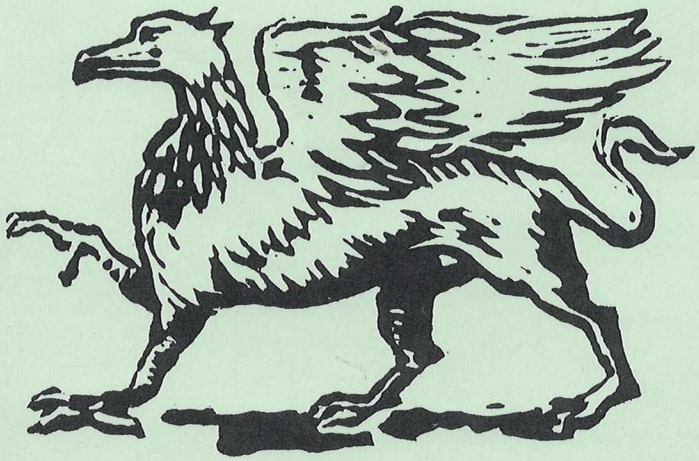
Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Two



The People's Computer Company Series

Dr. Dobb's Journal

of Computer Calisthenics & Orthodontia,

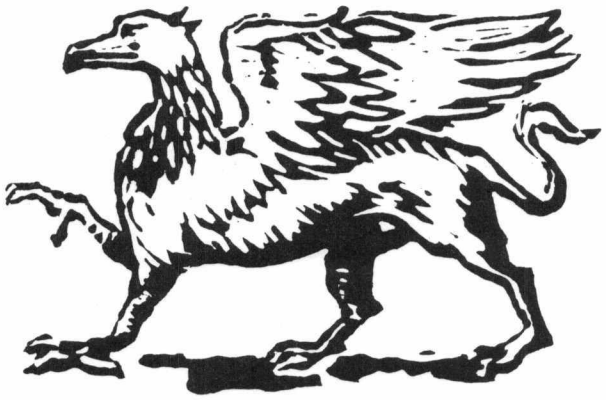
Volume Two

Dr. Dobb's Journal of COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

A Reference Journal for Users of Home Computers

Volume Two



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.

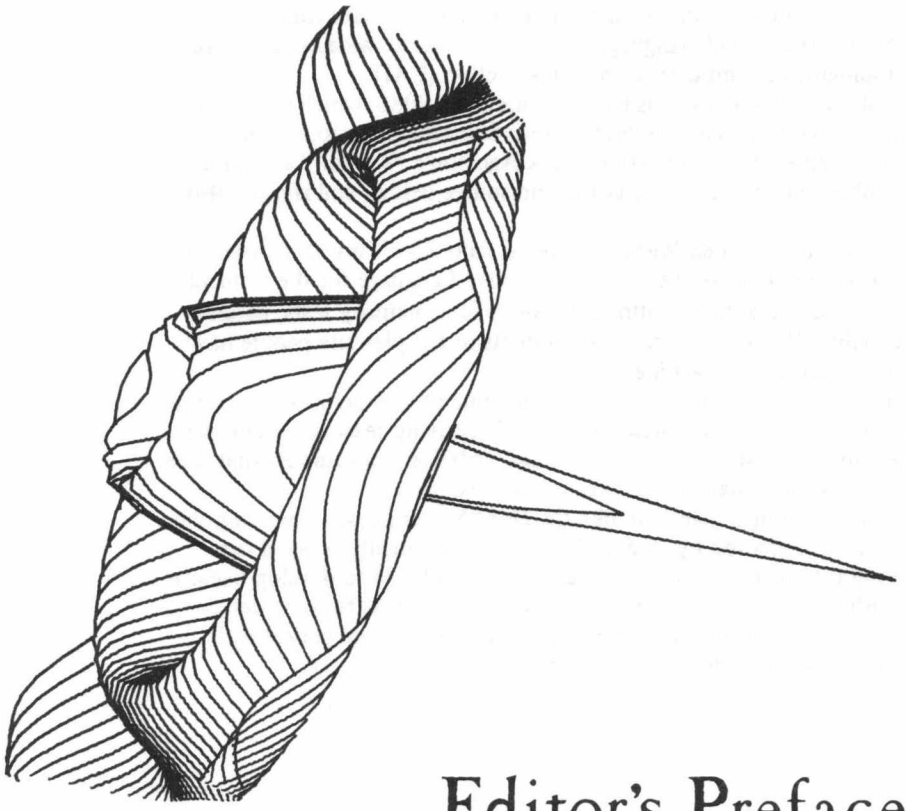
Linocut of griffin by Wendy Palmer. The mythological griffin, half lion and half eagle, has long stood as an enemy of ignorance.

ISBN 0-8104-5484-X

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

1	2	3	4	5	6	7	8	9	PRINTING
80	81	82	83	84	85	86	87	88	YEAR



Editor's Preface

The second volume of *Dr. Dobb's Journal* is being compiled quite some time after the publication of the magazines it contains. That makes it possible to write a preface with some degree of perspective. By contrast with the contents of Volumes 1 and 3, this volume provides a picture of an industry's growing pains.

Microcomputers were still very much the domain of hobbyists and technology junkies in early 1977, but, during the course of that year, computers began to emerge as useful tools for home and business. At the same time, *Dr. Dobb's* found itself acting as a consumer advocate in a market that was beginning an era of explosive growth.

In late 1977, it was my privilege to follow *Dr. Dobb's* founder, Jim Warren, as editor; I walked right into the middle of a world of dazzling machines, selfless people exploring difficult systems for the love of the science itself, and fast-talking carny barkers. Since it seemed that the field genuinely needed a publication to speak for the consumer, *Dr. Dobb's* began to publish independent product reviews and reports on delayed deliveries, and to pursue consumer complaints.

That task, however, became overwhelming. The industry was growing too fast for the reviewers to keep up with all the details of the major new products; companies were founded, and folded, overnight. To the extent of our small staff's ability, however, serious complaints were still pursued, and customers reported that our efforts helped resolve some of their problems.

Dr. Dobb's, Volume 2, chronicles the emergence of the small computer as a useful tool. Here you will find a complete source listing for Lawrence Livermore Basic, which became popular due to its implementation of a floating point math package. The quest for a standard floating point implementation is still being discussed by a committee of the IEEE, and LLL Basic directly contributed to that effort. The publication of LLL Basic also represented one of *Dr. Dobb's* primary functions – the introduction of useable software into the public domain.

In this volume, we have collected important pieces of systems software, all by authors who contributed their work for no reward other than the satisfaction gained by sharing. For ex-

ample, two issues of *DDJ* were required to present the full documentation and source code for Dr. John Starkweather's 8080 PILOT. This language is now used world-wide, and was the springboard for later developments in computer-assisted instruction (CAI).

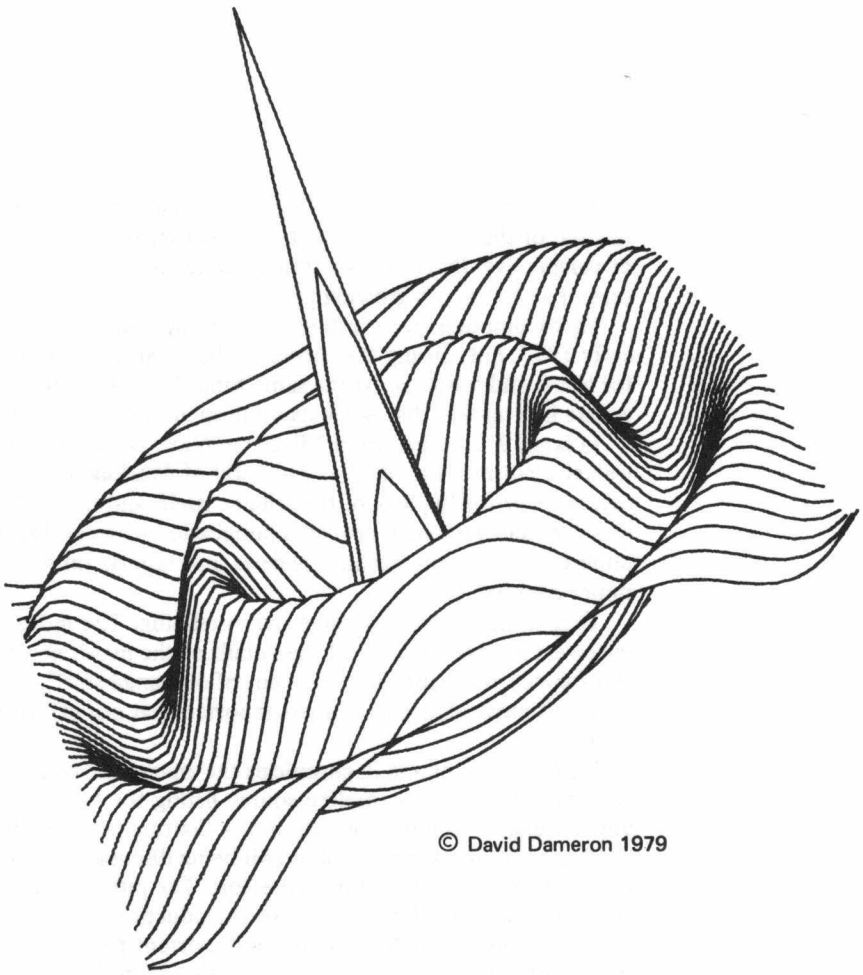
For the most part, the software in Volume 2 is based on the Intel 8080, which was the first microprocessor to come into general use with the MITS Altair 8800 microcomputer, and which emphasized systems operation rather than applications. Max Agoston's "Kernel," a complete 8080 operating system, is published with documentation and source listing, and is now, also in use world-wide.

Of course, it isn't all heavy-duty. Lichen Wang, author of Palo Alto Tiny Basic (which appeared in Volume 1 of *Dr. Dobb's*) offers the tasty "Interactive Language for the Control of Robots." Volume 2 also contains graphics routines, disassemblers, memory tests, patches and corrections to commercially offered software – all products of the pleasure people have had learning how to use these wonderous machines.

Following the course of the letters and short subjects gives some of the flavor of the debates among readers, including the importance of software authors' receiving reasonable compensation for their efforts; the endless debate about 'hex vs. octal' notation: and rumors that later turned out to be true, as well as others that turned out not to be (oops!).

The most delightful result of editing a magazine like *Dr. Dobb's Journal* was observing the intense interest of, and active participation by, the readers. I was continually amazed that people would send in material that had taken a great deal of work and care to develop because they wanted to share with others. The year 1977 produced a quantity of software which is still in use today, and which has formed the basis for a number of commercial packages. It was an unforgettable experience to have been involved in it.

Tom Williams



Contents

VOLUME 2, NUMBER 1, 1977

13 A PDP-11-Like, 16-Bit Micro for the S-100 Bus Dick Wilcox, Alpha Micro Technology
17 *DDJ* Editor Comments on the Alpha Micro System Jim C. Warren, Jr.
18 **Lawrence Livermore Lab's 8080 BASIC** John Dickenson, Jerry Barber, John Teeter, Royce Eckard, Eugene Fisher

VOLUME 2, NUMBER 2, 1977

77 **Free Software? Or – Support Your Local Software Vendor** Tom Pittman
79 String Handling for Pittman's 6800 Tiny BASIC Henry L. Kee
80 Extensions to Wang's Palo Alto Tiny BASIC Russ Hayden
81 Pittman 6800 Tiny BASIC Mods for SWTPC PR – 40 Printer Mickey E. Ferguson
82 STRINGOUT Mods for Espinosa's 6502 Routine H.T. Gordon
83 Symple Cyphering David Gruhn
86 **High Speed Interaction Without Interrupt or DMA** Eugene Fisher
92 Modifications to MicroSoft 8K BASIC 3.1 for Cromemco Z–80
95 $10 for 64-Characters & Scrolling on a TVT-II R. Lynn Smith
98 64 Char/Line TVT-II Mods, Requests & Suggestions Gary Alevy
100 **An 8080 Disassembler** Ward Christensen
114 Appendices to CASUAL Bob Van Valzah

VOLUME 2, NUMBER 3, 1977

122 Tiny Linkage Editor for "relocatable code" on 8080's and Z-80's Dan M. Shaw
124 Proposal for Standard Z-80 Code Peter G. Formaniak
126 **Graphics on the Poly-88** Harvey A. Cohen
129 **Extended Accuracy Arithmetic Routines** M.G. Dineley
137 Algorithm for computing square roots James A. Hinds
139 8080 Disassembler Written in MITS 3.2 BASIC Jef Raskin
144 M6800 Disassembler Gordon Stallings
148 Proposed Tiny PILOT Dennis Allison
150 An SC/MP Subroutine Supervisor Richard K. Peterson
152 **The Selectric Typewriter as An Output Device** Rod Graves

VOLUME 2, NUMBER 4, 1977

159 Washington Club Offers Free Access to Their On-Line Journal
162 Pros & Cons of Modems for Cassette Recording
162 Using an Acoustic Coupled Modem as a Bulk Storage Interface
164 **MITS Basic, Poly Basic, and NIBL – Product Review** Jef Raskin
166 Screen Image Hard Copy for the VDM-I Dan S. Parker
168 Here, Now: A Micro APL for 8080's Erik T. Mueller
173 **Guide to 8080 PILOT, Version 1.1** John A. Starkweather, Ph.D.
186 A 650X Program Relocater Ralph Sherman
188 A KIM-1 TV Sketch Program Stan Ockers
190 The Characteristics and Decodability of the Universal Product Code Symbol D. Savir & G.J. Laurer

VOLUME 2, NUMBER 5, 1977

203 What to Get Your Computer for its First Birthday Jef Raskin
205 Testing the DZ80-80 CPU: Nifty Z-80 Conversion for Altair & IMSAI Computers Jef Raskin
207 Jack Armstrong's Super Decoder Ring – Revisited Marvin R. Winzenread
208 Video Chase for 8080/VDM Joseph J. Sanger
216 **Source Code for 8080 PILOT, Version 1.1** John A. Starkweather, Ph.D.
232 A Practical, Low-Cost Home/School Microprocessor System Joe Weisbecker

VOLUME 2, NUMBER 6, 1977

245 Wayne Watch: A New *DDJ* Featurette for the Naive Novice Jim C. Warren, Jr., Editor, *DDJ*
246 **Which Manufacturer Offers the Longest Delivery Delay?** a pseudo-contest
247 Response to Tom Pittman's Editorial Dave Allen
248 More Support for Software Vendors Brent Longtin
252 A Story of Traffic Lights, or: A BASIC Program to Play the Game of Life R. Broucke
253 Conversions Between Octal and Hexadecimal R. Broucke
255 **SCHDL: A BASIC-coded 'Daily Reminder"** Bob Moody & Steve Williams
257 Curve Plotting on a TV Screen with the Polymorphics Video Interface and MITS BASIC R. Broucke
259 6800 Circle Maker David Hudson
261 Dompier Plus: A "Musical Keyboard" Bruce Hicks
262 Permanent Solution to Missing SP and PSW Definitions in PTC Sys Bruce R. Kendall
263 Tape LOAD and SAVE Routines for P.T.'s BASIC Ron Santore
265 KIM-1 Breakpoint Routines: Plain & Fancy Willi Kushe
268 A 'Turtle' for VDM Marvin R. Winzenread
271 **An Example of an M6800-Based GPIB Interface** S.C. Baunach
296 Errata: TVT-II Mods had Pin Numbers Permuted Christopher Terry
297 Lower Case Mod for SWTPC's CT-1024 R. Lynn Smith

VOLUME 2, NUMBER 7, 1977

301 SCCS Group Buy – Complaint & Response
302 About Those ASCII TTY's That You *Cannot* Trade For . . . David Jon Fylstra
303 **Some Dire Warnings of a Modem** Jef Raskin
305 **Pennywhistle Plays a Better Tune** Jef Raskin
306 VDM-1 Modifications Lichen Wang

308 Dissecting the SWTPC Control Interface Phillip Schuman
310 PCF is Center for Personal Communications Law Kenneth S. Widelitz
311 Towards a Good μ-C Language Joshua P. Gordon
312 On the Effects of Filling Cavities Within the Fillings of Cavities Within . . . Steve Witham
314 **The APL Core Project** Texas A&M APL Committee
316 Decoding 650X Opcodes H.T. Gordon
319 A High-Speed Memory Test Program for the 6502 Jim F. Butterfield
320 Mods to 8080 PILOT Steven Spence
321 Z-80 Patches for MITS Basic Martin D. Gray
322 One-for-One Patch to P.T. 5K Basic Morris G. Miller
322 Real-Time Modification for Sanger's Video CHASE David J. Morgan
326 An 8080 Tracer John Walker
327 Friend Finds Factorial Flaw Jef Raskin

VOLUME 2, NUMBER 8, 1977

331 About Those Elusive PILOT Tapes – Ronald C. Fryxell reply by Gregory Yob
332 Some Follow-Ups and A Few New Tidbits from Readers Jef Raskin
333 **Computer Applications for the Handicapped** Warren J. Dunning
334 AN INTERACTIVE PROGRAMMING LANGUAGE FOR CONTROL OF ROBOTS
Lichen Wang
346 **A MICROPROCESSOR OPERATING SYSTEM: THE KERNEL**
Max K. Agoston, Ph.D.
367 Basic-Coded Memory Dumper John B. Palmer
368 Patching Microsoft's 4.0 Basic on P.T.'s SOL Jack L. Calaway

VOLUME 2, NUMBER 9, 1977

376 **A Detailed Report on Product Delivery Complaints** Tom Williams, Assistant Editor
380 The Heath H-8: Pro & Con
385 Poly Doc Corrections Richard B. Emerson
389 Dijkstra's Minilanguage Neils Laughlin
391 **GRAFX – A Graphics Monitor for the 8080/TV Dazzler** Jonathan W. Mills
398 Alphanumeric Plotter for Poly Video Display Arthur Armstron
400 Utilities & Music on the Cosmac Elf Edward M. McCormack
404 PROMable 1K Operating System for RCA's Elf Edward M. McCormack
406 Multiple Column Accounting Charlie Pack
412 OPLEGL Correction & a 6502 Scanning-Debugger H.T. Gordon

VOLUME 2, NUMBER 10, 1977

420 New Voice Synthesis & Graphics Products Kenneth Young
423 Heathkit's H-8: A Consumer's View Paul R. Poduska, Ph.D.
426 A KIM-1 Disassembler Theodore E. Bridge
427 VDM-1 Driver Mods John Moorhead
429 **MATHPAC: A Kimath Supplement** John Eaton
436 GPM for the 6800 Frits van der Wateren
443 ZAPS Operating System and Assembler for the Z-80 Jef Raskin
444 SWTPC Editor Format Update Phil Hughes
445 **KAPIAR: AN 8080 MACROPROCESSOR** Steve Newberry

475 **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 11 **January 1977** **Volume 2, Issue 1**

