21
Kushe, Willi 16/25-27

language, interpretive
for tiny BASIC *1/6-12*
language, programming
comments on design of *3/7; 3/15; 17/15*
Dijkstra's (hacked up) *19/19-20*
implementation ideas *17/16-17*
miscellaneous *30/18+*
for robots (8080) 18/8-19; 25/17-20
translation techniques *25/4-16*
see also ACT; APL; BASEX; BASIC; CASUAL; EMPL; FOCAL; FORTH; GPM; HI; KAPIAR; LISP; MINOL; NIBL; PASCAL; PILOT; SAM76; SCELBAL; STRUBAL
Laughlin, Neils *19/19-20*
Laurer, G.J. *14/36-43*
law *28/36-37*
LAZARUS (erased program recoverer) (6502) 26/31-33
LEDIP (text editor) (6502) 29/4-12
linkage editor. *see* relocator
LISP
for 6800 *28/24-25*
for 8080 30/4-11
list processing
article on *24/4-8*
LLL BASIC. *see* BASIC: LLL (Lawrence Livermore Labs)
loader
for 8080 3/16, 7/31
for SC/MP 21/13
Longtin, Brent *16/8-10*
LSI-11, interface to S-100 bus *11/3*

McCormick, Edward M. 19/30-35
MacDougall, John 22/42-43
machine-readable characters *15/45*
McKeeman, William M. *25/4-16*
macroprocessor
ACT (8080) 23/20-45, 24/27-46
GPM (6800) 20/22-29
KAPIAR (8080) 20/31-59, errata *23/3*
SAM76 *21/18-37; see also* SAM76
Marek, Larry A. 27/42-45
MATHPAC (trig function routines): KIMATH extension (6502) 20/15-21
math programs. *see* arithmetic
memory clearer
for 6800 29/43
for 8080 8/32-33
memory dumper
for 8080 26/18-23; 27/4-18; (vocal) 28/42-46
(BASIC) 18/41
memory tester
for 6502 17/23; 21/4-5
for 6800 5/9; 25/28-33
for 8080 8/32-33
for Z-80 22/42-43
microcomputers
in electronic toys *16/24*
microprocessor. *see* 6502; 6800; 8080; 8086; COSMAC (1802); CP1600; F8; IM6100; PACE; SC/MP; Z-80
microprocessor instructions, proposed *16/60*
Microsoft BASIC
renumbering program 27/34-38
Mills, Jonathan W. 19/21-27
MINOL (8080)
documentation and code 4/9-17, errata 6/36, 7/32
game
tiny trek 6/37, errata 7/31
Mits BASIC
modification for two terminals 9/24-25

modem
as cassette interface *10/7, 14/6-7, 19/5*
driver 26/4-16, errata *27/43*, comments *29/41-42*
reviews
DC hayes board *20/5*; 26/4-16, *27/43, 29/41-42*
IDS board 26/4-16, *27/43*
Pennywhistle 103 *17/11-12, 17/13*
Monagan, James 22/8-9
monitor
for 6502 7/20-21
for 8080 4/18-32
for COSMAC (1802) 19/30-35
Moody, Bob 16/15-16
Morgan, David J. 24/22-25
Mork, Phil 28/42-46
Morrow, George *25/40-46*
Moser, Carl 21/4-5; 27/31-33
Mueller, Erik T. 4/9-17; *14/12-16*
music
program (8080) 2/6-7, 3/6 8/32-33, 16/21
printing of *3/10-11*
bibliography of computer music *7/33-35*

networks
Digicast™ project *29/16-17*
PCNET *17/7, 17/13*; mentions of *26/4; 30/47*
other: (D.C. area) *14/3*; (EFTS) *16/61-62*; (DEAFNET) *17/10*; (bulletin board) *26/44*
Newberry, Steve 20/31-59
NIBL(SC/MP)
documentation and code 10/34-50, errata *15/4*
criticism *22/21*, response *26/45*
Northstar 28/38-41
number systems
conversions
binary to decimal (algorithm) 1/2
octal to hex *16/13-14*
hex to octal *16/13-14*
hex vs octal *19/5; 22/45-46*
miscellaneous *15/4*
Nurse, Howard L. *28/30-32*