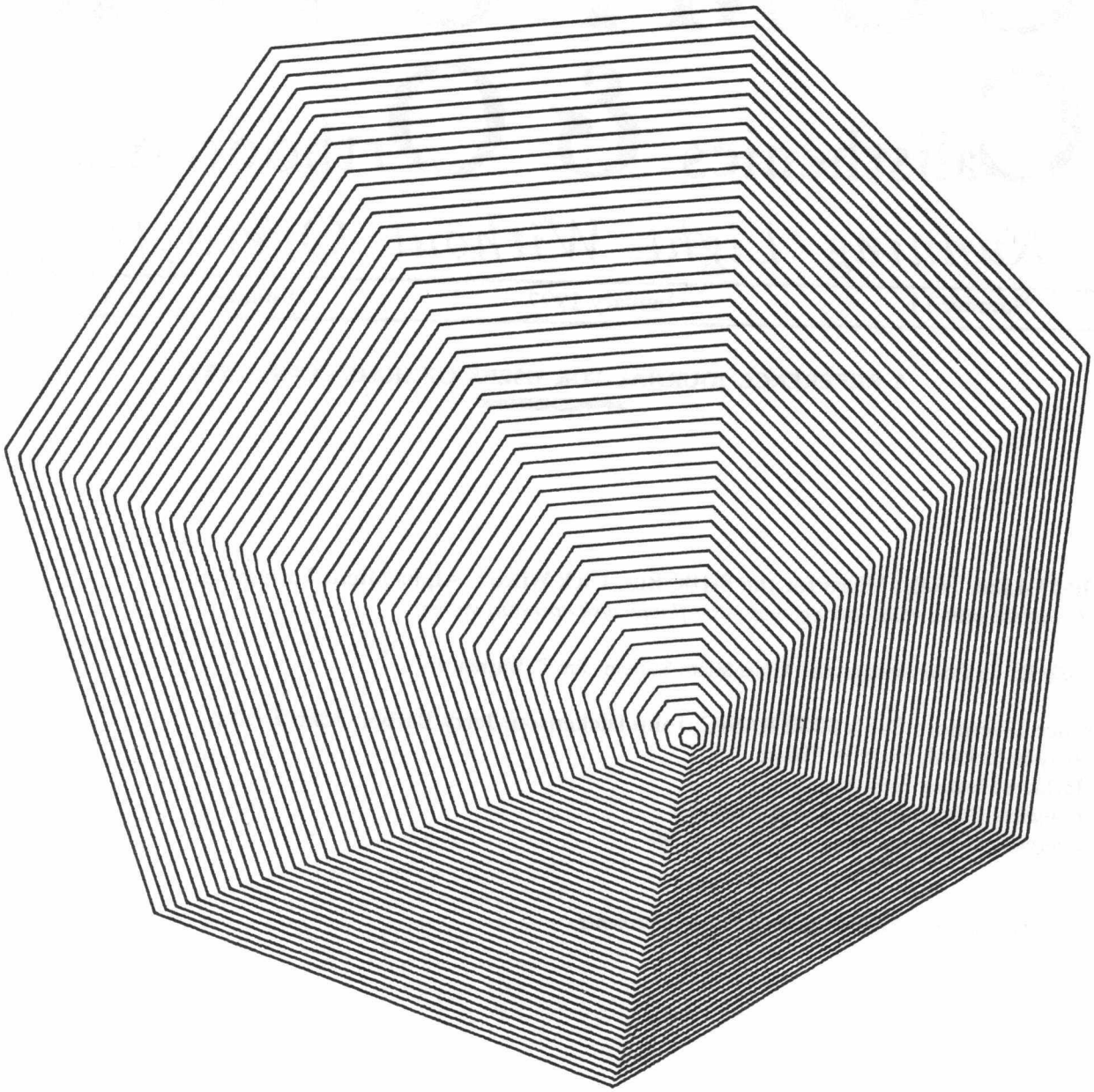
A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

A PDP-11-Like, 16-Bit Micro for the S-100 Bus, Dick Wilcox, Alpha Micro Technology 3
Complete with a multi-tasking, multi-user floppy disk operating system

DDJ Editor Comments on the Alpha Micro System, Jim C. Warren, Jr. 7

Lawrence Livermore Lab's 8080 BASIC, John Dickenson, Jerry Barber, John Teeter, Royce Eckard, Eugene Fisher 8
Includes Floating Point Package
Complete User Documentation & Examples
Complete, Assembled Annotated Source Code

- - and a few, miscellaneous hardware announcements - -



© David Dameron 1979

A PDP-11-Like, 16-Bit Micro for the S-100 Bus

16 BITS FOR YOUR IMSAI OR ALTAIR COMPUTER

by Dick Wilcox, 1342 Mauna Loa Road, Tustin, CA 92680
(714) 838-1381

As I was waiting for my computer to finish an assembly, I happened to be glancing over my September issue of *DDJ* when I came across your small but interesting article way down down in the corner of page 11 entitled, "HOT IDEA". In it you suggested that someone should interface the LSI-11 board to the S-100 bus used in the IMSAI and ALTAIR hobby computers. I would like to acquaint you with a recent development of mine which should prove to be of interest to the more advanced hobbyists who have run out of flexibility in their 8080 systems.

I have worked with the PDP-11 series ever since the original design first came out on the market (the now extinct PDP-11/20) and have gotten quite used to working with a full set of 16-bit registers. To drop back to an 8-bit single accumulator machine seemed to be intolerable and so I investigated just what you suggested. I have done extensive software and hardware interfacing to several of the PDP-11 models and after some study, came up with the conclusion that the low impedence bus structure of the LSI-11 was not easily adaptable to the S-100. In addition, the board size pretty much eliminated the incorporation of it directly into the actual computer box which all-in-all cast a dim shadow over the whole idea.

The only feasible solution to this problem seemed to be a totally new design based around some already existing microprocessor. The system I selected is the Western Digital MCP-1600 which is comprised of two system chips (40-pin) and from one to four microprogrammable ROMS (40-pin) which may be custom programmed to perform various macro-level instruction sequences. Working together with a personal friend, Rich Notari, I came up with a workable instruction set that could exist in the MCP-1600. Rich then took this list and added a few frills of his own based on some previous experiences with this particular chip set. A note of thanks is due at this point to him for his undying efforts and late hours for I feel that he is the only programmer I know who could have successfully completed the microprogramming of this system in the short time frame allotted. We have named this particular chip set (4 chips total) as it is currently microprogrammed, the WD-16 microprocessor system.

This system is then interfaced to the S-100 IMSAI/ALTAIR bus using some 70-plus TTL logic chips which fit totally on two standard sized S-100 interface PC boards. These cards plug directly into the motherboard of your computer and include as standard features, multi-level interrupt steering logic, priority DMA channels (up to eight of them), and in interrupt-driven real-time clock for timesharing job control or program timing. The real beauty of this new system, however, lies in the software which is now running on it (notice I said *now*). The complete unit including the WD-16 microprocessor and associated TTL interface is named the CM-16 microcomputer and is currently running in two IMSAI computers, undergoing rigorous testing at my house and at Alpha Micro Technology where much of the development was done. I am working in depth with the guys there to insure that the unit will be compatible with the majority of the currently available memory and peripheral devices on the market. So far the computer is performing admirably with no modifications necessary to any of the devices we have tried it out with.

Microprocessor Structure

The microprocessor is basically a 16-bit structured device which has been interfaced to the 8-bit IMSAI bus in a method which makes it transparent to the programmer. There are a total of eight registers (each 16-bits) which are used as a program counter (PC), a stack pointer (SP), and 6 general purpose work registers (R0-R5). The work registers are all identical and may be used for memory addressing, accumulators, or index registers. Clock cycle for the chip set is 2Mhz to make it compatible with current 8080 devices although the unit is capable of speeds up to 3.3 Mhz with some modifications. Each machine instruction is 1, 2, or 3 words in length depending on the addressing modes in use by the instruction. Instruction formats include control instructions, single operand instructions, and monitor call instructions. Instruction operands can be register data or memory data which may be referenced directly (no register involved) or indirectly (through one of the registers). The stack may be specified as any contiguous block of memory and any operand may be directly referenced (not just the one on the top). In addition to the specified stack register, any of the six general purpose registers may also be used as stack registers for multiple-stack compiler applications. Memory-to-memory operations may be done directly throughout the entire addressing range of 65K bytes (32K words) without disturbing any of the general purpose registers. These instructions include add, subtract, logical and, logical or, and logical bit testing.

Addressing of memory is normally done as either full 16-bit words or as single 8-bit bytes. Most instructions allow either mode of operation, thereby giving very flexible character-string handling without sacrificing arithmetic capabilities. The top 256 bytes of the addressing range (memory locations FF00-FFFF) are translated by the interface logic into 8080 I/O instructions and presented to the bus as such with the proper signals wiggling at the proper times. This eliminates the need for separate I/O instructions and most memory reference instructions may be used for I/O transfers with certain clearly defined restrictions. For all other memory access operations, the logic cleverly translates the 16-bit processor requests into a couple of ordinary old 8080 8-bit requests and both ends of the string are quite happy with the arrangement.

Interrupts are handled on a single level with up to eight hardware selectable priorities, totally compatible with the 8080 interrupt scheme. The user has the option of running with either vectored or non-vectored software handling of the interrupt request. Vectored interrupts merely use the device interrupt code (0-7) as a table pointer to develop an absolute address to the specific interrupt handling routine. DMA transfers require some slight modification to the device board since the DMA structure uses some of the currently spare lines of the 100 bus structure. The 16-bit unit (dubbed the CM-16) resides in the computer in complete harmony with the existing 8080 CPU processor and uses a single control line (PRDY) to perform the hardware switching. At any one time, one of the processors is using the bus and the other processor is tri-stated. This method avoids making the 8080 CPU ob-

solete and allows 8080 programs to be loaded under control of the extensive CM-16 software operating system and then executed using the 8080 CPU. At the present time, this switching is done mechanically but could be done via a software controlled output port if care was taken in the software timing.

Available Software

All of the software about to be described is currently running on my prototype units with the exception of the Extended BASIC Compiler system which is still undergoing development. The operating system is one which I have been using (in general design) on several computers over the past several years and has proven to be extremely useful in implementing new systems. I have recoded the various modules of it into the CM-16 macro instruction set and am running it with major success in all areas tested. Since most of the system is generalized in nature and is implemented in a very extensive macro defined language, translation to the CM-16 code was fairly easy. The basic structure of the command language used by the operating system and the file system used for disk management greatly resembles the Decsystem-10 operating system which I feel is one of the most powerful and easy to master.

Operating System

The operating system (or monitor) may be generated in one of several basic cofigurations. To gain all of the features inherent in the design of the command language, the operating system should be based on disk (either floppy or hard surface) where the loading and execution of programs is handled automatically by the system. This disk-based version contains all routines for file handling on the disk and other peripheral devices (including memory). It can be structured for real-time processing and full timesharing using fixed memory partitioning which is dynamically allocated during run-time. The timesharing routines include priority processing methods and dynamic device reassignment for non-sharable devices. A line printer spooler is included which runs in its own dedicated jobstream for those users lucky enough to have laid their hands on one of these expensive beasties. The operating system may also be generated in a much limited version which does not require a disk and allocated space in RAM dynamically for program and data files. In this version the user must explicitly load the run programs which perform most of the system tasks from some peripheral device such as a cassette tape.

The operating system includes all facilities for logical I/O handling on a device-independent basis. All programs are written using logical calls such as OPEN, READ, WRITE, CLOSE, etc., and the monitor assigns input and output devices (or files) at run time under operator control. This means that when the user purchases his new random access Magnetic Wimbis device, all he has to do is write a small driver for getting data into and out of the Wimbis and all of his programs (including all system programs) will operate with said bizarre device. Writing this driver is done in macro assembly language using a set of predefined rules. An extensive terminal service package allows various different devices to be defined as system consoles or peripheral I/O devices and interact with the operating system using specific control codes necessary for such things as carriage position and cursor control. I feel that with the myriad of different peripheral devices on the market (including surplus items) it is important to offer an easy method to implement these devices without having to rewrite all programs.

Command Language

The command language implemented in the CM-16 operating system offers the user the ability to easily define his own commands without modification to any of the system software. It works like this: when a command (1-6 characters) is typed on the terminal, the operating system searches for either a program by that name or a command file by that name. These programs and commands may be stored in a common system library or in the user's own area Searching is performed in a manner defined when the user builds his operating system. If a program is found by that name it is automatically loaded into memory and executed. In other words, the file renaming functions are performed by a program called RENAME.PRG which is part of the system utility package. When the user types RENAME on his terminal the program is executed and performs the function. If the command results in the location of a command file that file is loaded into memory and the lines in that file are then fed into the terminal input stream one at a time just as if they had been typed on the terminal in sequence by the user. Each command file is ASCII and is created and edited by the text editor just as any source file or data file would be. Lines within the command file may include program name commands, other command file commands, or input data required for the execution of the program called. Specific control codes (preceeded by a colon) allow the control of echoing, manual keyboard input for specific data, and operator messages within the command file processing.

Disk Data Organization

The disk (or multiple disks) is dynamically allocated as files are built using a bitmap technique which eliminates the need for a compression phase when the disk becomes fragmented due to file deletion. Each time a file is opened it eats up only as much space as it actually requires. Records within the file are linked in sequence and need not be contiguous except for special files which the user may optionally preallocate to a given size. When the file is deleted, all the space it occupied becomes imediately available for another file to use. On the larger scale, the disk may be allocated under one or more user areas with each user having a separate account number and an optional password (1-6 characters) which is required at login time. Each user area is protected from other user's vandalism by file protection codes and account numbers. Filenames need only be unique within each user area eliminating the necessaity of keeping track of who's doing what with which, etc. User areas need not be allocated when they are set up since space on the disk is allocated on a first-come, first-serve basis. Utility routines assist in creating and maintaining user accounts and files in the system area. The system area is merely another user area and also may dynamically change as required.

All input and output files and devices are referenced by a general format:

device:filename.extension[ppn]

The device code is a 3-character alpha name given to the peripheral unit to be referenced. This may be optionally followed by a drive number for units which may have more than one drive such as disk or tape. Typical assigned names might include DSK for disk, TAP for cassette tape, LPT for line printer, and TTY for terminal devices. DSK0 and DSK1 would therefore be the device codes for two individual drives connected to the same disk controller. These names are assigned by the user when he writes or renames the associated device driver routine. The exception to this is DSK which must be the name of the system disk when the disk operating system is used. The filename.extension follows the standard Decsystem-10 convention of using from 1-6 alphanumeric characters for the name of the file

and a 1-3 character extension (or modifier) which usually specifies the type of file it is. For example, given the program name of CONVRT for a conversion program, the following standard extensions are used by the system:

CONVRT.MAC	-	macro language source file
CONVRT.BAK	-	backup source file created by the editor
CONVRT.OBJ	-	object file created by the assembler
CONVRT.PRG	-	runnable program file created by the Link loader
CONVRT.SYM	-	symbol file used by the symbolic debugging program

The ppn within brackets represents the user account number (also called project-programmer number, hence ppn) when reference is to another user area. Filenames and ppns are used only for file structured devices and are ignored for non-file-structured devices such as a printer or paper-tape reader.

Some examples of dataset references are:

LPT:	– Line printer
DSK:CONVRT.MAC	– File CONVRT.MAC in current user area on disk
DSK1:CONVRT.PRG[24,5]	– File named CONVRT.PRG in user area 24,5 on disk 1
TTY:	– Current user terminal device

All routines for processing these dataset references and setting up the links to the correct devices and files are built into the resident operating system thereby relieving the programmer of that task. The programmer merely sets up a 20-word dataset definition block (DDB) and then uses a dozen or so monitor calls which reference this DDB to perform all I/O setup and accessing operations.

Assembly Language Development System

The assembly language programming fanatic (long may he live!) has plenty of support in the CM-16 development system. The assembler is an extensive macro-level one with complete macro definition facilities and conditional assembly pseudo-operators. Internal and external global reference statements allow a large program to be segmented into little hunks and assembled separately while allowing inter-segment data and procedure references. A linkage loader is then called to link all the object files together, resolve all external references, and generate a runnable program file. The resultant program may be totally relocatable without relinking if certain programming standards are adhered to due to the addressing capabilities of the hardware. A copy statement in the assembler allows system definition libraries and standard routines to be included in the source program with a single statement. These libraries and routines are normal ASCII source files and may reside in a common system area (DSKO:[7,7]) or in the user area. A symbolic debug program (DDT) allows the user to test his program under the supervision of a controlling program. Using DDT the user may set breakpoints, single-step execution, and inspect data at any point during execution. All data areas and program instructions are referenced by the symbols used in the source program so the programmer need not work in that diabolical numbering system known as hexadecimal. This DDT basically parallels the program by the same name in the Decsystem-10 for those of you who are familiar with it. Input to the DDT program is the program file (.PRG) and the symbol file (.SYM) which was created by the symbol file creation program which interprets the assembler output object file (.OBJ) as required. As a final frosting on the cake, there is a segmentation system in the assembler and the monitor which allows a program to be broken into logical segments and executed in a much smaller amount of memory by having each segment overlay the preceeding one.

BASIC Compiler and Run System

As I mentioned previously, this is the main piece of software which is not totally complete at this time, although it is sufficiently up and running for many test programs to be demonstrated. I have implemented the language in a kind of unique way which makes it a true compiler but which allows it to look like an interpreter in the development code. The source code may be edited via the standard text editor as a file on disk or it may be edited using the line editor built into BASIC which follows the standards used by the other popular BASIC systems on the market. The program is automatically compiled once when the RUN command is executed and the object code (actually a series of Polish-mode driver codes) is built in memory and then executed. This compiled code may optionally be stored off on disk and later recalled for execution, requiring only the run-time package to be resident.

The language elements implemented (and to be implemented) include most of the currently available functions found in the popular extended BASIC interpreters. Timings taken on some various programs indicate timings on the order of 5-7 times as fast as the current version of MITS 8K BASIC depending on the amount of math being performed and the types of variables being used. String functions are fully supported as in integer and double-precision arithmetic. I also intend to include some fairly extensive assembly language and subroutine linkages to enable the high-speed routines to be implemented where needed. Disk file data access methods are also supported along with matrix operations.

I have also included some rather unique features which are not a part of the standard BASIC format but which should provide the sophisticated user some additional flexibility. Variable names may be any length alphanumeric and are not limited to one or two characters. A redefinition scheme allows you to define any name as a command or set of commands to be referenced later in the program or in the terminal command system itself. For example, the name TOM could be equated to the multiple command sequence PRINT "TOM IS ALIVE":GOTO 230 and each time TOM was entered into the program, the above sequence would replace it. This is quite similar to the macro definition facility commonly found in assemblers. A copy statement allows external routines to be included in the source program when the compile mode is in use. There are also some other minor functions to numerous to mention here. If I could just find a function that would automatically document all the other features I would be in fat city.

Text Editor

For the Decsystem-10 fans, the text editor is basically a limited version of the very popular TECO program. It is a character-oriented free-form ASCII file editor with commands to add, delete, and insert both line and character strings in a text buffer. A pointer keeps track of your position within the text buffer and is manipulated by commands which move it forward and backward by a fixed number of lines or characters. Search commands may be used to locate a specific character string and leave the pointer set immediately after it. An iteration scheme allows a series of editor commands to be enclosed in angle brackets and repeated a predetermined number of times. There are 26 registers (A-Z) which may be used for the holding and moving of varia-

ble lengths of text or numeric values including the pointer location. The text editor may be used to create and maintain source files, command files, and data files (ASCII).

System Utility Programs

To supplement the creation, maintenance and operation of the system there exist a number of utility programs which perform specific functions. I will describe briefly a few of the more important ones. Probably the single most important one is the monitor generation program which allows the user to tailor the monitor to his specific needs and create a runnable monitor which includes selected routines and may optionally include some user written functions for fast access. The system allows the storing of several different flavors of monitors which may be called upon at any time for specific operations. Another important utility is one which creates and maintains the user accounts on the various disks. Accounts may be added and deleted and passwords may be changed. This program is also used to initialize a new disk removing all files and accounts and freeing up all the blocks. An extensive disk analysis program may be run which reads the entire disk structure and plots out the allocation of all files. Checks are made to insure that no two files are accessing the same physical disk record or records reserved for system use (such as the master directory record and bitmaps). All files and directory records are read and checked for errors. The bitmap is analyzed for records marked in use but not in a file and vice-versa. The bitmap is then recreated and rewritten onto the disk. This is an effective garbage-collection program which is very helpful in recovering from major disk failures which may destroy part of the file structure.

There also exists the usual collection of support routines for copying files, dumping memory and disk data, print directory and file listings, and monitoring system status and operations. Most programs are accessible to all user accounts but a few of the programs are privileged in the respect that they may be run only when the user is logged under the system manager's account number of 1,2. This prevents the non-informed users (such as wives and kids) from inadvertantly eating up their own system disks. Now please don't go and write about the intelligence levels of wives versus husbands, since I was only kidding. Really, some of my best friends are wives!

8080 Cross Assembler

For those of you who have some 8080 assembly language programs and do not wish to convert them immediately I have have a full macro and conditional assembly system for the standard 8080 instruction set which will accept 8080 source code and generate 8080 runnable programs. This development may be done using all the flexibility of the CM-16 operating system whereby the source file and object file may be stored on disk (or other peripheral). The runnable program may then be loaded into memory using a special loader and the processor switched to 8080 mode to run the program. Since both microprocessors physically share the bus this is not a major inconvenience and makes for a pretty nifty little 8080 development system also.

Current Progress of System Testing

As I mentioned previously, this system is currently running and not just a pipe dream and I invite anyone interested to come down to Alpha Micro for a live demonstration. The current system is running in an IMSAI computer with 40K bytes of memory of various types, a dual ICOM Frugal Floppy disk unit, a Tarbell cassette, and a 3P+S connected to a Lear Siegler Dumb Terminal. The memory requirements for the software operating system range from about 8K for a limited version to around 14K for the full timesharing disk-based one. We will be recommending around 24K as a minimum memory size which includes space for user program assembly and BASIC compilations. Due to the power of the microprocessor itself and the extensive software available, I do not feel that the main marketplace will be the limited memory systems and with the price of memories dropping constantly, the tradeoffs involved in limiting the monitor to 4K bytes or less is not worth it.

We are testing the prototype unit with various memories and peripherals to insure compatibility with most of the more popular ones. The following is a list of the devices that we have tested with the CM-16 with complete success:

Seals 8K Memory
Processor Tech 4K and 8K Memories
IMSAI 4K Memory
Solid State Music 4K Memory
IMSAI SIO-2 Serial I/O Interface
IMSAI PIO-4 Parallel I/O interface
Cromemco 8K Bytesaver
Processor Tech 3P+S I/O Interface
ICOM Frugal Floppy Disks
Tarbell Cassette Interface
Lear Siegler ADM-3 Dumb Terminal
Teletype ASR-33 Terminal
Processor Tech VDM-1 Video Display

Marketing Approach

The finished CM-16 computers are scheduled for delivery on January 15, 1977 and will be distributed exclusively by Alpha Micro Technology of Tustin, CA. The unit will be sold as a kit with the price being somewhere under $999 (does that sound better than $1000?) which will include all software mentioned above. The software will be distributed in one of several standard versions in runnable binary format on either floppy disk or cassette tape media. I firmly believe that the hobby market should be basically just that and therefore we intend to make all software available for as little as possible to cover handling and a small profit margin. I have done commercial consulting for over six years and feel that companies making a profit from my software can well afford to pay top prices for it. The hobbyist, on the other hand, merely wants to enjoy his toys and perhaps gain some real insight into the world of computers while doing so. I see no threat to my own existence there and intend to make every effort to be personally responsive to all questions and comments concerning the CM-16 software library and also intend to maintain some form of update service to users who care to get all the latest improvements as they come out. The actual cost of the software system (included in the price of the CM-16 kit) is somewhere around $75.00 for the runnable binary system. This will cover the license to use that software for non-profit only. Commercial users desiring to OEM either the CM-16 computer or the WD-16 chip set alone will be asked to pay a nominal licensing fee for each version sold.

An unusual approach not commonly found in systems of this magnitude is that we intend to sell all source files and listings in selected groups for a similarly nominal price. They will be distributed on floppy disks and will be modifiable by the user if he so desires. I intend to also distribute a fairly complete set of manuals giving information on writing your own monitor routines and peripheral drivers. Since much of the software is modular in scope, many of the routines used to build the monitor, assembler, and compiler will be available for the user. I have seen the enthusiasm that the hobby market has produced and feel that the feedback that I receive on my software will more than offset any additional profit to be made by overpricing.

I hope that this brief description will spur the interest of some of you out there and I invite you to write or call me for further information on this new microprocessor system. Most of my time lately has been spent chasing gremlins and I welcome the opportunity to engage in some casual conversation now and then with other hobbyists interested in advancing the field of computers. You can find me chasing gremlins at the following location:

Alpha Micro Technology
1342 Mauna Loa Road
Tustin, CA 92680

```
Supervisor Call Instructions:
     SVCA  arg          Jumps to a monitor routine defined by a table base
                        address in location 22 plus twice the argument value
     SVCB  arg          Pushes all registers onto the stack including the SP
                        and PC and then jumps to a monitor routine as in SVCA
                        getting the table base address from location 24
     SVCC  arg          Same as SVCB using table from location 26

Immediate Mode Instructions:
     ADDI  imm,reg      Adds the immediate data (range 1-16) to reg
     SUBI  imm,reg      Subtracts the immediate data (range 1-16) from reg
     MOVI  imm,reg      Moves the immediate data (range 1-16) to reg
     BICI  imm,reg      Clears the bits in reg corresponding to immediate data

Single Address Instructions:
     ROR   dst          Rotates dst right one bit through the carry flag
     RORB  dst          Rotates dst byte right one bit through the carry flag
     ROL   dst          Rotates dst word left one bit through the carry flag
     ROLB  dst          Rotates dst byte left one bit through the carry flag
     TST   dst          Tests the dst word and sets the CPU flags per the result
     TSTB  dst          Tests the dst byte and sets the CPU flags per the result
     ASL   dst          Shifts the dst word left one bit into the carry flag
     ASLB  dst          Shifts the dst byte left one bit into the carry flag
     SET   dst          Sets the dst word to all ones
     SETB  dst          Sets the dst byte to all ones
     CLR   dst          Clears the dst word to zeros
     CLRB  dst          Clears the dst byte to zeros
     ASR   dst          Shifts the dst word left one bit into the carry flag
     ASRB  dst          Shifts the dst byte left one bit into the carry flag
     SWAB  dst          Swaps the two bytes in the dst word
     SWAD  dst          Swaps the two hex digits in the dst byte
     COM   dst          Compliments the dst word (one's compliment)
     COMB  dst          Compliments the dst byte (one's compliment)
     NEG   dst          Negates the dst word (two's compliment)
     NEGB  dst          Negates the dst byte (two's compliment)
     INC   dst          Increments the dst word by one
     INCB  dst          Increments the dst byte by one
     DEC   dst          Decrements the dst word by one
     DECB  dst          Decrements the dst byte by one
     IW2   dst          Increments the dst word by two
     SXT   dst          Extends the sign bit (N flag) through the dst word
     LSTS  dst          Loads the processor flags and interrupt from the dst word
     SSTS  dst          Stores the processor flags and interrupt into the dst word
     TJMP  dst          Table jump based on data in dst and table following the inst
     TCALL dst          Table subroutine call similar to TJMP
     ADC   dst          Adds the carry flag to the dst word
     SBC   dst          Subtracts the carry flag from the dst word

Double Address Instructions:
     ADD   src,dst      Adds the src word to the dst word
     SUB   src,dst      Subtracts the src word from the dst word
     AND   src,dst      Logical AND's the src word into the dst word
     BIC   src,dst      Clears bits in the dst word corresponding to the src word
     BIS   src,dst      Logical OR's the src word into the dst word
     BISB  src,dst      Logical OR's the src byte into the dst byte
     BIT   src,dst      Sets processor flags based on corresponding src-dst word bits
     XOR   src,dst      Logical XOR's the src word into the dst word
     MOV   src,dst      Moves the src word to the dst word
     MOVB  src,dst      Moves the src byte to the dst byte
     CMP   src,dst      Compares the src word to the dst word and sets flags
     CMPB  src,dst      Compares the src byte to the dst byte and sets flags

Register-Address Instructions:
     CALL  dst          Call the subroutine at dst (PC pushed onto stack)
     CALL  reg,dst      Call the subroutine at dst using reg as link register
     LEA   reg,dst      Loads the effective address dst into reg
     JMP   dst          Special form of LEA which jumps to dst routine
     XCH   reg,dst      Exchanges the words in the reg and dst
     SOB   reg,dst      Decrements reg by one and branches to dst if result not zero
     ASH   reg,dst      Shifts the word in reg by (dst) bits (left if +, right if -)
     ASHC  reg,dst      Shifts the double-word unit  in reg,reg+1 same as ASH
     MUL   reg,dst      Multiplies reg by dst leaving result in reg,reg+1
     DIV   reg,dst      Divides reg,reg+1 by dst leaving result in reg (remainder in
                                                                        reg+1)
Branch Instructions:
     BR    adr          Branch unconditionally
     BEQ   adr          Branch if equal
     BNE   adr          Branch if not equal
     BPL   adr          Branch if plus
     BMI   adr          Branch if minus
     BHI   adr          Branch if higher (unsigned)
     BHIS  adr          Branch if higher or same (unsigned)
     BLO   adr          Branch if lower (unsigned)
     BLOS  adr          Branch if lower or same (unsigned)
     BGT   adr          Branch if greater (signed)
     BGE   adr          Branch if greater or equal (signed)
     BLT   adr          Branch if less (signed)
     BLE   adr          Branch if less or equal (signed)
     BVC   adr          Branch if overflow flag clear
     BVS   adr          Branch if overflow flag set
     BCC   adr          Branch if carry flag clear
     BCS   adr          Branch if carry flag set
```

DDJ EDITOR COMMENTS ON THE ALPHA MICRO SYSTEM

by Jim C. Warren, Jr.

I saw this widgit running at a December meeting of the Homebrew Computer Club, and was totally blown away by it! It comes close to being an order of magnitude more potent than any other system I have seen. In prototype form, I watched John French running BASIC on one L-S terminal while Dick Wilcox was running the editor and the macro assembler on a second terminal. At the same time, a DECsystem-10 style SYSTAT system status display was running on a Processor Technology VDM, showing which jobs were alive, which one was active, etc., and also displaying a hours-minutes-seconds clock. It is, truly, a multi-tasking, multi-user system.

All of this was running on an IMSAI with 48K bytes of RAM and dual iCom floppy disc drives. John told me the 8080 board was still in there, doing little more than supporting the front panel. Incidentally, at one point the system crashed—I believe Dick was running in privileged mode and garbled his memory partitioning commands (i.e., operator error, not system error). He simply reached over, hit the front panel RESET switch, and booted off the disc in a few seconds. He asked John how much memory he wanted (for running BASIC), allocated it, and both of them were off and running again.

Evidently the system truly *is* device-independent. I talked to Dick about two days ago, asking if he could easily get the system running on the $350 IBM 2311 multi-platter, hard-disc, 7.25 MEGAbyte disc-pack drive that is being interfaced to the S-100 bus by another group in Southern California (ahhh, but that's another story for another issue). He said that it shouldn't take more than a coupla hours to write the driver for it; then it would run with no further changes to the software.

Concerning the BASIC "compiler": as I understand it, Dick's BASIC processor translates into an efficient intermediate code which is both compact and fast to interpret. In this way, it gains the reported 5-7 speed advantage over the MITS BASIC. It is not, however, a true compiler in the sense of generating absolute object code for the target machine.

I would like to particularly applaud Dick's position regarding low-cost distribution of software for not-for-profit use. He is recognizing and adjusting to the realities of the new world of personal computing in a manner that I feel is fair and reasonable for everyone concerned. His willingness to release his very extensive systems software *sources* is to be particularly applauded.

Why bother with a multi-tasking operating system on a "personal" computer? Let's daydream for a moment. Wouldn't it be nice to be able to start a lengthy listing on our hardcopy device; while that was running, start an assembly of a large program; and then go about editing the source for another program from our softcopy terminal? That's *exactly* what you can do with a multi-tasking system . . . the Sixteen/8 (a later name for the CM-16).

Why bother with multi-user capabilities? Well, let's say that your kids want to run Space War at the same time you want to do some "serious work" on your home system. Or, let's say a school has already purchased a number of IMSAI's—such as is the case with the San Jose Unified School District in California—and would like to allow more students to use the machines at the same time, than is possible in a one-on-one environment. Since the capability is already there—it was in Dick's home-grown operating system on the 11/20 for four years—it might be useful. On the other hand, you can ignore it if you wish.

Watch for a comprehensive evaluation of the Sixteen/8 in a future issue of *DDJ*. Dick and John have promised that, as soon as they get their first batch of production units, they will let me come down, pick any one I choose out of the bunch, bring it back to the Bay Area, and bang away on it to my heart's content. Be assured that an evaluation of that unit—particularly its software and systems aspects—will appear in *Dr. Dobb's* shortly thereafter.

```
Multiple-bit Shift Instructions:
     SSRR  reg,cnt      Shifts the word in reg right cnt times through the carry
     SSLR  reg,cnt      Shifts the word in reg left cnt times through the carry
     SSRA  reg,cnt      Shifts the word in reg right cnt times from the carry
     SSLA  reg,cnt      Shifts the word in reg left cnt times into the carry
     SDRR  reg,cnt      Shifts the double-word in reg,reg+1 same as SSRR
     SDLR  reg,cnt      Shifts the double-word in reg,reg+1 same as SSLR
     SDRA  reg,cnt      Shifts the double-word in reg,reg+1 same as SSRA
     SDLA  reg,cnt      Shifts the double-word in reg,reg+1 same as SSLA
```

Floating Point Instructions: at this time the floating point microprogramming is not complete and the final format is still undergoing change. The basic floating point instructions will be available as an optional fifth microprocessor chip.

Please note that many of the instructions are rather exotic and were designed specifically to be used in the operating system for subroutine processing. Many of the instructions in this category will not be used for application programs but are available for those who wish to dabble with them.

Lawrence Livermore Lab's 8080 BASIC

includes floating point package

user documentation & complete, annotated source

Foreward

The BASIC interpreter described in this user's guide was developed at the University of Idaho by John Dickenson, Jerry Barber, and John Teeter under a contract with the Lawrence Livermore Laboratory. In addition, Jerry Barber, as an LLL summer employee, made significant contributions to this document and to implementing the BASIC language in an MCS-8080 8080 microprocessor.

Abstract

Scientists are finding increased applications for microprocessors as process controllers in their experiments. However, while microprocessors are small and inexpensive, they are difficult to program in machine or assembly language. A high-level language is needed to enable scientists to develop their own microprocessor programs for their experiments on location. Recognizing this need, LLL contracted to have such a language developed. This report describes the result—the LLL BASIC interpreter.

Introduction

The BASIC interpreter described in this user's manual was designed to operate with the MCS-8080 microprocessor. It consists of a 5K-byte-PROM resident interpreter used for program generation and debug.

The goal in developing the 8080 BASIC was to provide a high-level, easy-to-use language for performing both control and computation fuctions in the MCS-8080 microprocessor. To minimize system size and cost, the interpreter was constrained to fit into 5K bytes. It was necessary, therefore, to limit the commands to those considered the most useful in microprocessor applications.

A list of these commands is given in Table 1, and a list of the statements making up the BASIC interpreter is presented in Table 2. Average assembly-language execution times and the various operations allowed in the BASIC floating-point package are given in Table 3.

Table 1. BASIC interpreter commands.

Command	Action
RUN	Begins program execution
SCR	Clears program from memory
LIST	Lists ASCII program in memory
PLST	Punches paper-tape copy of program
PTAPE	Reads paper-tape copy of program using high-speed reader
CNTRL S	Interrupts program during execution

Table 2. BASIC statements.

Statement	Function
0 to 32767	Indicates BASIC line number (maximum range 0 to 32767).
REM	Indicates a comment (Spaces are ignored excpet when enclosed in quotes, therefore, comments are generally enclosed in quotes)
END	Indicates end of program
STOP	Stops program
GO to XX	Transfers to line number XX
DIM	Declares an array (Only one-dimensional arrays with an integer number of elements are allowed)
LET	Indicates an assignment statement (Additio, subtraction, multiplication, division, or special function may be used) tion, subtraction, multiplication, division,
IF expression THEN XX	Condition statement which transfers to line number XX if the condition of the expression is met
INPUT	Allows numeral data to be inputed via a terminal
PRINT	Allows numerical data and character strings to be printed on a terminal
FOR	Causes program to iterate through a loop a designated number of times
NEXT	Signals end of loop and at which point the computer adds the step value to the variable and checks to see if the variable is still less than the terminal value
GO SUB NN	Transfers control to a subroutine that begins at line NN
RETURN	Returns control to the line after the last GO SUB
CALL	CALL (N,A,B, ...) N = subroutine No. as listed in assembly patch table A, B, etc. = parameters, constants, variables, or expressions.
GET	(X) = READ 8080 INPUT PORT X
PUT	(Y) = OUTPUT A BYTE OF DATA TO OUTPUT PORT Y

Table 3. BASIC operations and execution times.

Operation	Execution times on 8080 (msec)
ADD	2.4
SUBTRACT	2.4
MULTIPLY	5.4
DIVIDE	7.0

USING THE BASIC INTERPRETER

Starting the Interpreter

The BASIC interpreter is presently configured so that it is located in memory pages 11_8 to 34_8. The starting address is 17_8, location 0. This address begins an initialization sequence a second starting address–page 17_8, location 23_8–can be used. This starting address is used if the user wishes to retain any program that might exist in memory.

Once started the interpreter responds with READY.

Entering a Line

Each line entered is terminated with the carriage-return key. The line-feed key is ignored. It is possible to correct errors on a line being entered by either deleting the entire line or by deleting one or more characters on the line. A character is deleted with either the rubout key or the shift/ 0 kry. Several characters can be deleted by using the rubout key several times in succession. Character deletion is, in effect, a logical backspace. To delete the line you are currently typing, use the CNTRL/Y key.

The above line-editing features can be used on command, program, or data lines.

Commands

The following commands are available:

RUN - Begins program execution
SCR - Clears program from memory
LIST - Lists program in memory
PLST - Punches paper-tape copy of program
PTAPE - Reads in paper-tape copy of program using high-speed reader

The LIST and PLST commands can be followed by one or two line numbers to indicate that only a part of the program is to be listed. If one line number follows the command, the program is listed from that line number to the end of the program. If two line numbers (separated by a comma) follow the command, the listing begins at the first line number and ends at the second.

When a command is completed, READY will be typed on the teletype. Once initialized by a command, a process will normally go to completion. However, if you wish to interpret an executing program or a listing, simply strike CNTRL S and the process will terminate and a READY message will be typed. typed.

Statements

Each statement line begins with a line number, which must be an integer between 0 and 32767. Statements can be entered in any order, but they will be executed in numerical order. All blanks are ignored.

A program can be edited by using the line numbers to insert or delete statements. Typing a line number and then typing a carriage return causes statement at that line number to be deleted. Since the statements can be entered in any order, a statement can be inserted between between two existing statements by giving it a line number between the two existing statement line numbers. To replace a statement, the new statement should have the same line number as the old statement. The following types of statements are allowed:

REM - Indicates a remark (comment). The system deletes bln
deletes blanks from all character strings that are not enclosed in quotes ("). Therefore, it is suggested that characters following the REM key word be enclosed in quotes.

END - Indicates the end of a program. The program stops when it gets to the END statement. All programs must end with END.

STOP - Stops the program. This statement is used when the program needs to be stopped other than at the end of the program text.

GOTO - Transfers to a line number. This statement is used to loop or jump within a program.

DIM - Declares an array. Only one-dimensional arrays with an integer constant number of elements are allowed. An array with N elements uses indexes 0 through N-1. All array locations are set to zero. No check is made on subscripts to ensure that they are within the declared array. An array variable must be a single letter.

LET - Indicates an assignment statement. Non-array variables can be either a single letter or a letter followed by a digit. It is possible to have an array and a non-array variable with the same name. The general form of the LET statement is:

line number LET identifier = expression,

where "identifier" is either a subscripted array element or a non-array variable or function (see section on functions) and "expression" is a unary or binary expression. The expression will be one of the following ten types:

variable	-variable - variable
-variable	variable * variable
variable + variable	-variable * variable
variable - variable	variable / variable,
-variable + variable	-variable / variable,

where "variable" is an identifier, function, or number. The subscript of an array can also be an expression.

Numbers in a program statement or input via the teletype are handled with a floating-point package provided by LLL. Numbers can have any of the following forms:

4	±4	.123
4.	±4.0	±.123
4.0	1.23	0.123
±4	±1.23	±0.123

and the user may add an exponent to any of the above forms using the letter E to indicate powers of 10. The forms of the exponent are:

E±1	E±15
E 1	E 15
E 1	E 15.

The numbers are stored with seven-digit accuracy; therefore, seven significant figures can be entered. The smallest and largest numbers are ±2.71051E-20 and ±9.22337E18.

IF - This is the conditional statement. It has the form: line number IF expression relation expression THEN transfer line number. The possible relations are:

Equal	=
Greater than	>
Less than	<
Greater than or equal	>= =<
Less than or equal	<= =<
Not equal	<> ><

If the relation between the two expressions is true then the program transfers to the line number, otherwise it continues sequentially.

INPUT - This command allows numerical data to be input via the teletype. The general form is:

Line number INPUT identifier list,

where an "identifier list" is a sequence of identifiers separated by commas. There is no comma after the last identifier so, if only one identifier is present, no comma is needed. When an INPUT statement is executed, a colon (:) is output to

the teletype to indicate that data are expected. The data are entered as numbers separated by commas. If fewer data are entered than expected, another colon is output to the teletype, indicating again that data are expected. For example, where

```
50 INPUT I,J,K,P
```

is executed, a colon is output to the teletype. Then, if only 3 numerical values are entered, another colon will be output to indicate that more data are expected; e.g.,

```
: 4,4,6.2 C/R
: 10.3 C/R
```

where C/R is the carriage-return key. If an error is made in the input-data line, an error message is issued and the entire line of data must be re-entered. If, for the above example,

```
: 4,4,6M2,10.3 C/R
```

is entered, the system will respond

```
INPUT ERROR, TRY AGAIN
:
```

At this time, the proper response would be

```
4,4,6.2,10.3 C/R.
```

PRINT - This command allows numerical data and character strings to be printed on the teletype. Two types of print items are legal in the print statement: character strings enclosed in quotes (") and expressions. These items are separated by either a comma or a semicolon. If print items are separated by a comma, a skip occurs to the next pre-formatted field before printing of the item following the comma begins. The pre-formatted fields begin at columns 1, 14, 27, 40, and 52. If print items are separated by a semicolon, no skip occurs. If a semicolon or comma is the last character on a print statement line, the appropriate formatting occurs and the carriage-return-line feed is suppressed. A print statement of the form

```
50 PRINT
```

will generate a carriage-return-line feed. Thus, the two lines below

```
50 PRINT "INPUT A NUMBER";
60 INPUT A
```

will result in the following output:

```
INPUT A NUMBER:
```

For more examples, see sample programs in Appendix A.