Ockers, Stan 14/32
octal. *see* number systems
opcode decoding (6502) 17/20-22, 19/42-44; 22/5-7
operating system
CHAOS *6/3; 7/3*
ISIS (analysis of) *21/10-17*
Kernel (8080) 18/20-40
Unix™ *4/3, 6/3, 7/3*
optimizing (a program) 22/5-7
OZNAKI project 13/12-13, 24/10-11

PACE
review of PACER *9/12-13*
Pack, Charlie 4/18-32, 19/36-41
Palmer, John B. 18/41
Parker, Dan S. 14/10-11
PASCAL *23/16-19*
PCM-12 (review of) *7/10*
PCNET. *see* networks
Pearce, Craig A. *21/5*
Peterson, Richard K. 13/36-37
Pfafman, Fred *13/7*
PILOT
8080
code for 15/18-33
documentation *14/17-29*
errata *19/18, 20/63*
modifications
editor 22/43-44; 24/22-25
other 17/24
on North Star *27/26*
Common *27/46-47*
information exchange *14/17*
tiny
proposal *13/34-35*, responses *18/43, 19/35*
see also FOCAL
Pittman, Tom *6/9-10; 8/19-21; 12/3-4*
Poduska, Paul R. *20/9-11*
'polish joke' *4/18*, explanation *7/35*
Poly88 (review of) *10/16-18*
Polymorphic's BASIC
review of *14/8-10*
printer driver for 27/27-30
Polymorphic's documentation, corrections *19/15-17*
Poly VTI. 13/12-13; 16/17-19
POW(word processor) (8080) 29/18-29
printer
driver routines (8080) 14/10-11, *19/35*; 27/27-30
(6800) 12/7
Processor Technology
BASIC
INT function fix *8/34*; 10/9, 13/4; 17/26
tape load/save 16/23-24
assembler fix *13/7*; 16/22
G.P. Memory module fix *27/41*
programming languages. *see* languages, programming
programming tricks *6/9-10; 28/29*; 29/3
programs
miscellaneous
interactive peek/poke (BASIC) 22/8-9
stencil cutter (BASIC) *29/17*
see also under processor type (for machine and assembly language) or program type
Pugh, Tim 26/4-16
Purdin, Titus 28/38-41

Rankin, Roy 7/17-19
Raskin, Jef 13/25-29; *also most reviews*
reformator (intel to proc tech assembler) (8080) 6/10
relocating machine code
article on *22/10-13*
relocator
for 8080 13/8-9; 22/14-20
for 6502 14/30-31
for Z-80 13/8-9
renumbering routines for BASIC 23/4-9; 27/34-38
reviews
computers
DZ-80 *15/7*
H-8 *20/9-11; 30/22-25*
PACER *9/12-13*
Poly88 *10/16-18*, errata *13/5*
miscellaneous *8/15-17; 14/46; 15/5; 18/5; 24/19-20*
modems *17/11-12; 17/13; 20/5*
software
Mits BASIC, Poly BASIC and NIBL *14/8-10*, errata *16/7, 17/31*
other 20/29; 26/17
Rice, Lloyd *4/6-8*
Ritter, Terry 25/28-33
robots *22/4*; 18/8-19; 25/17-20
Roichel, Ancelme *21/18-37*
Roth, Richard L. 22/14-20

SAM76 *21/18-37; 23/46; 26/46-47; 30/19-21; 30/44-45*
Sanger, Joseph Jay 15/10-17