FOR - Causes program to iterate through a loop a designated number of times.

NEXT - Signals end of loop at which point the computer adds the step value to the variable and checks to see if the variable is still less than the terminal value.

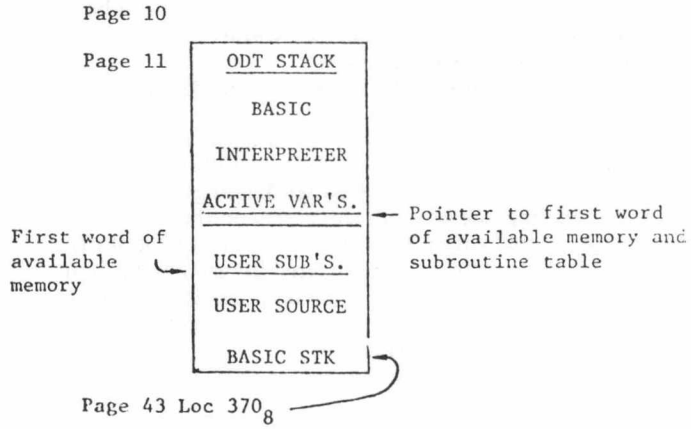
GOSUB NN - Transfer control to a subroutine that begins at line NN.

RETURN - Returns control to the next sequential line after the last GOSUB statement executed. A return statement executed before a GOSUB is equivalent to a STOP statement.

CALL - Calls user-written assembly-language routines of the form

```
CALL (N, A, B, . . .),
```

where N is a subroutine number from 0 - 254 and A, B, . . . are parameters. The parameters can be constants, variables, or expressions. However, if variables and constants or expressions are intermixed, all variables should have been referenced before the CALL statement. Otherwise, the space reserved for newly referenced variables may overwrite the results of constants and expressions. A memory map of one configuration of the system is shown below:



The subroutine table contains 3-byte entries for each subroutine. The table directly follows the pointer to the first word of available memory (FWAM) and must end with an octal 377. A sample table and its subroutines is shown below:

```
ORG 1/65112Q
DW SUBEND          ; Define FWAM
DB 1               ; Subroutine #1
DW SUB1            ; Starting add of
                     subroutine #1
DB 4               ; Subroutine #4
DW SUB4            ; Starting add of
                     subroutine #4
DB 5               ; Subroutine #5
DW SUB5            ; Starting add of
                     subroutine #5
DB 2               ; Subroutine #2
DB SUB2            ; etc.
DB 377Q            ; end of subroutine table
SUB1: ↓            ; Subroutine #1
      RET
SUB5: ↓            ; Subroutine #5
      RET
      •
      •
      •
      RET
SUBEND EQU $       ; FWAM
```

Addresses to passed parameters are stored on the stack. The user must know how many parameters were passed to the subroutine. These must be taken off the stack before RET is executed. Addresses are stored last parameter first on the stack. Thus, on entry to a subroutine, the first POP instruction will recover the address to the last parameter in the call list. The next will recover the next to last, etc.

Each scalar variable passed results in the address to the first byte of a four-byte block of memory. Each array element passes the address to the first byte of a (N-M) x four-byte memory block, where N is the number of elements given the array in the DIM STMT and M is the array subscript in the CALL STMT.

For passed parameters to be handled in expressions within BASIC, they must be in the proper floating-point format.

Functions

Two special functions not found in most BASIC codes are available to input or output data through Intel 8080 port numbers.

The function GET allows input from a port and the function PUT allows output to a port. Their general forms are:

GET (expression)
PUT (expression).

The function GET may appear in statements in a position that implies that a numerical value is used. The function PUT may appear in statements in a position that implies that a numerical value will be stored or saved. This is because GET inputs a number and PUT outputs a number. For example,

LET PUT(I) = GET(J) is valid

while

LET GET(I) = PUT(J) is invalid.

These functions send or receive one byte of data, which in BASIC is treated as a number from 0 to 255.

Error Messages

If an unrecognizable command is entered, the word WHAT? is printed on the teletype. Simply retype the command. It may also have been caused by a missing line number on a BASIC statement, in which case you should retype the statement with a line number.

If an error is encountered while executing a program, an error message is typed out that indicates an error number and the line number in which the error occurred. The meanings of the error numbers are given in Table 4.

Table 4. Meanings of error numbers.

Error number	Error
1	**Program has no END statement**
2	**Unrecognizable keyword at beginning of statement**
3	**Source statements exist after END statement**
4	**Designation line number is improperly formed in a GOTO, GOSUB, or IF statement**
5	**Destination line number in a GOTO, GOSUB, or IF statement does not exist**
6	**Unexpected character**
7	**Unfinished statement**
8	**Illegally formed expression**
9	**Error in floating-point conversion**
10	**Illegal use of a function**
11	**Duplicate array definition**
12	**An array is referenced before it is defined**
13	**Error in the floating-point-to-integer routine, number is too big**
14	**Invalid relation in an IF statement**

During program execution and whenever new lines are added to the program, a test is made to see if there is sufficient memory. If the memory is full, MEMORY FULL is printed on the teletype. At this point, you should enter one of the single digits below to indicate what you wish to do:

Number entered	Meaning
0	(RUN) runs the program in memory
1	(PLST) outputs program in memory to paper tape punch
2	(LIST) lists program in memory
3	(SCR) erases program in memory
4	none of the above (will cause WHAT? to be printed in teletype).

To help you select the best alternative, a brief description of how the statements are manipulated in memory will be helpful. All lines entered as program are stored in memory. If lines are deleted or replaced, the original still remains in memory. Thus, it is possible, if a great deal of line editing has been done, to have a significant portion of memory taken up with unused statements. If a MEMORY FULL message is obtained in these circumstances, then the best thing to do is punch a tape of the program (entering number 1), then erase the program memory with a SCR command (or a number 3, if memory is too full to accept commands), and then re-enter your program using the high-speed paper-tape reader with the PTAPE command.

Appendix A: Sample Programs

The program below gives a few examples of the use of the print statement.

```
LIST
1PRINT"THE PRE-FORMATTED COLUMNS ARE SHOWN
      BELOW
2PRINT1,2,3,4,5
4PRINT
1Ø PRINT"INPUT 1ST NUMBER":
2Ø INPUTA
3ØPRINT"INPUT 2ND NUMBER",
4ØINPUTB
5ØPRINT
6ØPRINT"A IS";A
7ØPRINT"B IS",B
8ØPRINT"A IS";A;"B IS",B,"A+B IS";A+B
1ØØEND
READY
RUN
THE PRE-FORMATTED COLUMNS ARE SHOWN BELOW
1.ØØØØE ØØ  2.ØØØØE ØØ  3.ØØØØE ØØ  4.ØØØØE ØØ
                                          5.ØØØØE ØØ
INPUT 1ST NUMBER:2
INPUT 2ND NUMBER          :3

A is 2.ØØØØE ØØ
B IS        3.ØØØØE ØØ
A is 2.ØØØØE ØØB IS        3.ØØØØE ØØ  A+B IS 5.ØØØØE ØØ
READY
```

The following program plots a function on a display. It uses four user-written assembly-language subroutines. The display works as follows: The contents of memory locations on pages 274_8 to 277_8 are displayed as 16 rows of 64 characters each. Thus, if location 201_8 on page 274 contains 301_8 (ASCII A), an A appears in column 2 of Row 3. An example of this program's execution is shown below:

```
RUN
WHAT SHOULD PLOT BE LABELED? MCS80
                BASIC INTERPRETER
READY
```

The BASIC and assembly-language programs and the display output are shown on the following pages.

BASIC Program

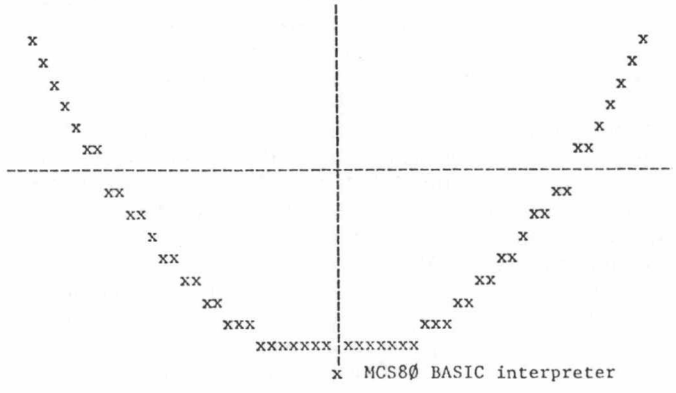
```
LIST
1REM"   THIS ROUTINE WILL PLOT A SET OF AXIS AND A QUADRATIC FUNCTION
2REM"   ON A DISPLAY AND THEN LABEL IT.  IT USES A 4 USER WRITTEN
3REM"   SUB-ROUTINES:
4REM
5REM"   CALL (1,X,Y,C) - PLACES C IN COLUMN X, ROW Y OF THE DISPLAY
6REM"     WHERE C IS AN ASCII CODED CHARACTER
7REM
8REM"   CALL(2,A(Ø)) - READS A CHARACTER STRING FROM THE TTY AND STORES
9REM"     IT"IN ARRAY A
10REM
11REM"  CALL(3,A(Ø)) - WRITES THE CHARACTER STRING STORED IN ARRAY A
12REM"    TO THE DISPLAY
13REM
14REM"  CALL(4) - CLEARS THE DISPLAY
15REM
16REM"    START OF PROGRAM
17REM
18REM"    RESERVE STORAGE AREA FOR TITLE
2ØDIMA(1Ø)
3ØREM"    CLEAR SCREEN
4ØCALL(4)
5ØREM"    ASK FOR AND INPUT TITLE
55PRINT"WHAT SHOULD PLOT BE LABELED?";
6ØCALL(2,A(Ø))
7ØREM"    DRAW AXIS
8ØGOSUB5ØØ
9ØREM"    PLOT FUNCTION
1ØØLETX=-29
11ØGOSUB1ØØØ
12ØCALL(1,31+X,8-Y,248)
13ØLETX=X+1
14ØIFX><31 THEN11Ø
15ØREM"   OUTPUT TITLE
16ØCALL(3,A(Ø))
165REM"   WE'RE DONE
17ØSTOP
5ØØREM"   THIS SUB. WILL DRAW A SET OF AXIS
5Ø5LETX=1
51ØLETY=7
52ØLETC=173
53ØCALL(1,X,Y,C)
54ØLETX=X+1
55ØIFX><65THEN53Ø
56ØLETX=31
57ØLETY=1
575LETC=252
58ØCALL(1,X,Y,C)
59ØLETY=Y+1
6ØØIFY><17THEN58Ø
61ØRETURN
1ØØØREM"  GIVEN X THIS SUB. CALCULATES (17/9ØØ)*X**2-8
1ØØ5REM"  FIRST CHECK IF X=Ø AS IT WILL UPSET FLT. PNT. PACK.
1Ø1ØIFX=ØTHEN1Ø45
1Ø15REM"  WE'RE OK - CALCULATE FUNCTION
1Ø2ØLETY=X*X
1Ø25LETK=17/9ØØ
1Ø3ØLETY=Y*K
1Ø35LETY=Y-8
1Ø4ØRETURN
1Ø45LETY=-8
1Ø5ØRETURN
2ØØØEND
READY
```

Assembly-language program

```
 !!!!!!!
1
8080 MACRO ASSEMBLER  VER 2 2  ERRORS = 0 PAGE 1

                        ;DEFINE EXTERNALS
  014012                FIX       EQU     14012Q          ;FIX ROUTINE
  013212                COPDH     EQU     13212Q          ;COPY ROUTINE
  016567                FREG1     EQU     16567Q          ;FLOATING PNT  REGISTER
  016614                          ORG     16614Q
  016614    027 036               DW      SBEND           ;FWAM
                        ;ENTRIES IN SUB TABLE
  016616    001                   DB      1
  016617    233 035               DW      SCOPE
  016621    002                   DB      2
  016622    334 035               DW      SUB2
  016624    003                   DB      3
  016625    364 035               DW      SUB3
  016627    004                   DB      4
  016630    003 036               DW      SUB4
  016632    377                   DB      377Q            ;NO MORE ENTRYS
                        ;THE CALL TO THIS ROUTINE IS OF THE FORM
                        ;       CALL(1 X Y C)
                        ;THE VALUE OF C IS PLACED IN COLUMN X  LINE Y
                        ;OF THE DISPLAY
  016633    321         SCOPE:    POP     D               ;ADDRESS OF CHARACTER
  016634    041 167 035           LXI     H FREG1         ;COPY TO FREG1
  016637    315 212 026           CALL    COPDH
  016642    353                   XCHG                    ;ADDRESS TO DE
  016643    315 012 030           CALL    FIX             ;FIX IT
  016646    023                   INX     D               ;PNT TO 4TH BYTE
  016647    023                   INX     D
  016650    023                   INX     D
  016651    032                   LDAX    D               ;GET CHARACTER
  016652    107                   MOV     B A             ;SAVE IN B
  016653    321                   POP     D               ;ROW ADD
  016654    041 167 035           LXI     H FREG1         ;COPY TO FREG1
  016657    315 212 026           CALL    COPDH
  016662    353                   XCHG
  016663    315 012 030           CALL    FIX             ;FIX IT
  016666    023                   INX     D               ;GET BYTE 4 TO A
  016667    023                   INX     D
  016670    023                   INX     D
  016671    032                   LDAX    D
  016672    117                   MOV     C A             ;SAVE IN C
  016673    321                   POP     D               ;GET COLUMN ADD
  016674    041 167 035           LXI     H FREG1         ;COPY TO FREG1
  016677    315 212 026           CALL    COPDH
  016702    353                   XCHG
  016703    315 012 030           CALL    FIX             ;FIX IT
  016706    023                   INX     D               ;PNT TO 4TH BYTE
  016707    023                   INX     D
  016710    023                   INX     D
  016711    032                   LDAX    D               ;GET IT TO A
  016712    041 377 273           LXI     H 135777Q       ;CALCULATION OF ADDRESS
  016715    021 100 000           LXI     D 100Q
  016720    015         LUP:      DCR     C
  016721    312 330 035           JZ      ADINC
  016724    031                   DAD     D
  016725    303 320 035           JMP     LUP
  016730    137         ADINC:    MOV     E A
  016731    031                   DAD     D               ;ADD IN COLUMN LOC
  016732    160                   MOV     M B             ;STORE CHARACTER
  016733    311                   RET                     ;DONE
                        ;SUB2 READS A TITLE FROM TTY VIA ODT
  000333                READ      EQU     333Q            ;ODT ROUTINE
  016734    341         SUB2:     POP     H               ;GET STORAGE AREA ADD
  016735    345                   PUSH    H
  016736    016 000               MVI     C Ø             ;INIT CNTR
  016740    043         LUP2:     INX     H               ;BUMP PNTR
  016741    315 333 000           CALL    READ            ;READ A CHARACTER
  016744    376 215               CPI     215Q            ;CR?
  016746    312 356 035           JZ      DUN2            ;YES - DONE
  016751    014                   INR     C               ;INCR  CNT
  016752    167                   MOV     M A             ;SAVE CHARACTER
  016753    303 340 035           JMP     LUP2
  016756    341         DUN2:     POP     H               ;STORE CNT
  016757    161                   MOV     M C
  016760    076 212               MVI     A 212Q          ;SEND A LF
  016762    367                   RST     6
  016763    311                   RET                     ;DONE
                        ;SUB3 WRITES TITLE TO DISPLAY
  016764    341         SUB3:     POP     H               ;GET ADD
  016765    021 341 277           LXI     D 137741Q       ;SCREEN ADD
  016770    116                   MOV     C M             ;CNT
  016771    043                   INX     H
  016772    176         LUP3:     MOV     A M             ;SEND STRING
  016773    022                   STAX    D
  016774    043                   INX     H
  016775    023                   INX     D
  016776    015                   DCR     C
  016777    302 372 035           JNZ     LUP3
  017002    311                   RET                     ;DONE
                        ;SUB4 CLEARS SCREEN
  017003    041 000 274 SUB4:     LXI     H 136000Q       ;SCREEN ADD
  017006    076 240               MVI     A 240Q          ;SPACE
  017010    026 000               MVI     D Ø             ;CNTR S
  017012    016 004               MVI     C 4
  017014    167         LUP4:     MOV     M A             ;CLEAR IT
  017015    043                   INX     H
  017016    025                   DCR     D
  017017    302 014 036           JNZ     LUP4
  017022    015                   DCR     C
  017023    302 014 036           JNZ     LUP4
  017026    311                   RET                     ;DONE
  017027                SBEND     EQU     $
                                  END
NO PROGRAM ERRORS
```

SYMBOL TABLE

```
* 01

A       000007      ADINC   016730      B       000000      C       000001
COPDH   013212      D       000002      DUN2    016756      E       000003
FIX     014012      FREG1   016567      H       000004      L       000005
LUP     016720      LUP2    016740      LUP3    016772      LUP4    017014
M       000006      PSW     000006      READ    000333      SBEND   017027
SCOPE   016633      SP      000006      SUB2    016734      SUB3    016764
SUB4    017003
```



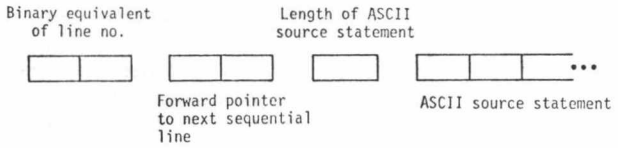
Display output for preceding program.

Appendix B: Description of Basic Interpreter

Following is a brief description of the BASIC interpreter. Hopefully, with this description, it will not be a major project to modify the BASIC to satisfy the reader's specific needs.

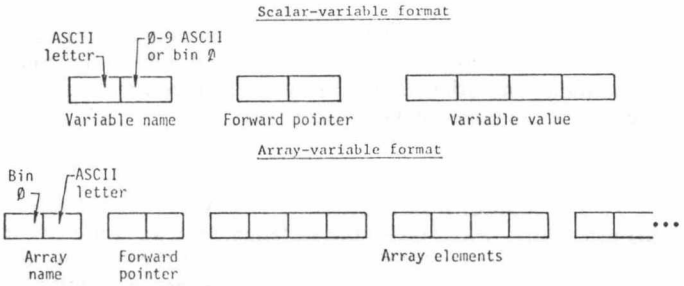
Formats

Source statements are stripped of blanks on input (character strings enclosed in " "s are an exception) and stored as is in memory, using the following format:



The forward pointer links statements by ascending line numbers. The last line's forward pointer (supposedly an end statement) has value 177777_8 to indicate end of the list.

The symbol table is built up at run time and begins after the most recently entered source statement (the variable STSPAC points to where the symbol table will start). Symbol table entries are shown below.



Subroutines

Following is a list of potentially useful subroutines, with a brief description on each subroutine:

ALPHA - Value pointed to by Hand L is tested to see if it is an ASCII letter. CY = 1 => Yes
CY = 0 => No

NUMB - Same as above but tests for a decimal number (ASCII 0-9).

CHAR2 - Inputs a character from the teletype to a register.

CHAR5 - Same as above for HSR.

CHK1 - Checks to see if HL are equal to 177777_8 (-1). CY = 1 => Yes.
CY = 0 => No.

CONV (CURT) - One of the floating-point routines. Converts floating-point number to a character string. Output is padded to the output buffer.

COPDH - Copies floating-point number pointed to by D,E to location pointed to by H,L; uses copy.

COPY - One of the floating-point routines. Copies floating-point value pointed to by A,L to location pointed to by H,C.

CUB - Converts the integer-character string pointed to by H,L to its binary equivalent. Value returns in D,E registers.

DCOMP - Double-byte comparison routine. Compares value in CB to the in ED.

Z = 1	=>	CB = ED
CY = 1	=>	CB > ED
CY = 0	=>	CB $\geq$ ED.

DFXL - One of the floating-point routines. Used to float an unsigned integer H,L point to first of four bytes; integer is right justified in first three bytes.

EVAL - Evaluates an expression the first element of which is pointed to by H,L and the length of which is in C. Used to evaluate expressions wherever they are legal in BASIC. C usually contains the length of the source statement line containing the expression.

FINPT - One of the floating-point routines. Converts character string to floating-point number. The variable HLINP contains a pointer to the character string, and the variable CREG contains the length of line containing character string. Mode = 0 => data comes from teletype (i.e., only delimiters are g's). Mode = 1 => data comes from source statements.

FIX - Fixes a floating-point number. DE points to number to be fixed. Error code 13 is given if number is too big to fix.

FSYM - Finds symbols in symbol table. BC contains symbol. Returns with HL pointing to symbol value. CY = 1 => symbol was found. CY = 0 and a scalar => symbol not found, but inserted and initialized to 0. CY = 0 and an array => not found, no action taken: HL are meaningless.

LADD - Floating-point add routine.

LSUB - Floating-point subtract routine.

LOIU - Floating-point divide routine.

LMUL - Floating-point multiply routine.

LMCM - One of the floating-point routines. Compares two floating-point values HL Point to first
HB point to second
z = 1 => Equality
CY = 1 => first < second
(Note: compares absolute only, does not reference mantissa sign.)

MCHK - Waits for flag from port 3. Proper mask is sent in register B.

MEMFUL - Checks to see if memory is full. HL point to location of memory to be checked. Memory is considered full if it is within 50_{10} locations of the current value of stack pointer.

MULT - Multiplies two two-byte binary numbers. HL point to last byte of four bytes. First two contain first number. Last two contain second number. Answer returns in BCDE.

NSRCH - Routine to locate source line in memory passed binary value of line number in DE. Returns address of line in HL, CY = 1 => not found.

OUTR - Used by CONV (CURT) to pad output to output buffer.

PAD - Pads characters to output buffer. A contains character; B contains number of pads.

SYMSRT - Checks a character string to see if it is a BASIC symbol. HL contains address pointing to first character of symbol, C contains length of line that contains symbol. A contains type of symbol sought.
0 = command 1 = keyword
z = operator or delimiter 3 = function
Returns with 377_8 in a register if nothing found. Otherwise A contains

symbol number in appropriate KDAT table. Thus, for symbol type 2, if a 4 is returned, the symbol found was the fourth one (starting with 0) in table KDAT3 (KDAT concatenated with 2 and 1 or A). CIS is updated, but HL is not.

TTYIW - Inputs a line from teletype. Stores starting at location pointed to by HL. Line edits. Returns length of line in A register (maximum line length is 72 characters).

VALUE - Called with HL pointing to a variable, constant, or function; C contains line length, returns with DE pointing to floating-point value. HL, C are updated.

VAR - Called with HL pointing to character string, C has line length. Determines if character string is a variable. If so, returns with CY = 1, DE pointing to value (subscripts of arrays are evaluated, etc.). HL, C updated. If not, a variable returns CY = 0, HL, C untouched.

WRIT - Dumps contents of output buffer to teletype. Uses entry WRIT1 with D register equal to one to suppress CR/LF.

ZROL - Part of floating-point subroutines. Writes a floating-point zero, starting at location pointed to by HL.

The preceding list contains those subroutines most likely to be used by someone modifying BASIC. If you plan on using one of the routines, you should examine it and its comments carefully.

Variables

Following is a list of interpreter variables, with a description of each variable.

MEMST - Assembly time variable. Contains the first available RAM location. This is where active variables start.

MEMEND - Assembly time variable. Contains the last available location in RAM

SEND - Has value 6, used with RST instruction to print characters via ODT.

OBUFF - Output buffer, the first location contains the number of characters in the buffer +1.

IBUF - Input buffer, occupies same area as OBUFF.

STLINE - Points to first source line to be executed. If no source, contains 177777_8.

NLINE, NLZ, NL4, NL6 - Contain address, binary-equivalent line number, forward pointer, and length of next input line.

KLINE, KL2, KL4, KL6 - Same as above, but used by a subroutine that inserts lines in sequential order (insert).

PLINE, PL2, PL4, PL6 - Subroutine insert to order statements sequentially.

KASE, LEN - Temporary storage for commandmode routines.

MULT1, MULT2 - Used to store binary values to be multiplied.

SBSAV - Temporary storage for commandmode routines.

STSPAC - Next available location in memory, symbol table starts here at run time.

LPNT - Pointer to the current line at run time.

CPNT - Pointer to current character in current line at run time.

KFPNT - Point to next sequential line at run time.

FREG1, FREG2 - Two floating-point registers.

HLINP, CREG - Temporary storage for HL and C registers for routine INP.

NXTSP - Pointer to next available space of memory for symbol table.

GREG - General register, in and out instructions are stored here and executed for get and put functions.

MODE - Indicates to INP routine whether input data comes from source or teletype.

MESCR - Temporary storage for call-statement processor. Points to next available space after symbol table. Area after the symbol table is used to store intermediate results of expressions or constants passed to user subroutines.

VARAD - Temporary storage space for input-statement processor.

VEND - Assembly time variable. Indicates end of interpreter variable-storage area and where FWAM pointer is to go.

FWAM - First word of available memory pointer. This is where user source programs go.

Some of the above variables occupy the same area of memory. This is because some variables are used only in the command mode and others only at runtime. To conserve space, they share the same memory locations.

New Statements

To add additional statements to the BASIC, use the following procedure. First, insert the statement keyword in the data tables for subroutine SYMSRT. Then, insert the starting address of the statement processor in the interpreter JUMP table. Finally, the statement processor itself must be inserted.

The keyword must be entered in the table KDAT2. The first byte must be the keyword length and the next bytes hold the ASCII-coded keyword. The table must end with a 377_8. If the keyword is the Nth entry in the table, on return from SYMSRT, the A register will hold N-1 if the keyword is found.

The starting address of the statement processor must be inserted into table JTBL. The order of keywords in KDAT2 must correspond with statement processor addresses in JTBL since, on return from SYMSRT, the A register times two is used as offset in JTBL to determine processor address.

The statement processor must be placed somewhere in memory. Generally, the first thing done in the statement processors is to load the pointer to the statement (LHLD CPNT) and increment past the keyword (since HL is not updated by SUMSRT). On entry, C contains the number of characters in the line minus those checked by SYMSRT. The end of the processor should be a "JMPIEND" instruction.

New Functions

New functions must be added to SYMSRT Data Table KDAT4 in the same manner as for key words. The function itself must be placed in subroutine "VALUE." Presently, the only function in VALUE is GET.

Message Lines

The following description tells how to incorporate messages into B ASIC output routines. Currently, to output a message to the teletype, the user executes an LXI H,ODATA, then a call to FORMK where K is an integer indicating which message is wanted (i.e., K = z indicates "TURN ON PUNCH"). FORM pads the message into the output buffer. Then A "CALL ?

WRIT" writes the contents of the fubber.

Suppose the message "POTATO BASIC" is to be added. Preceding the form 9 instruction, we will insert "FOR10: INR L." At the end of the ODATA table, we add "DB ODAT8 and 377Q.". And, after message ODAT 7, we add ODAT8 DB *12, "POTATO BASIC." Now, the following program segment:

```
LXI H,ODATA
CALL FOR10
CALL WRIT,
```

will cause "POTATO BASIC" to be output to the teletype.

Complete Assembled, Annotated Source Code Begins on Next Page

*12 is the character count in the message.

FIRST JAPANESE COMPUTER CLUB & A WORLD-WIDE CONTEST

Japan has been slow to enter the microcomputer hobby; however, it looks as though the dam is breaking: there is now now a Japan Microcomputer Club (address: The 1st Ohkura Building 4F, 2-1 Nihonbashi, Chou-ku, Tokyo, Japan). At the Mini/Micro Exposition in San Francisco we picked up the first issue of their newsletter (translated from Japanese) dated October, 1976. They already have a significant organizational and administrative structure that includes club officers associated with the University of Tokyo, Intel Japan, Waseda University, Keio University, Seikei University, SORD Company, Tokyo Institute of Technology, the University of Gifu, Hosei University, Nippon Tel & Tel, and JIPDEC . . . as well as other organizations, less well known in the U.S. They are having almost weekly symposia this fall, including speakers from NEC, TDK Fairchild, Toshiba, Fjuitsu, Hitachi, Intel Japan, Mitsubishi, and other Japanese microprocessor manufacturers.

The back page of their "Micro Computer News" mentions a microcomputers application contest: "For developing techniques, directing a spotlight on to unknown talents, and encouraging new ideas, we are planning a contest in March, 1977. Anyone in the world can be an entrant. Subscription should be sent not later than February, 1977. No only winners but also all the entrants will have marvelous prizes!"

Another interesting tidbit is that one of the Club's General Council Members, Hidetoshi Takahashi, was part of the JIPDEC Microprocessor Study Team which visited People's Computer Company last spring [see *DDJ*, May 1976].

BOOK ON DIGITAL TROUBLESHOOTING

Many computer hobbyists are either neophytes or "software people who may have mad little or no experience in the care and feeding of LSI hardware. When their JOLT, KIM, or POLY breaks a leg they either ship it back to the factory, tote it to the nearest computer store, or shoot it. There is, however, a book that should be quite helpful in providing a basic understanding of digital troubleshooting techniques. This is Richard E. Gasperini's *Digital Troubleshooting*, available for $9.95 from the Hayden Book Co., 50 Essex St., Rochelle Park, NJ 07662.

This book introduces the reader to the fundamentals of bipolar and MOS technology, explains how to read logic diagrams, and discusses basic tools and diagnostic methods. It then covers the use of common digital circuits such as decoders, multiplexers, flip-flops, buffers, counters, and shift registers. The book does assume some prior knowledge of electronics, but not of digital circuitry. It discusses the use of memory and display devices, and gives many useful tips on soldering and unsoldering ICs, obtaining replacement parts, etc. Chapter 18 contains some good suggestions on obtaining additional information, and an appendix explains how to interpret manufacturer's part numbers, such as SN74LS195N.

Jim Day

FLOPPY DISC SYSTEM(S?) FOR NON-HARDWARE TYPES

Dear Jim, Dec. 22, 1976

A few weeks ago I called you seeking advice for a client. They were interested in acquiring a microcomputer with dual floppies that had a good BASIC and text editing capabilities. Sounds easy so far. The only hook was that the system was to be used by non-professionals in a non-technical environment with no hardware types around and only BASIC programs available.

After talking to you, I sought advice from a few others, as well. I was surprised at the result . . . if my clients want to buy *today*, there appears to be only one professional quality (non-hobby) system that fits their needs. That is the DTC Microfile. All other possibilities are in various stages of "development," "testing," "debugging" or are too new to have any track record. Microfile has been installed in the U.S. and abroad and appears to be a proven system. It's easy to use and beginners, with a little training, can easily use the text editor.

I assume that more users like my clients exist, and I assume that more vendors will attempt to match what DTC has already done or improve on it. How about you guys (vendors) keeping us informed as the systems come on the market.

Sincerely,
LeRoy Finkel

CACHE SURVEY

Results of a survey of the members of CACHE (Chicago Area Computer Hobby Enthusiasts) taken at the September meeting.

- How many members have an 8080 or Z80 computer "up & running" as of this date? **37 members.**
- How many members have a system "up & running" with any other CPU? **16 members.**
- Of those with 8080's, how much contiguous memory is currently available for applications program (RAM)?

Greater than 0 but less than 1K:	1	member
Greater than or equal to 1 but less than 2K:	2	
Greater than or equal to 2 but less than 4K:	1	
Greater than or equal to 4 but less than 8K:	7	
Greater than or equal to 8 but less than 12K:	11	
Greater than or equal to 12 but less than 16K:	6	
Greater than or equal to 16 but less than 24K:	4	
Greater than or equal to 24 but less than 32K:	5	
Greater than or equal to 32 but less than 64K:	0	

- Of those with 8080's, how many members do not have Page 0 available for applications programs? **1 member.**
- Of those with 8080's, how many have a terminal device "up & running" through parallel I/O ports (no software UARTS)? **16 members.**
- Of those above, how many know how to write I/O drivers for their terminal? **All 16 members knew how.**
- Of those above, how many can use an I/O driver of the form below?

```
GET:    IN      TIRP
        ANI     TIRM
        J(N)Z   GET
        IN      TIDP
        ANI     177Q
        RET

PUT:    PUSH    PSW
PUT1:   IN      TORP
        ANI     TORM
        J(N)Z   PUT1
        POP     PSW
        OUT     TODP
        RET
```