Santore, Ron 6/38-39; 16/23-24
Savir, D. *14/36-43*
SCELBAL *2/8-12*
schedule program (BASIC) 16/15-16
Schuman, Philip *17/35-37*
SC/MP
  loader 21/13
  language
    NIBL. *see* NIBL
  subroutine manager 13/36-37, 30/44
Scully, Tim 22/43-44
Selectric: converting to an I/O device *13/41-44*
Shaw, Dan M. 13/8-9
Shepard, John C. 5/10-11
Sherman, Ralph 14/30-31
Simmons, Webb *21/8-9*
Simpson, Philip 22/30-41
Skovgaard, Erik 21/13
Smith, Glendon *6/30-33*
Smith, R. Lynn *12/25-27; 16/57*
Smith, Stephen P. 21/6
Snell, John *7/33-35; 9/12-13*
software
  editorial on *12/3-4*, response *16/7, 16/8-10*
  piracy *5/3;19/20; 22/20; 25/3*
SOL(8080)
  disassembler for 21/38-46
  file routines for ALS-8 27/42-45
  patches to run Mits BASIC 18/42-45
sort routine (quicksort) *24/20*
speaker, using TV, for computer output *12/4*
speech
  digitizing 28/42-46; 29/30-33
  synthesis
    article on *4/6-8*
    reference to paper on *8/9*
    products *3/12; 5/7-8; 8/7-8; 20/6-8*
Stallings, Gordon 13/30-33
standards
  proposed
    memory map *9/22*
    S-100 bus *7/8-9; 25/40-46*
    Z-80 code *13/10-11*, response *17/34*
Starkweather, John A. *14/17-29*; 15/18-33
status push/pop (8080) 29/3
Stedman, John O. *9/16-18*
Stiefel, Mal *7/12-15*
string output routine(6502) 8/33, 10/50, 12/8
STRUBAL *23/10; 28/28, 30/44-46*
structured programming
  article on *21/8-9*
Stumpf, Adolph P. 2/34; *10/9*; 13/4
Suding, Robert *1/18*
Swank, Joel 28/33-35, 30/32-33
SWTPC
  6800 I/O control interface *17/35-37*
  miscellaneous *20/30*

Teeter, John 11/8-61
telephone. *see also* networks
telephone company *7/36-37*
terminals
  for trade *16/14*, retraction *17/10*
  Singer/Friden/Burroughs *10/9; 12/27; 15/5; 17/33*
Terry, J.M. *24/4-8*
text editor
  for 6502 29/4-12
  for 8080 6/13-26
Thompson, Bill *6/11-12*
timer routines
  for 6800 9/21
  for COSMAC (1802) 19/30-35
tiny BASIC. *see* BASIC, tiny
tiny BASIC extended. *see* BASIC, tiny: Texas
trace routine (Z-80) 23/11-12
tree, binary
  manipulation routines (8080) 30/15-17
Trenholme, John B. 27/34-38
TVT-II
  driver (8080) 2/34
  see also hardware: modifications
TV. *see* display
TV Dazzler. *see* graphics: 8080: dazzler

Univac II
  article on *28/20-23*
Unix™ (operating system) *4/3, 6/3, 7/3*
utilities *see under type of processor, computer or utility*

Van Buer, Darrel *14/6-7*; 30/4-11
Van Der Wateren, Frits *28/24-25*; 20/22-29
Van Dyke, Don 21/38-46
Van Valzah, Bob 10/19-32, *12/44-45*

Wadsworth, Nat 2/8-12
Wang, Lichen 5/12-25; 6/8; *17/33-34*; 18/8-19
Warme, Paul K. *30/26-31+*
Warren, Jim C., Jr.
  editorials *2/3; 3/3; 4/3; 5/3; 7/3; 9/3-4; 10/3-4*
  other *4/5; 8/3-6; 9/6-7; 11/7; 16/5-6; 29/16-17*
Watson, Herman 29/18-29
Wayne Watch *16/5-6*
Weisbecker, Joe *15/34-44*
Whatis (expression evaluator) 29/34-40
Whipple, Dick 1/13-17; 2/13-31
Wilcox, Bob *3/6*
Wilcox, Dick *11/3-7*
Williams, Steve 16/15-6
Williams, Tom *19/6-9; 30/22-25*
Winzenread, Marvin R. 8/28-29; 9/30-31; 10/58-59; 10/60; 15/9; 16/28-30
Witham, Steve *17/16-17*
word processor
  for 8080 29/18-29
Wozniak, Steve 7/17-19; 8/22-25; 8/26-27; 23/4-9

Yob, Gregory 22/22-29
Young, Kenneth *20/6-8*

Z-8
  article on *26/28-30*
Z-80
  differences Zilog-Mostek chips *19/9*, retraction *21/13*
  disassembler (BASIC) 26/42-44
  miscellaneous *7/11*
  patches
    for Microsoft 8K BASIC 12/20
    for Mits 12K BASIC 17/25
  utilities
    debugger 24/12-18
    memory tester 22/42-43
    miscellaneous 27/22-26
    relocator 13/8-9
    trace troutine 23/11-12
  see also 8080
Zeigler, John 3/17-19
Zolman, Leor 26/34-41

5490-4
$18.95

HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5490-4