All 16 members could.

```
030000                          MEMST   EQU     30000Q          ;MUST BE ON PAGE BOUNDARY
037777                          MEMEND  EQU     37777Q
000006                          SEND    EQU     6               ;RST FOR ODT
030000                          OBUFF   EQU     MEMST           ;INPUT AND OUTPUT BUFFERS OCCUPY
030001                          IBUF    EQU     MEMST+1         ;SAME AREA
030111                          STLINE  EQU     MEMST+111Q
030113                          NLINE   EQU     MEMST+113Q
030115                          NL2     EQU     MEMST+115Q
030117                          NL4     EQU     MEMST+117Q
030121                          NL6     EQU     MEMST+121Q
030122                          KLINE   EQU     MEMST+122Q
030124                          KL2     EQU     MEMST+124Q
030126                          KL4     EQU     MEMST+126Q
030130                          KL6     EQU     MEMST+130Q
030131                          PLINE   EQU     MEMST+131Q
030133                          PL2     EQU     MEMST+133Q
030135                          PL4     EQU     MEMST+135Q
030135                          SBSAV   EQU     PL4             ;RETURN ADD. SAVE FOR CALL STMT.
030137                          PL6     EQU     MEMST+137Q
030140                          KASE    EQU     MEMST+140Q
030141                          LEN     EQU     MEMST+141Q
030142                          MULT1   EQU     MEMST+142Q
030144                          MULT2   EQU     MEMST+144Q
030131                          NXTSP   EQU     MEMST+131Q
030113                          STSPAC  EQU     MEMST+113Q
007707                          NORM    EQU     7707Q
007712                          FLOAT   EQU     7712Q
007715                          ZROL    EQU     7715Q
030122                          LPNT    EQU     MEMST+122Q
030130                          KLEN    EQU     MEMST+130Q
030133                          CPNT    EQU     MEMST+133Q
030126                          KFPNT   EQU     MEMST+126Q
030200                          FREG2   EQU     MEMST+200Q
030204                          CREG    EQU     MEMST+204Q
007720                          LADD    EQU     7720Q
007723                          LMUL    EQU     7723Q
007726                          LDIV    EQU     7726Q
007731                          LSUB    EQU     7731Q
007734                          DFXL    EQU     7734Q
007737                          LMCM    EQU     7737Q
030206                          HLINP   EQU     MEMST+206Q
030167                          GREG    EQU     MEMST+167Q
030174                          FREG1   EQU     MEMST+174Q
030146                          SCR     EQU     MEMST+146Q
007745                          CONV    EQU     7745Q
030205                          MODE    EQU     MEMST+205Q
007750                          FINPT   EQU     7750Q
007753                          MULT    EQU     7753Q
007756                          PTVAL   EQU     7756Q
007761                          DCOMP   EQU     7761Q
007764                          MCHK    EQU     7764Q
007767                          CHAR2   EQU     7767Q
030210                          MESCR   EQU     MEMST+210Q      ;DEFINE MEMORY SCR AREA PNTR
030212                          VARAD   EQU     MEMST+212Q      ;TEMP SPACE FOR INP. STMT.
030214                          VNAME   EQU     MEMST+214Q      ;TEMP SPACE FOR 'FOR-NEXT'
030216                          VLOC    EQU     MEMST+216Q      ;TEMP SPACE FOR 'FOR-NEXT'
030220                          FLIMT   EQU     MEMST+220Q      ;TEMP SPACE FOR 'FOR-NEXT'
030224                          NEST    EQU     MEMST+224Q      ;NESTING STACK-POINTER
030226                          STAC    EQU     MEMST+226Q      ;FOR-NEXT NESTING STACK
000024                          STSIZ   SET     20              ;STACK SIZE,ALLOWS 10 NESTED FOR-
030226                          TOPNS   EQU     STAC            ;TOP OF STACK                NEXT
030252                          BOTNS   EQU     STAC+STSIZ      ;BOTTOM OF STACK
030252                          VEND    EQU     MEMST+252Q      ;DEF  END OF VAR. STORAGE AREA
                                ;  MAIN ROUTINE--HANDLES ALL USER INPUT
010000                                  ORG     10000Q
010000  041 000 060            M1:     LXI     H,OBUFF
010003  066 001                         MVI     M,1
010005  041 111 060                     LXI     H,STLINE
010010  066 377                         MVI     M,377Q
010012  054                             INR     L
010013  066 377                         MVI     M,377Q
010015  052 320 037                     LHLD    FWAM            ;GET ADDRES OF FWA MEM.
010020  042 113 060                     SHLD    NLINE           ;STORE IN FREE SPACE PNTR.
010023  061 377 077            M1A:    LXI     SP,MEMEND
010026  041 132 024            M2:     LXI     H,ODATA
010031  315 110 024                     CALL    FORM1
010034  315 320 022                     CALL    WRIT
010037  052 113 060            M3:     LHLD    NLINE
010042  043                             INX     H
010043  043                             INX     H
010044  043                             INX     H
010045  043                             INX     H
010046  043                             INX     H
010047  315 311 023                     CALL    TTYIN
010052  117                             MOV     C,A
010053  376 000                         CPI     0
010055  312 037 020                     JZ      M3
010060  315 042 022                     CALL    ALPHA
010063  332 102 020                     JC      M4
010066  315 024 022                     CALL    NUMB
010071  324 236 021                     CNC     WHAT
010074  315 217 020                     CALL    INSERT
010077  303 037 020                     JMP     M3
```

```
010102  076 000          M4:     MVI     A,0
010104  315 367 022              CALL    SYMSRT
010107  074              M4A:    INR     A
010110  314 236 021              CZ      WHAT
010113  075                      DCR     A
010114  312 173 026              JZ      RUN
010117  075                      DCR     A
010120  314 252 021              CZ      TAPE
010123  312 026 020              JZ      M2
010126  075                      DCR     A
010127  314 321 021              CZ      LIST
010132  312 026 020              JZ      M2
010135  075                      DCR     A
010136  312 000 020              JZ      M1
010141  075                      DCR     A
010142  304 236 021              CNZ     WHAT
                         ; ROUTINE TO INPUT FROM HSR
010145  315 010 025      PTAPE:  CALL    CHAR5
010150  376 000                  CPI     0
010152  312 145 020              JZ      PTAPE
010155  315 025 024      PT1:    CALL    HSRIN
010160  117                      MOV     C,A
010161  376 000                  CPI     0
010163  312 145 020              JZ      PTAPE
010166  315 042 022              CALL    ALPHA
010171  332 102 020              JC      M4
010174  315 217 020              CALL    INSERT
010177  315 010 025              CALL    CHAR5
010202  376 000                  CPI     0
010204  312 026 020              JZ      M2
010207  043                      INX     H
010210  043                      INX     H
010211  043                      INX     H
010212  043                      INX     H
010213  043                      INX     H
010214  303 155 020              JMP     PT1
                         ; ROUTINE TO HANDLE ALL SOURCE LINE INPUT.
                         ; THIS INCLUDES INSERTION, DELEATION, AND
                         ; ADDITION OF NEW SOURCE LINES.
010217  053              INSERT: DCX     H
010220  161                      MOV     M,C
010221  043                      INX     H
010222  315 052 022              CALL    CVB
010225  376 005                  CPI     5
010227  332 242 020              JC      ISR1A
010232  304 236 021              CNZ     WHAT
010235  173                      MOV     A,E
010236  027                      RAL
010237  334 236 021              CC      WHAT
010242  052 113 060      ISR1A:  LHLD    NLINE
010245  162                      MOV     M,D
010246  043                      INX     H
010247  163                      MOV     M,E
010250  041 113 060              LXI     H,NLINE
010253  315 356 017              CALL    PTVAL
010256  052 111 060              LHLD    STLINE
010261  315 243 022              CALL    CHK1
010264  322 324 020              JNC     ISRT3
010267  052 113 060              LHLD    NLINE
010272  042 111 060              SHLD    STLINE
010275  026 377          ISRT1:  MVI     D,377Q
010277  132                      MOV     E,D
010300  315 213 021              CALL    STPNT
010303  043                      INX     H
010304  176              ISRT2:  MOV     A,M
010305  306 005                  ADI     5
010307  052 113 060              LHLD    NLINE
010312  205                      ADD     L
010313  157                      MOV     L,A
010314  076 000                  MVI     A,0
010316  214                      ADC     H
010317  147                      MOV     H,A
010320  042 113 060              SHLD    NLINE
010323  311                      RET
010324  042 122 060      ISRT3:  SHLD    KLINE
010327  041 122 060      ISRT4:  LXI     H,KLINE
010332  315 356 017              CALL    PTVAL
010335  041 115 060              LXI     H,NL2
010340  126                      MOV     D,M
010341  054                      INR     L
010342  136                      MOV     E,M
010343  041 124 060              LXI     H,KL2
010346  106                      MOV     B,M
010347  054                      INR     L
010350  116                      MOV     C,M
010351  315 361 017              CALL    DCOMP
010354  312 041 021              JZ      ISRT6
010357  332 172 021              JC      ISR12
010362  052 126 060              LHLD    KL4
010365  315 243 022              CALL    CHK1
010370  332 017 021              JC      ISRT5
010373  345                      PUSH    H
010374  052 122 060              LHLD    KLINE
```

```
010377  042 131 060             SHLD    PLINE
010402  041 131 060             LXI     H,PLINE
010405  315 356 017             CALL    PTVAL
010410  341                     POP     H
010411  042 122 060             SHLD    KLINE
010414  303 327 020             JMP     ISRT4
010417  052 113 060     ISRT5:  LHLD    NLINE
010422  315 221 021             CALL    NOLINE
010425  310                     RZ
010426  353                     XCHG
010427  052 122 060             LHLD    KLINE
010432  315 213 021             CALL    STPNT
010435  353                     XCHG
010436  303 275 020             JMP     ISRT1
010441  052 113 060     ISRT6:  LHLD    NLINE
010444  315 221 021             CALL    NOLINE
010447  302 110 021             JNZ     ISRT8
010452  052 111 060             LHLD    STLINE
010455  353                     XCHG
010456  052 122 060             LHLD    KLINE
010461  345                     PUSH    H
010462  301                     POP     B
010463  315 361 017             CALL    DCOMP
010466  052 126 060             LHLD    KL4
010471  312 104 021             JZ      ISRT7
010474  353                     XCHG
010475  052 131 060             LHLD    PLINE
010500  315 213 021             CALL    STPNT
010503  311                     RET
010504  042 111 060     ISRT7:  SHLD    STLINE
010507  311                     RET
010510  052 126 060     ISRT8:  LHLD    KL4
010513  353                     XCHG
010514  052 113 060             LHLD    NLINE
010517  315 213 021             CALL    STPNT
010522  052 122 060     ISRT9:  LHLD    KLINE
010525  353                     XCHG
010526  052 111 060             LHLD    STLINE
010531  345                     PUSH    H
010532  301                     POP     B
010533  315 361 017             CALL    DCOMP
010536  312 161 021             JZ      ISR11
010541  052 113 060             LHLD    NLINE
010544  353                     XCHG
010545  052 131 060             LHLD    PLINE
010550  315 213 021             CALL    STPNT
010553  041 121 060     ISR10:  LXI     H,NL6
010556  303 304 020             JMP     ISRT2
010561  052 113 060     ISR11:  LHLD    NLINE
010564  042 111 060             SHLD    STLINE
010567  303 153 021             JMP     ISR10
010572  052 122 060     ISR12:  LHLD    KLINE
010575  353                     XCHG
010576  052 113 060             LHLD    NLINE
010601  315 221 021             CALL    NOLINE
010604  310                     RZ
010605  315 213 021             CALL    STPNT
010610  303 122 021             JMP     ISRT9
                        ; ROUTINE TO STORE POINTERS INTO MEM ARRAY
010613  043             STPNT:  INX     H
010614  043                     INX     H
010615  163                     MOV     M,E
010616  043                     INX     H
010617  162                     MOV     M,D
010620  311                     RET
                        ; ROUTINE TO CHECK NEW LINE FOR SOURCE STMT.
010621  345             NOLINE: PUSH    H
010622  043                     INX     H
010623  043                     INX     H
010624  043                     INX     H
010625  043                     INX     H
010626  116                     MOV     C,M
010627  043                     INX     H
010630  315 161 022             CALL    LENGTH
010633  341                     POP     H
010634  271                     CMP     C
010635  311                     RET
                        ; ROUTINE TO RESPOND WITH 'WHAT?' FOR UNIDENTIFIED
                        ; COMMAND.
010636  041 132 024     WHAT:   LXI     H,ODATA
010641  315 104 024             CALL    FORM7
010644  315 320 022             CALL    WRIT
010647  303 023 020             JMP     M1A
                        ; ROUTINE TO PUNCH PAPER TAPE OF SOURCE.
010652  365             TAPE:   PUSH    PSW
010653  305                     PUSH    B
010654  041 132 024             LXI     H,ODATA
010657  315 107 024             CALL    FORM2
010662  315 320 022             CALL    WRIT
010665  076 000                 MVI     A,0
010667  301                     POP     B
010670  006 100                 MVI     B,100Q
010672  365                     PUSH    PSW
```

```
010673  305                     PUSH    B
010674  315 255 022             CALL    PAD
010677  315 320 022             CALL    WRIT
010702  301                     POP     B
010703  305                     PUSH    B
010704  315 321 021             CALL    LIST
010707  301                     POP     B
010710  361                     POP     PSW
010711  315 255 022             CALL    PAD
010714  315 320 022             CALL    WRIT
010717  361                     POP     PSW
010720  311                     RET
                        ; ROUTINE TO LIST TO TTY THE SOURCE STMTS.
010721  052 111 060     LIST:   LHLD    STLINE
010724  315 243 022             CALL    CHK1
010727  332 023 020             JC      M1A
010732  042 131 060             SHLD    PLINE
010735  041 377 377             LXI     H,177777Q
010740  042 122 060             SHLD    KLINE
010743  015                     DCR     C
010744  304 115 025             CNZ     BOUND
010747  052 131 060             LHLD    PLINE
010752  043             LIS1:   INX     H
010753  043                     INX     H
010754  106                     MOV     B,M
010755  043                     INX     H
010756  116                     MOV     C,M
010757  305                     PUSH    B
010760  043                     INX     H
010761  315 111 024             CALL    FORM5
010764  315 320 022             CALL    WRIT
010767  301                     POP     B
010770  052 122 060             LHLD    KLINE
010773  353                     XCHG
010774  315 361 017             CALL    DCOMP
010777  310                     RZ
011000  150                     MOV     L,B
011001  141                     MOV     H,C
011002  315 010 022             CALL    QUITT           ;CHECK FOR INTERRUPTION
011005  303 352 021             JMP     LIS1            ;NONE - CONTINUE
                        ;THIS ROUTINE CHECKS PORT 2 FOR A CNTRL/S CHARACTER
                        ;IF ONE IS FOUND THEN EXECUTION IS TO BE INTERRUPTED
                        ;CONTROL IS PASSED TO M1A
011010  333 003         QUITT:  IN      3               ;TEST FLAG PORT
011012  037                     RAR                     ;FLAG TO CY
011013  320                     RNC                     ;NOTHING THERE
011014  333 002                 IN      2               ;FLAG WAS SET, GET DATA
011016  376 223         QTCHK:  CPI     223Q            ;WAS IT CNTRL/S?
011020  312 023 020             JZ      M1A             ;YES
011023  311                     RET                     ;NO, RETURN
                        ;   ROUTINES NUMB AND ALPHA CHECK IF CONTENTS OF MEMORY
                        ;   LOCATION IN HL CONTAIN ASCII NUMERIC OR ALPHBETIC
                        ; CHARACTER. RETURN CY=1 IF YES, CY=0 IF NO.
011024  305             NUMB:   PUSH    B
011025  006 260                 MVI     B,260Q
011027  016 272                 MVI     C,272Q
011031  176             C1:     MOV     A,M
011032  270                     CMP     B
011033  077                     CMC
011034  322 040 022             JNC     BAC
011037  271                     CMP     C
011040  301             BAC:    POP     B
011041  311                     RET
011042  305             ALPHA:  PUSH    B
011043  006 301                 MVI     B,301Q
011045  016 333                 MVI     C,333Q
011047  303 031 022             JMP     C1
                        ; ROUTINE TO CONVERT ASCII NUMERIC CHAR. STRING TO
                        ; EQUIVALENT BINARY NUMBER. RETURNS EQUIVALENT IN
                        ; DE REG.  LENGTH OF LINE PASSED IN REG C AND
                        ; RETURNED POINTING TO LAST NUMERIC CHAR. LENGTH
                        ; OF CHAR STRING RETURNED IN REG A.
011052  345             CVB:    PUSH    H
011053  305                     PUSH    B
011054  315 161 022             CALL    LENGTH
011057  365                     PUSH    PSW
011060  345                     PUSH    H
011061  376 000                 CPI     0
011063  312 150 022             JZ      CVB2
011066  041 140 060             LXI     H,KASE
011071  167                     MOV     M,A
011072  054                     INR     L
011073  161                     MOV     M,C
011074  041 012 000             LXI     H,10
011077  042 142 060             SHLD    MULT1
011102  041 000 000             LXI     H,0
011105  042 144 060             SHLD    MULT2
011110  041 145 060             LXI     H,MULT2+1
011113  315 353 017     CVB1:   CALL    MULT
011116  343                     XTHL
011117  176                     MOV     A,M
011120  336 260                 SBI     260Q
011122  202                     ADD     D
```

```
011123  127                             MOV     D,A
011124  076 000                         MVI     A,0
011126  213                             ADC     E
011127  137                             MOV     E,A
011130  043                             INX     H
011131  343                             XTHL
011132  162                             MOV     M,D
011133  054                             INR     L
011134  163                             MOV     M,E
011135  345                             PUSH    H
011136  041 141 060                     LXI     H,LEN
011141  065                             DCR     M
011142  055                             DCR     L
011143  065                             DCR     M
011144  341                             POP     H
011145  302 113 022                     JNZ     CVB1
011150  341                     CVB2:   POP     H
011151  361                             POP     PSW
011152  301                             POP     B
011153  041 141 060                     LXI     H,LEN
011156  116                             MOV     C,M
011157  341                             POP     H
011160  311                             RET
                        ; ROUTINE TO EVALUATE LENGTH OF ASCII NUMERIC
                        ; CHAR STRING: PASSED ADD OF FIRST CHAR IN HL REG.
                        ; RETURNS LENGTH IN REG A.
011161  305                     LENGTH: PUSH    B
011162  345                             PUSH    H
011163  006 000                         MVI     B,0
011165  315 024 022             NLE1:   CALL    NUMB
011170  322 204 022                     JNC     NLE2
011173  043                             INX     H
011174  004                             INR     B
011175  015                             DCR     C
011176  312 204 022                     JZ      NLE2
011201  303 165 022                     JMP     NLE1
011204  170                     NLE2:   MOV     A,B
011205  341                             POP     H
011206  301                             POP     B
011207  311                             RET
                        ; ROUTINE TO LOCATE SOURCE LINE IN MEM. PASSED BIN VALUE
                        ;OF LINE NUMBER IN DE(LOW,HIGH) REG. RETURNS ADDRESS OF
                        ;SOURCE LINE IN HL REGS.(HIGH,LOW). CY SET=@ NOT FOUND.
011210  052 111 060             NSRCH:  LHLD    STLINE
011213  315 243 022             L2:     CALL    CHK1
011216  330                             RC
011217  106                             MOV     B,M
011220  043                             INX     H
011221  116                             MOV     C,M
011222  315 361 017                     CALL    DCOMP
011225  312 240 022                     JZ      FOUND
011230  043                             INX     H
011231  176                             MOV     A,M
011232  043                             INX     H
011233  146                             MOV     H,M
011234  157                             MOV     L,A
011235  303 213 022                     JMP     L2
011240  053                     FOUND:  DCX     H
011241  267                             ORA     A
011242  311                             RET
                        ; ROUTINE TO COMPARE CONTENTS OF HL TO 177777Q.
                        ; RETURNS CY=1 IF YES: CY=0 IF NO.
011243  305                     CHK1:   PUSH    B
011244  345                             PUSH    H
011245  006 000                         MVI     B,0
011247  016 001                         MVI     C,1
011251  011                             DAD     B
011252  341                             POP     H
011253  301                             POP     B
011254  311                             RET
                        ; ROUTINE TO PAD OUTPUT BUFFER WITH CONTENTS OF REG A.
                        ; REG B CONTAINS NUMBER OF CHAR TO PAD.
011255  305                     PAD:    PUSH    B
011256  325                             PUSH    D
011257  345                             PUSH    H
011260  041 000 060                     LXI     H,OBUFF
011263  115                             MOV     C,L
011264  156                             MOV     L,M
011265  127                             MOV     D,A
011266  076 111                         MVI     A,73
011270  275                     P1:     CMP     L
011271  302 302 022                     JNZ     P2
011274  151                             MOV     L,C
011275  167                             MOV     M,A
011276  315 320 022                     CALL    WRIT
011301  054                             INR     L
011302  162                     P2:     MOV     M,D
011303  054                             INR     L
011304  005                             DCR     B
011305  302 270 022                     JNZ     P1
011310  172                             MOV     A,D
011311  105                             MOV     B,L
011312  151                             MOV     L,C
```

```
011313  160                     MOV     M,B
011314  341                     POP     H
011315  321                     POP     D
011316  301                     POP     B
011317  311                     RET
                        ; ROUTINE TO DUMP OUTPUT BUFFER TO TTY.
011320  026 000        WRIT:   MVI     D,0
011322  365            WRIT1:  PUSH    PSW
011323  345                     PUSH    H
011324  305                     PUSH    B
011325  041 000 060             LXI     H,OBUFF
011330  345                     PUSH    H
011331  116                     MOV     C,M
011332  015                     DCR     C
011333  312 360 022             JZ      W2
011336  054                     INR     L
011337  176            W1:      MOV     A,M
011340  367                     RST     SEND            ;PRINT VIA ODT
011341  054                     INR     L
011342  015                     DCR     C
011343  302 337 022             JNZ     W1
011346  025                     DCR     D
011347  312 360 022             JZ      W2
011352  076 215                 MVI     A,215Q
011354  367                     RST     SEND            ;PRINT VIA ODT
011355  076 212                 MVI     A,212Q
011357  367                     RST     SEND            ;PRINT VIA ODT
011360  341            W2:      POP     H
011361  066 001                 MVI     M,1
011363  301                     POP     B
011364  341                     POP     H
011365  361                     POP     PSW
011366  311                     RET
                        ; ROUTINE TO LOCATE COMMANDS, KEY WORDS, OPERATORS,
                        ; AND FUNCTION.  HL CONTAINS ADD OF FIRST CHAR.:
                        ; REG C CONTAINS LENGTH OF LINE: RETURNS SYMBOL NUMBER
                        ; IF FOUND IN REG A, 377Q IN A IF NOT FOUND.
011367  325            SYMSRT:  PUSH    D
011370  305                     PUSH    B
011371  345                     PUSH    H
011372  345                     PUSH    H
011373  041 141 060             LXI     H,LEN           ;SAVE C IN LEN
011376  161                     MOV     M,C
011377  041 113 023             LXI     H,KDATA         ;LOCATE TYPE OF SYMBOL SOUGHT.
011402  036 000                 MVI     E,0             ;REG A CONTAINS:
011404  205                     ADD     L               ;  0 FOR COMMAND
011405  157                     MOV     L,A             ;  1 FOR KEYWORD
011406  156                     MOV     L,M             ;  2 FOR OPERATOR AND DELIMITER
011407  116            S2:      MOV     C,M             ;  3 FOR FUNCTION
011410  054            S3:      INR     L
011411  106                     MOV     B,M
011412  343                     XTHL
011413  176                     MOV     A,M
011414  270                     CMP     B
011415  302 043 023             JNZ     S4
011420  015                     DCR     C
011421  312 077 023             JZ      S5
011424  345                     PUSH    H
011425  041 141 060             LXI     H,LEN
011430  065                     DCR     M
011431  341                     POP     H
011432  312 042 023             JZ      S4A
011435  043                     INX     H
011436  343                     XTHL
011437  303 010 023             JMP     S3
011442  014            S4A:     INR     C
011443  341            S4:      POP     H
011444  171                     MOV     A,C
011445  205                     ADD     L
011446  124                     MOV     D,H
011447  341                     POP     H
011450  301                     POP     B
011451  305                     PUSH    B
011452  345                     PUSH    H
011453  345                     PUSH    H
011454  041 141 060             LXI     H,LEN
011457  161                     MOV     M,C
011460  157                     MOV     L,A
011461  142                     MOV     H,D
011462  176                     MOV     A,M
011463  034                     INR     E
011464  117                     MOV     C,A
011465  074                     INR     A
011466  302 010 023             JNZ     S3
011471  041 141 060             LXI     H,LEN
011474  064                     INR     M
011475  036 377                 MVI     E,377Q
011477  173            S5:      MOV     A,E             ; MOVE SYMBOL NUMBER INTO REG A
011500  041 141 060             LXI     H,LEN
011503  136                     MOV     E,M
011504  035                     DCR     E
011505  341                     POP     H
011506  341                     POP     H
```

```
011507  301                             POP     B
011510  113                             MOV     C,E             ;MOVE NUMBER OF CHAR. LEFT IN
011511  321                             POP     D                                       LINE INT
011512  311                             RET
                        ;*****************************************************
                        ;THE CODE FROM HERE TO THE NEXT LINE OF *'S MUST BE ON ONE PAGE
                        ;THIS MACRO ADDS PARITY BITS TO CHARACTERS
                1       CHPAR   MACRO   CH
                1               DB      CH OR 200Q
                1               ENDM
011513  117             KDATA:  DB      KDAT1 AND 377Q
011514  144                     DB      KDAT2 AND 377Q
011515  236                     DB      KDAT3 AND 377Q
011516  300                     DB      KDAT4 AND 377Q
011517  003 322 325 313 KDAT1:  DB      3,322Q,325Q,316Q        ;RUN
011523  003 320 314 323         DB      3,320Q,314Q,323Q        ;PLS
011527  003 314 311 323         DB      3,314Q,311Q,323Q        ;LIS
011533  003 323 303 322         DB      3,323Q,303Q,322Q        ;SCR
011537  003 320 324 301         DB      3,320Q,324Q,301Q        ;PTA
011543  377                     DB      377Q
011544  003 314 305 324 KDAT2:  DB      3,314Q,305Q,324Q        ;LET
011550  003 320 322 311         DB      3,320Q,322Q,311Q        ;PRI
011554  003 322 305 315         DB      3,322Q,305Q,315Q        ;REM
011560  003 323 324 317         DB      3,323Q,324Q,317Q        ;STO
011564  003 305 316 304         DB      3,305Q,316Q,304Q        ;END
011570  003 307 317 324         DB      3,307Q,317Q,324Q        ;GOT
011574  002 311 306             DB      2,311Q,306Q             ;IF
011577  003 311 316 320         DB      3,311Q,316Q,320Q        ;INP
011603  003 304 311 315         DB      3,304Q,311Q,315Q        ;DIM
011607  003 303                 DB      3,'C'+200Q              ;CAL
011611  301                     DB      'A'+200Q
011612  314                     DB      'L'+200Q
011613  004 307                 DB      4,'G'+200Q              ;GOSU
011615  317                     DB      'O'+200Q
011616  323                     DB      'S'+200Q
011617  325                     DB      'U'+200Q
011620  003 322                 DB      3,'R'+200Q              ;RET
011622  305                     DB      'E'+200Q
011623  324                     DB      'T'+200Q
011624  003 306                 DB      3,'F' OR 200Q           ;FOR
011626  317                     DB      'O' OR 200Q
011627  322                     DB      'R' OR 200Q
011630  004 316                 DB      4,'N' OR 200Q           ;NEXT
011632  305                     DB      'E' OR 200Q
011633  330                     DB      'X' OR 200Q
011634  324                     DB      'T' OR 200Q
011635  377                     DB      377Q
                        ;DELIMITERS HAVE FOLLOWING VALUES:
                        ;
                        ;               <       0
                        ;               >       1
                        ;               ,       2
                        ;               =       3
                        ;               )       4
                        ;               :       5
                        ;               THEN    6
                        ;               TO      7
                        ;               STEP    8
                        ;               *       9
                        ;               /       10
                        ;               +       11
                        ;               -       12
                        ;
011636  001 274 001 276 KDAT3:  DB      1,274Q,1,276Q           ;'<','>'
011642  001 254 001 275         DB      1,254Q,1,275Q           ;',','='
011646  001 251                 DB      1,251Q                  ;')'
011650  001 273                 DB      1,';'+200Q              ;':'
011652  004                     DB      4                       ;THEN
                1               CHPAR   'T'
011653  1 324           +       DB      'T' OR 200Q
                1       +       CHPAR   'H'
011654  1 310           +       DB      'H' OR 200Q
                1       +       CHPAR   'E'
011655  1 305           +       DB      'E' OR 200Q
                1       +       CHPAR   'N'
011656  1 316           +       DB      'N' OR 200Q
011657  002                     DB      2                       ;TO
                1       +       CHPAR   'T'
011660  1 324           +       DB      'T' OR 200Q
                1       +       CHPAR   'O'
011661  1 317           +       DB      'O' OR 200Q
011662  004                     DB      4                       ;STEP
                1       +       CHPAR   'S'
011663  1 323           +       DB      'S' OR 200Q
                1       +       CHPAR   'T'
011664  1 324           +       DB      'T' OR 200Q
                1       +       CHPAR   'E'
011665  1 305           +       DB      'E' OR 200Q
                1       +       CHPAR   'P'
011666  1 320           +       DB      'P' OR 200Q
011667  001 252                 DB      1,'*'+200Q              ;'*'
011671  001 257 001 253         DB      1,257Q,1,253Q           ;'/','+'
011675  001 255                 DB      1,255Q                  ;'-'
```

```
011677  377                     DB      377Q
011700  003 307 305 324 KDAT4:  DB      3,307Q,305Q,324Q  ;GET
011704  003 320 325 324         DB      3,320Q,325Q,324Q  ;PUT
011710  377                     DB      377Q
                        ;*****************************************************
                        ; ROUTINE TO INPUT SOURCE LINE FROM TTY. PASSED ADD
                        ; OF FIRST CHAR IN HL. RETURNS LENGTH OF LINE IN REG A
011711  345             TTYIN:  PUSH    H
011712  006 000                 MVI     B,0
011714  315 367 017     TIN1:   CALL    CHAR2
011717  376 231                 CPI     231Q            ;CNTRL Y?
011721  312 001 024             JZ      TIN5
011724  376 377                 CPI     377Q            ;RUBOUT?
011726  312 366 023             JZ      TIN2
011731  376 337                 CPI     337Q            ;BACK ARROW (RUBOUT)?
011733  312 371 023             JZ      TIN2+3
011736  376 212                 CPI     212Q            ;LF?
011740  312 314 023             JZ      TIN1
011743  376 215                 CPI     215Q            ;CR
011745  312 010 024             JZ      TIN4
011750  376 214                 CPI     214Q            ;FORM FEED?
011752  312 314 023             JZ      TIN1            ;IGNORE
011755  167                     MOV     M,A
011756  043                     INX     H
011757  004                     INR     B
011760  315 024 025             CALL    MEMFUL
011763  303 314 023             JMP     TIN1
011766  076 337     TIN2:   MVI     A,337Q
011770  367                     RST     SEND            ;PRINT VIA ODT
011771  053                     DCX     H
011772  005                     DCR     B
011773  362 314 023             JP      TIN1
011776  341                     POP     H
011777  257                     XRA     A               ;ZERO A
012000  311                     RET
012001  076 334     TIN5:   MVI     A,334Q
012003  367                     RST     SEND            ;PRINT VIA ODT
012004  076 000     TIN5A:  MVI     A,0
012006  341                     POP     H
012007  311                     RET
012010  076 212     TIN4:   MVI     A,212Q
012012  367                     RST     SEND            ;PRINT VIA ODT
012013  016 000     TIN4A:  MVI     C,0
012015  341                     POP     H
012016  170                     MOV     A,B
012017  271                     CMP     C
012020  310                     RZ
                        ;ROUTINE TO REMOVE BLANKS FROM SOURCE UNLESS ENCLOSED IN !'S
012021  325                     PUSH    D               ;SAVE REG'S
012022  345                     PUSH    H
012023  345                     PUSH    H
012024  036 242                 MVI     E,'"'+200Q      ;INIT E FOR COMPARES
012026  026 000                 MVI     D,0             ;D=1=>WITHIN QUOTES,LEAVE BLANKS
012030  257             PK1:    XRA     A               ;CLEAR A
012031  272                     CMP     D               ;CHECK INPUT MODE
012032  176                     MOV     A,M             ;GET CHAR
012033  302 053 024             JNZ     QSTRG           ;WITHIN QUOTE STRING
012036  273                     CMP     E               ;IS IT 1ST !?
012037  302 046 024             JNZ     $+7             ;NO - PROCEED
012042  024                     INR     D               ;YES, SET FLAG
012043  303 060 024             JMP     QSTR1           ;CONTINUE
012046  376 240                 CPI     240Q            ;IS IT A SPACE?
012050  312 065 024             JZ      PK2             ;YES - LEAVE OUT
012053  273             QSTRG:  CMP     E               ;2ND "?
012054  302 060 024             JNZ     $+4             ;NO - CONTINUE
012057  025                     DCR     D               ;RESET FLAG
012060  343             QSTR1:  XTHL                    ;GET DESTINATION ADDRESS
012061  167                     MOV     M,A             ;SAVE
012062  043                     INX     H               ;BUMP PNTR.
012063  343                     XTHL                    ;GET SOURCE ADD.
012064  014                     INR     C               ;BUMP CHAR. CNT
012065  043             PK2:    INX     H               ;BUMP PNTR.
012066  005                     DCR     B               ;DCR INPUT LINE CHAR CNT
012067  302 030 024             JNZ     PK1             ;MORE - GO AGAIN
012072  171                     MOV     A,C             ;CHAR CNT TO A
012073  341                     POP     H               ;RESTORE REG'S, RETURN
012074  341                     POP     H
012075  321                     POP     D
012076  311                     RET
                        ; ROUTINES TO  PAD MESSAGES TO  OUTPUT BUFFER.
                        ; FOR12 PADS 'UNDERFLOW'
                        ; FOR11 PADS 'OVERFLOW'
                        ; FOR10 PADS 'ZERODIVIDE'
                        ; FORM9 PADS 'INPUT ERROR, TRY AGAIN'
                        ; FORM8 PADS 'MEMORY FULL'
                        ; FORM7 PADS 'WHAT?'
                        ; FORM4 PADS 'IN LINE'
                        ; FORM3 PADS 'ERROR'
                        ; FORM2 PADS 'TURN ON PUNCH'
                        ; FORM1 PADS 'READY'
                        ; FORM5 PADS SOURCE LINE, PASSED ADDRESS OF
                        ; LENGTH OF LINE IN HL REGS.
                        ; FORM6 PADS CHAR STRING, PASSED ADD OF FIRST CHAR IN
                        ; HL, LENGTH OF STRING IN REG C
```

```
012077  054                     FOR12:  INR     L
012100  054                     FOR11:  INR     L
012101  054                     FOR10:  INR     L
012102  054                     FORM9:  INR     L
012103  054                     FORM8:  INR     L
012104  054                     FORM7:  INR     L
012105  054                     FORM4:  INR     L
012106  054                     FORM3:  INR     L
012107  054                     FORM2:  INR     L
012110  156                     FORM1:  MOV     L,M
012111  116                     FORM5:  MOV     C,M
012112  171                             MOV     A,C
012113  376 000                         CPI     0
012115  310                             RZ
012116  043                     F1:     INX     H
012117  176                     FORM6:  MOV     A,M
012120  006 001                         MVI     B,1
012122  315 255 022                     CALL    PAD
012125  015                             DCR     C
012126  302 116 024                     JNZ     F1
012131  311                             RET
                        ;**********************************************
                        ;THE CODE FROM HERE TO THE NEXT LINE OF *'S MUST BE ON ONE PAGE
012132  144                     0DATA:  DB      0DAT1 AND 377Q
012133  152                             DB      0DAT2 AND 377Q
012134  170                             DB      0DAT3 AND 377Q
012135  201                             DB      0DAT4 AND 377Q
012136  213                             DB      0DAT5 AND 377Q
012137  221                             DB      0DAT6 AND 377Q
012140  240                             DB      0DAT7 AND 377Q
012141  267                             DB      0DAT8 AND 377Q
012142  302                             DB      0DAT9 AND 377Q
012143  313                             DB      0DA10 AND 377Q
012144  005 122 105 101 0DAT1:  DB      5,'READY'
012150  104 131
012152  015 124 125 122 0DAT2:  DB      13,'TURN ON PUNCH'
012156  116 040 117 116
012162  040 120 125 116
012166  103 110
012170  010 215 212 105 0DAT3:  DB      8,215Q,212Q,'ERROR
012174  122 122 117 122
012200  040
012201  011 040 111 116 0DAT4:  DB      9,' IN LINE '
012205  040 114 111 116
012211  105 040
012213  005 127 110 101 0DAT5:  DB      5,'WHAT?'
012217  124 077
012221  016 115 105 115 0DAT6:  DB      14,'MEMORY FULL',215Q,212Q,'?'
012225  117 122 131 040
012231  106 125 114 114
012235  215 212 077
012240  026 111 116 120 0DAT7:  DB      22,'INPUT ERROR, TRY AGAIN'
012244  125 124 040 105
012250  122 122 117 122
012254  054 040 124 122
012260  131 040 101 107
012264  101 111 116
012267  012 111 116 104 0DAT8:  DB      10,'INDEFINITE'
012273  105 106 111 116
012277  111 124 105
012302  010 117 126 105 0DAT9:  DB      8,'OVERFLOW'
012306  122 106 114 117
012312  127
012313  011 125 116 104 0DA10:  DB      9,'UNDERFLOW'
012317  105 122 106 114
012323  117 127
                        ;**********************************************
                        ; ROUTINE TO INPUT SOURCE LINE FROM HSR. PASSED ADD
                        ; OF FIRST CHAR IN HL. RETURNS LENGTH OF LINE IN REG A
012325  345                     HSRIN:  PUSH    H
012326  006 000                         MVI     B,0
012330  303 336 024                     JMP     PIN1A
012333  315 010 025             PIN1:   CALL    CHAR5
012336  376 231                 PIN1A:  CPI     231Q            ;CNTRL Y?
012340  312 004 024                     JZ      TIN5A
012343  376 377                         CPI     377Q
012345  312 000 025                     JZ      PIN3
012350  376 337                         CPI     337Q
012352  312 000 025                     JZ      PIN3
012355  376 212                         CPI     212Q
012357  312 013 024                     JZ      TIN4A
012362  376 215                         CPI     215Q
012364  312 333 024                     JZ      PIN1
012367  167                             MOV     M,A
012370  043                             INX     H
012371  004                             INR     B
012372  315 024 025                     CALL    MEMFUL
012375  303 333 024                     JMP     PIN1
012400  053                     PIN3:   DCX     H
012401  005                             DCR     B
012402  362 333 024                     JP      PIN1
012405  341                             POP     H
012406  257                             XRA     A               ;ZERO A
012407  311                             RET
```

```
                                ; ROUTINE TO INPUT CHAR FROM HSR
012410  305                     CHAR5:  PUSH    B
012411  006 004                         MVI     B,4
012413  323 005                         OUT     5
012415  315 334 017                     CALL    MCHK
012420  333 005                         IN      5
012422  301                             POP     B
012423  311                             RET
                                ; ROUTINE TO INSURE SOURCE DOES NOT OVERFLOW MEM SPACE
                                ; COMPARES CURENT MEM ADDRESS TO SP.
012424  305                     MEMFUL: PUSH    B
012425  325                             PUSH    D
012426  345                             PUSH    H
012427  076 062                         MVI     A,50
012431  205                             ADD     L
012432  107                             MOV     B,A
012433  076 000                         MVI     A,0
012435  214                             ADC     H
012436  117                             MOV     C,A
012437  041 000 000                     LXI     H,0
012442  071                             DAD     SP
012443  125                             MOV     D,L
012444  134                             MOV     E,H
012445  315 361 017                     CALL    DCOMP
012450  341                             POP     H
012451  321                             POP     D
012452  301                             POP     B
012453  320                             RNC
012454  041 132 024                     LXI     H,ODATA
012457  315 103 024                     CALL    FORM8
012462  315 320 022                     CALL    WRIT
012465  315 367 017                     CALL    CHAR2
012470  315 255 022                     CALL    PAD
012473  315 320 022                     CALL    WRIT
012476  336 260                         SBI     260Q
012500  376 004                         CPI     4
012502  314 236 021                     CZ      WHAT
012505  061 377 077                     LXI     SP,MEMEND
012510  016 001                         MVI     C,1
012512  303 107 020                     JMP     M4A
                                ; ROUTINE TO EVALUATE BOUNDS FOR LIST AND PLIST
                                ; COMMANDS. RETURNS PLINE AS FIRST LINE, KLINE
                                ; AS LAST LINE TO BE LISTED.
012515  052 113 060             BOUND:  LHLD    NLINE
012520  076 011                         MVI     A,9
012522  205                             ADD     L
012523  157                             MOV     L,A
012524  076 000                         MVI     A,0
012526  214                             ADC     H
012527  147                             MOV     H,A
012530  345                             PUSH    H
012531  315 024 022                     CALL    NUMB
012534  324 236 021                     CNC     WHAT
012537  315 052 022                     CALL    CVB
012542  365                             PUSH    PSW
012543  305                             PUSH    B
012544  315 243 025                     CALL    BND2
012547  301                             POP     B
012550  053                             DCX     H
012551  042 131 060                     SHLD    PLINE
012554  361                     BND1:   POP     PSW
012555  341                             POP     H
012556  074                             INR     A
012557  205                             ADD     L
012560  157                             MOV     L,A
012561  076 000                         MVI     A,0
012563  214                             ADC     H
012564  147                             MOV     H,A
012565  076 000                         MVI     A,0
012567  271                             CMP     C
012570  310                             RZ
012571  015                             DCR     C
012572  315 024 022                     CALL    NUMB
012575  324 236 021                     CNC     WHAT
012600  325                             PUSH    D
012601  315 052 022                     CALL    CVB
012604  325                             PUSH    D
012605  305                             PUSH    B
012606  315 243 025                     CALL    BND2
012611  301                             POP     B
012612  043                             INX     H
012613  126                             MOV     D,M
012614  043                             INX     H
012615  136                             MOV     E,M
012616  353                             XCHG
012617  042 122 060                     SHLD    KLINE
012622  321                             POP     D
012623  341                             POP     H
012624  171                             MOV     A,C
012625  376 000                         CPI     0
012627  302 236 021                     JNZ     WHAT
012632  104                             MOV     B,H
012633  115                             MOV     C,L
```

```
012634  315 361 017             CALL    DCOMP
012637  320                     RNC
012640  303 236 021             JMP     WHAT
012643  052 111 060     BND2:   LHLD    STLINE
012646  106             BND3:   MOV     B,M
012647  043                     INX     H
012650  116                     MOV     C,M
012651  315 361 017             CALL    DCOMP
012654  330                     RC
012655  310                     RZ
012656  345                     PUSH    H
012657  043                     INX     H
012660  176                     MOV     A,M
012661  043                     INX     H
012662  146                     MOV     H,M
012663  157                     MOV     L,A
012664  315 243 022             CALL    CHK1
012667  301                     POP     B
012670  322 246 025             JNC     BND3
012673  305                     PUSH    B
012674  341                     POP     H
012675  311                     RET
                        ; ROUTINE TO OUTPUT ERROR MSG. TO USER.
                        ; REG A CONTAINS BCD ERROR NUMBER, HL
                        ; LOADED WITH VALUE OF KLINE.
012676  041 023 020     ERROR:  LXI     H,M1A           ;RETURN ADDRESS
012701  345                     PUSH    H               ;PUT ON STACK
012702  041 132 024             LXI     H,ODATA         ;OUTPUT BUFFER DATA TABLES
012705  345                     PUSH    H
012706  127                     MOV     D,A             ;SAVE ERROR NUMB. IN D
012707  315 106 024             CALL    FORM3           ;PAD 'ERROR '
012712  006 001                 MVI     B,1             ;INIT FOR PADS
012714  110                     MOV     C,B             ;INIT AS CNTR.
012715  172                     MOV     A,D             ;GET ERROR NUMB.
012716  007                     RLC                     ;ROTATE HIGH 4 BITS TO LOW 4
012717  007                     RLC
012720  007                     RLC
012721  007                     RLC
012722  346 017         ERRR1:  ANI     17Q             ;MASK
012724  306 260                 ADI     260Q            ;CONVERT TO ASCII
012726  315 255 022             CALL    PAD             ;PAD IT
012731  172                     MOV     A,D             ;GET ERROR NUMB.
012732  015                     DCR     C               ;ANOTHER PASS?
012733  362 322 025             JP      ERRR1           ;YES
012736  341                     POP     H               ;NO-CONTINUE
012737  315 105 024     ERLN:   CALL    FORM4
012742  052 122 060             LHLD    KLINE
012745  043                     INX     H
012746  043                     INX     H
012747  043                     INX     H
012750  043                     INX     H
012751  116                     MOV     C,M
012752  043                     INX     H
012753  315 161 022             CALL    LENGTH
012756  117                     MOV     C,A
012757  315 117 024             CALL    FORM6
012762  315 320 022             CALL    WRIT
012765  311                     RET
                        ;THIS ROUTINE INCREMENTS H AND L AND
                        ;DECR. C(CHARS IN LINE) SHOULD C RESULT
                        ;IN 0 THEN THE ERROR CORRES. TO ENTRY PNT.
                        ;IS GIVEN
012766  076 007         ICP7:   MVI     A,7
012770  303 007 026             JMP     INCPT
012773  076 010         ICP8:   MVI     A,8
012775  303 007 026             JMP     INCPT
013000  076 004         ICP4:   MVI     A,4
013002  303 007 026             JMP     INCPT
013005  076 002         ICP2:   MVI     A,2
013007  043             INCPT:  INX     H
013010  015                     DCR     C
013011  300                     RNZ
013012  303 276 025             JMP     ERROR
                        ;FSYM FINDS SYMBOLS IN TABLE
                        ;B,C CONTAIN SYMBOL
                        ;RET WITH B,C,D,E SAME
                        ;H AND L PNT TO VALUE (1ST BYTE)
                        ;CY=1   =@ FOUND
                        ;CY=0  AND A SCALAR VAR. =@ INSERTED
                        ;       AND SET TO 0
                        ;CY=0  AND AN ARRAY  =@ NO ACTION,
                        ;       H AND L PNT TO LAST ENTRY IN SYMBOL TABLE
013015  325             FSYM:   PUSH    D
013016  257                     XRA     A
013017  260                     ORA     B               ;SET CARRY IF NOT
013020  312 024 026             JZ      AR              ;AN ARRAY AND SAVE
013023  077                     CMC
013024  365             AR:     PUSH    PSW
013025  052 131 060             LHLD    NXTSP           ;GET NEXT AVAILABLE
013030  305                     PUSH    B               ;SPACE PNTR.
013031  104                     MOV     B,H
013032  115                     MOV     C,L             ;CHECK TO SEE
013033  052 113 060             LHLD    STSPAC          ;IF SYMBOL TABLE
                                                        ;EMPTY
```

```
013036  124                     MOV     D,H
013037  135                     MOV     E,L
013040  315 361 017             CALL    DCOMP           ;DOUBLE BYTE COMPARE
013043  301                     POP     B               ;GET VAR. BACK
013044  312 104 026             JZ      NOSYM
013047  315 243 022     LUKON:  CALL    CHK1            ;CHECK FOR END
013052  332 106 026             JC      NOENT
013055  124                     MOV     D,H             ;SAVE OLD PNTR
013056  135                     MOV     E,L
013057  170                     MOV     A,B
013060  276                     CMP     M               ;DO VARIABLES MATCH
013061  302 073 026             JNZ     NOMAT
013064  043                     INX     H
013065  171                     MOV     A,C
013066  276                     CMP     M
013067  312 164 026             JZ      ENTRY
013072  053                     DCX     H
013073  043             NOMAT:  INX     H               ;NO MATCH GET NEW PNT.
013074  043                     INX     H
013075  176                     MOV     A,M
013076  043                     INX     H
013077  146                     MOV     H,M
013100  157                     MOV     L,A
013101  303 047 026             JMP     LUKON
                        ;ARRIVE HERE IF SYMBOL TABLE IS EMPTY
013104  033             NOSYM:  DCX     D               ; =STSPAC-2 SO STPNT WORKS RIGHT
013105  033                     DCX     D
                        ;ARRIVE HERE WHEN NO ENTRY FOUND
013106  052 131 060     NOENT:  LHLD    NXTSP           ;ADD. OF FREE MEMORY
013111  353                     XCHG                    ;TO DE, HL HAVE LAST SYM.TAB.
013112  361                     POP     PSW             ;ARRAY?                 ENTRY
013113  322 171 026             JNC     FBAC            ;YES, RETURN
013116  315 260 037             CALL    CHKLC           ;CHECK FOR PAGE BOUNDARY CROSSING
013121  315 213 021             CALL    STPNT           ;UPDATE PNTR
013124  353                     XCHG                    ;NXTSP TO HL
013125  160                     MOV     M,B             ;STORE VAR.
013126  043                     INX     H
013127  161                     MOV     M,C
013130  043                     INX     H
013131  345                     PUSH    H
013132  043                     INX     H               ;STORE NXTSP+8 IN NXTSP
013133  043                     INX     H
013134  043                     INX     H
013135  043                     INX     H
013136  043                     INX     H
013137  043                     INX     H
013140  042 131 060             SHLD    NXTSP
013143  315 024 025             CALL    MEMFUL          ;MEMORY FULL?
013146  341                     POP     H               ;SET FWD PNT. TO -1
013147  066 377                 MVI     M,377Q
013151  043                     INX     H
013152  066 377                 MVI     M,377Q
013154  043                     INX     H               ;INIT TO FLT. PNT. 0
013155  315 315 017             CALL    ZROL
013160  267                     ORA     A               ;CLEAR CY
013161  303 171 026             JMP     FBAC            ;RESET CARRY AND RETURN
013164  361             ENTRY:  POP     PSW             ;VAR FOUND
013165  043                     INX     H               ;MOVE PNT. TO FIRST BYTE
013166  043                     INX     H               ;OF FLT. PNT. NO.
013167  043                     INX     H
013170  067                     STC                     ;SET CY AND RET.
013171  321             FBAC:   POP     D               ;RESTORE D
013172  311                     RET
                        ;
                        ;
                        ;RUN - THE INTERP.
                        ;
                        ;
                        ;INIT. NXTSP
013173  052 113 060     RUN:    LHLD    STSPAC
013176  353                     XCHG
013177  315 301 037             CALL    CKDIM           ; ADJUST START OF SYMBOL TABLE SO
                                                        ; IT STARTS ON AN EVEN 4 WORD BOUN
013202  315 260 037             CALL    CHKLC           ; ADJUST START OF SYMBOL TABLE SO IT
                                                        ; DOES NOT CROSS PAGE BOUNDARY
013205  353                     XCHG
013206  042 113 060             SHLD     STSPAC
013211  042 131 060             SHLD    NXTSP
013214  041 252 060             LXI     H,BOTNS         ;INIT SP FOR NESTING STACK
013217  042 224 060             SHLD    NEST
013222  041 023 020             LXI     H,M1A           ;PRECAUTION, IN CASE RETURN IS
013225  345                     PUSH    H               ;EXECUTED BEFORE A GOSUB
013226  345                     PUSH    H
013227  052 111 060             LHLD    STLINE          ;START OF SOURCE
013232  315 010 022     ILOOP:  CALL    QUITT           ;CHECK FOR INTERRUPTION
013235  315 243 022             CALL    CHK1            ;HL=-1 =@ NO MORE SOURCE
013240  322 250 026             JNC     SORCE
013243  076 001                 MVI     A,1
013245  303 276 025             JMP     ERROR           ;ERROR 1, NO END STMT.
013250  042 122 060     SORCE:  SHLD    LPNT
013253  345                     PUSH    H
013254  041 122 060             LXI     H,LPNT          ;DEFINE VALUES OF
013257  315 356 017             CALL    PTVAL           ;KBIN,KFPNT,KLEN
013262  072 130 060             LDA     KLEN            ;CHAR'S IN LINE TO C
```

```
013265  117                             MOV     C,A
013266  014                             INR     C
013267  341                             POP     H               ;MOVE PNTR. TO 1ST CHAR
013270  043                             INX     H               ;IN SOURCE REC.
013271  043                             INX     H
013272  043                             INX     H
013273  043                             INX     H
013274  315 005 026     L1:             CALL    ICP2            ;INCR. H,L DCR C
013277  315 042 022                     CALL    ALPHA           ;FIND FIRST LETTER
013302  322 274 026                     JNC     L1
013305  257                             XRA     A
013306  074                             INR     A               ;LETTER FOUND
013307  315 367 022                     CALL    SYMSRT          ;DETERMINE KEYWORD
013312  376 377                         CPI     377Q
013314  302 324 026                     JNZ     GKEY
013317  076 002                         MVI     A,2             ;BAD KEYWORD
013321  303 276 025                     JMP     ERROR
013324  042 133 060     GKEY:           SHLD    CPNT
013327  041 344 026                     LXI     H,JTBL          ;LOAD JUMP TABLE PNTR.
013332  207                             ADD     A               ;DOUBLE A
013333  137                             MOV     E,A
013334  026 000                         MVI     D,0
013336  031                             DAD     D               ;PNT. TO PROPER PROC.
013337  176                             MOV     A,M             ;ADD. IN JUMP TABLE
013340  043                             INX     H               ;GET PROC. ADD.
013341  146                             MOV     H,M
013342  157                             MOV     L,A
013343  351                             PCHL                    ;INDIRECT JUMP TO PROC.
013344  363 033         JTBL:           DW      LET             ;JMP TABLE
013346  253 032                         DW      PRI
013350  242 033                         DW      IEND            ;REM STMT. - NO ACTION
013352  023 020                         DW      M1A             ;STOP STMT.-RETURN TO EDIT MODE
013354  000 027                         DW      ENDD
013356  016 027                         DW      GOTO
013360  223 034                         DW      IFRT
013362  142 033                         DW      INPUT
013364  056 027                         DW      DIM
013366  160 035                         DW      CALLP
013370  346 035                         DW      GOSUB
013372  373 035                         DW      RETRN
013374  375 035                         DW      FOR
013376  357 036                         DW      NEXT
013400  052 126 060     ENDD:           LHLD    KFPNT           ;CHECK TO SEE IF MORE
013403  315 243 022                     CALL    CHK1            ;SOURCE AFTER END
013406  332 023 020                     JC      M1A
013411  076 003                         MVI     A,3             ;MORE SOURCE ERROR 3
013413  303 276 025                     JMP     ERROR
013416  052 133 060     GOTO:           LHLD    CPNT            ;GOTO STMT. PROC.
013421  043             GSENT:          INX     H               ;INCREMENT PAST KEYWORD
013422  043                             INX     H
013423  043                             INX     H
013424  315 000 026                     CALL    ICP4            ;POSSIBLE ERROR 4
013427  315 052 022     GTRA:           CALL    CVB             ;GET DESTINATION
013432  267                             ORA     A               ;MAKE SURE IT WAS OK
013433  302 043 027                     JNZ     OKN
013436  076 004                         MVI     A,4
013440  303 276 025                     JMP     ERROR
013443  315 210 022     OKN:            CALL    NSRCH           ;GET NEXT LPNT
013446  322 232 026                     JNC     ILOOP           ;MAKE SURE IT EXISTED
013451  076 005                         MVI     A,5
013453  303 276 025                     JMP     ERROR           ;NON-EXISTENT
013456  052 133 060     DIM:            LHLD    CPNT            ;DIM STMT. PROC.
013461  043                             INX     H               ;PNT TO FIRST VAR.
013462  043                             INX     H
013463  043                             INX     H
013464  315 042 022     DLOOP:          CALL    ALPHA           ;CHECK IF IT IS A VAR.
013467  332 077 027                     JC      OKLET
013472  076 006         ER6:            MVI     A,6             ;ERROR 6
013474  303 276 025                     JMP     ERROR
013477  106             OKLET:          MOV     B,M
013500  315 366 025                     CALL    ICP7            ;INCR.CPNT
013503  076 250                         MVI     A,250Q          ;CHECK FOR (
013505  276                             CMP     M
013506  302 072 027                     JNZ     ER6
013511  315 366 025                     CALL    ICP7            ;INCR. CPNT
013514  315 052 022                     CALL    CVB             ;CONV. TO BIN NO.
013517  205                             ADD     L               ;UPDATE CPNT
013520  157                             MOV     L,A             ;ED CONTAIN ARRAY LEN.
013521  076 000                         MVI     A,0
013523  214                             ADC     H               ;C CONT. NO. CHARS LEFT
013524  147                             MOV     H,A             ;IN LINE
013525  076 251                         MVI     A,251Q          ;CHECK FOR )
013527  276                             CMP     M
013530  302 072 027                     JNZ     ER6
013533  345                             PUSH    H
013534  305                             PUSH    B               ;SAVE B,C,H,L
013535  110                             MOV     C,B             ;SET UP FOR CALL TO FSYM
013536  006 000                         MVI     B,0
013540  315 015 026                     CALL    FSYM
013543  322 155 027                     JNC     NDOU
013546  301                             POP     B
013547  341                             POP     H
013550  076 021                         MVI     A,11H           ;ERROR 11
```

```
013552  303 276 025             JMP     ERROR           ;DUPLICATE ARRAY DEF.
013553  325             NDOU:   PUSH    D               ;SAVE DIM. LENGTH
013556  353                     XCHG                    ;ADD. OF LAST SYM. TAB.ENTRY TO DE
013557  052 131 060             LHLD    NXTSP           ;GET ADD. OF AVAILABLE MEM.
013562  353                     XCHG                    ;SET UP FOR CALL
013563  315 301 037             CALL    CKDIM           ; CHECK START OF 'DIM' ARRAY
013566  315 213 021             CALL    STPNT           ;STORE NEW PNTR
013571  353                     XCHG                    ;NXTSP TO HL
013572  321                     POP     D               ;RESTORE D
013573  066 000                 MVI     M,0
013575  043                     INX     H               ;INSERT VAR IN SYMB. TAB.
013576  161                     MOV     M,C
013577  043                     INX     H
013600  066 377                 MVI     M,377Q          ;FPNT TO -1
013602  043                     INX     H
013603  066 377                 MVI     M,377Q
013605  043                     INX     H               ;PNTS TO FIRST DATA
013606  172                     MOV     A,D             ;GET ONE'S COMPLEMENT OF
013607  057                     CMA                     ;NUMBER OF ELEMENTS
013610  117                     MOV     C,A             ;IN ARRAY TO B,C
013611  173                     MOV     A,E
013612  057                     CMA
013613  107                     MOV     B,A
013614  315 315 017     CONT:   CALL    ZROL            ;ZEROE OUT ELEMTS.
013617  043                     INX     H               ; OF ARRAY
013620  043                     INX     H
013621  043                     INX     H
013622  043                     INX     H
013623  003                     INX     B
013624  345                     PUSH    H
013625  315 024 025             CALL    MEMFUL          ;MEMORY FULL?
013630  140                     MOV     H,B
013631  151                     MOV     L,C
013632  315 243 022             CALL    CHK1
013635  341                     POP     H
013636  322 214 027             JNC     CONT
013641  042 131 060             SHLD    NXTSP           ;NEW VALUE OF NXTSP.
013644  301                     POP     B               ;RESTORE REG'S
013645  341                     POP     H
013646  043                     INX     H
013647  015                     DCR     C               ;MORE ELEMTS IN LINE?
013650  312 242 033             JZ      IEND
013653  015                     DCR     C
013654  312 072 027             JZ      ER6
013657  076 254                 MVI     A,254Q          ;NEXT ELEMENT A ,
013661  276                     CMP     M
013662  043                     INX     H
013663  312 064 027             JZ      DLOOP
013666  303 072 027             JMP     ER6
                        ;ROUTINE TO COPY CONTENTS PNTED TO
                        ;BY DE TO LOCATION H,L
013671  365             COPDH:  PUSH    PSW             ;SAVE REGISTERS
013672  305                     PUSH    B
013673  325                     PUSH    D
013674  345                     PUSH    H
013675  006 004                 MVI     B,4             ;COUNT
013677  032             COPD1:  LDAX    D               ;GET FROM SOURCE
013700  167                     MOV     M,A             ;PUT TO DESTINATION
013701  023                     INX     D               ;BUMP PNTRS, CNT
013702  043                     INX     H
013703  005                     DCR     B
013704  302 277 027             JNZ     COPD1
013707  341                     POP     H               ;RESTORE REGISTERS
013710  321                     POP     D
013711  301                     POP     B
013712  361                     POP     PSW
013713  311                     RET
                        ;OUTR PADS OUTPUT FROM CONV INTO
                        ;OUTPUT BUFFER USING ROUTINE PAD
                        ;ALL REG'S MAINTAINED
013714  305             OUTR:   PUSH    B               ;SAVE REG B
013715  006 001                 MVI     B,1             ;PAD ONCE
013717  315 255 022             CALL    PAD             ;DO IT
013722  301                     POP     B               ;RESTORE B AND RET.
013723  311                     RET
                        ;VALUE RETURNS IN D(H),E(L) PNTR.
                        ;TO THE VALUE OF A TOKEN
                        ;C,H,L ARE UPDATED
                        ;A,B ARE DESTROYED
013724  315 203 030     VALUE:  CALL    VAR             ;IS IT A VARIABLE?
013727  330                     RC                      ;YES - ALL DONE
013730  076 003                 MVI     A,3             ;NO CHEK IF A FUNC.
013732  315 367 022             CALL    SYMSRT
013735  376 377                 CPI     377Q
013737  312 114 030             JZ      KONT            ;NOT A FUNCTION -
013742  376 001                 CPI     1               ;WAS IT PUT(--)?
013744  302 352 027             JNZ     GET             ;NO - OK
013747  303 216 034             JMP     ER10            ;ILLEGAL USE OF FUNCTION
013752  043             GET:    INX     H               ;OK, IT'S GET(--)
013753  043                     INX     H               ;UPDATE H,L
013754  043                     INX     H
013755  171                     MOV     A,C             ;CHECK FOR PREMATURE EOL
013756  267                     ORA     A
```

```
013757  312 070 032             JZ      ER8
013762  076 250                 MVI     A,250Q          ;CHEK FOR (
013764  276                     CMP     M
013765  302 070 032             JNZ     ER8
013770  315 373 025             CALL    ICP8            ;BUMP PNTR'S
013773  315 335 031             CALL    EVAL            ;GET PORT =
013776  345                     PUSH    H               ;SAVE REG H,L
013777  041 174 060             LXI     H,FREG1
014002  315 271 027             CALL    COPDH           ;COPY IT
014005  353                     XCHG
014006  341                     POP     H               ;RESTORE H,L
014007  315 053 031             CALL    FIX             ;FIX IT
014012  023                     INX     D
014013  023                     INX     D               ;GET LOWEST BYTE TO
014014  023                     INX     D               ;REG D
014015  032                     LDAX    D
014016  127                     MOV     D,A
014017  171                     MOV     A,C             ;EOL?
014020  267                     ORA     A
014021  312 070 032             JZ      ER8
014024  076 251                 MVI     A,251Q          ;CHECK FOR )
014026  276                     CMP     M
014027  302 070 032             JNZ     ER8
014032  043                     INX     H               ;BUMP PNTR'S
014033  015                     DCR     C
014034  345                     PUSH    H               ;SAVE H,L,B,C
014035  305                     PUSH    B               ;STORE PROGRAM SEGMENT
014036  001 167 060             LXI     B,GREG          ;IN RAM,START AT GREG
014041  041 107 030             LXI     H,RINST         ;ADD. OF INST'S
014044  036 005                 MVI     E,5             ;NUMB. OF BYTES
014046  176             V1:     MOV     A,M             ;GET BYTE
014047  002                     STAX    B               ;STORE IN RAM
014050  043                     INX     H
014051  003                     INX     B
014052  035                     DCR     E               ;BUMP PNTR'S,DCR CNT
014053  302 046 030             JNZ     V1
014056  041 170 060             LXI     H,GREG+1        ;STORE PORT =
014061  162                     MOV     M,D             ;IN RAM
014062  303 167 060             JMP     GREG            ;OK - TRANSFER
014065  041 171 060     HOME:   LXI     H,GREG+2        ;SET UP FOR FLOAT
014070  167                     MOV     M,A             ;STORE AWAY INPUT
014071  053                     DCX     H
014072  257                     XRA     A               ;ZERO OUT HIGHER BYTES
014073  167                     MOV     M,A             ;BUT CHAR. DOESN'T MATTER
014074  053                     DCX     H
014075  167                     MOV     M,A
014076  315 234 017             CALL    DFXL            ;FLOAT IT
014101  021 167 060             LXI     D,GREG          ;FIX D,E RESTORE C,H,L
014104  301                     POP     B
014105  341                     POP     H
014106  311                     RET
014107  333 000         RINST:  IN      0               ;RAM INSTRUCTIONS
014111  303 065 030             JMP     HOME
014114  315 024 022     KONT:   CALL    NUMB            ;NUMBER
014117  332 130 030             JC      OKK
014122  076 256                 MVI     A,256Q          ;DEC. PNT.?
014124  276                     CMP     M
014125  302 070 032             JNZ     ER8
014130  076 001         OKK:    MVI     A,1             ;MODE=1, IE. INPUT FROM SOURCE
014132  315 144 030             CALL    RDKON           ;READ CONSTANT TO GREG
014135  332 176 030             JC      ER9             ;IF ERROR THEN CY=1
014140  021 167 060             LXI     D,GREG          ;PNTS. TO CONSTANT
014143  311                     RET
                        ;THIS ROUTINE READS A CONSTANT INTO GREG FROM ASCII
                        ;CHARACTERS POINTED TO BY HL AND C
                        ;ENTER WITH A=0 => DATA FROM TTY
                        ;ENTER WITH A=1 => DATA FROM SOURCE
                        ;RETURN WITH CY=1 => ERROR IN CONVERSION
014144  062 205 060     RDKON:  STA     MODE            ;SAVE MODE FOR ROUT. INP
014147  042 206 060             SHLD    HLINP           ;SAVE HL FOR ROUT. INP
014152  171                     MOV     A,C
014153  062 204 060             STA     CREG            ;SAVE C FOR ROUT. INP
014156  041 167 060             LXI     H,GREG          ;WHER VALUE WILL GO
014161  016 146                 MVI     C,SCR AND 377Q  ;SET UP AND CALL FINPT
014163  315 350 017             CALL    FINPT
014166  052 206 060             LHLD    HLINP           ;RETORE H,L AND C
014171  072 204 060             LDA     CREG
014174  117                     MOV     C,A
014175  311                     RET                     ;DONE
014176  076 011         ER9:    MVI     A,9
014200  303 276 025             JMP     ERROR
                        ;VAR DECIDES WHETHER A TOKEN IS
                        ;A VARIABLE IF SO CY=1 AND
                        ;ADDRESS IS COMPUTED,(SUBSCRIPT IS
                        ;EVALUATED ETC.), RETURNS WITH DE PNTING
                        ;TO VAR. REFERENCED H,L,C,UPDATED
                        ;A,B DESTROYED
                        ;IF NOT A VARIBLE CY=0
                        ;H,L,C ARE LEFT UNTOUCHED
014203  315 042 022     VAR:    CALL    ALPHA           ;1ST CHAR A LETTER?
014206  320                     RNC                     ;NO-NOT VAR.
014207  043                     INX     H               ;BUMP PNTR'S
014210  015                     DCR     C
```

```
014211  302 225 030          JNZ   MORE      ;MORE TO LINE
014214  305          SC1:    PUSH  B         ;SAVE B,EOL
014215  016 000              MVI   C,0       ;SET FOR CALL TO FSYM
014217  053                  DCX   H         ;GET SINGLE LETTER
014220  106                  MOV   B,M       ;VAR TO B
014221  043                  INX   H
014222  303 307 030          JMP   SCALR
014225  315 042 022  MORE:   CALL  ALPHA     ;2ND A LETTER?
014230  322 252 030          JNC   SFSG      ;SO FAR SO GOOD
014233  305                  PUSH  B         ;SAVE C
014234  076 002              MVI   A,2       ;CHECK FOR DELIMITER
014236  315 067 022          CALL  SYMSRT
014241  301                  POP   B         ;RESTORE C
014242  074                  INR   A         ;FOUND?
014243  302 214 030          JNZ   SC1       ;YES
014246  014          BUPT:   INR   C         ;NOT A VAR.
014247  053                  DCX   H         ;BACK UP PNTR'S
014250  267                  ORA   A         ;CY=0 AND RET
014251  311                  RET
014252  315 024 022  SFSG:   CALL  NUMB      ;TEST FOR NUMBER
014255  322 317 030          JNC   ARCK      ;MAYBE AN ARRAY
014260  043                  INX   H         ;ITS A SCALAR
014261  015                  DCR   C         ;BUMP PNTR'S
014262  312 300 030          JZ    SLOAD     ;EOL
014265  305                  PUSH  B         ;SAVE C
014266  076 002              MVI   A,2       ;SET UP FOR SYMSRT
014270  315 067 022          CALL  SYMSRT    ;TEST FOR LEGAL
014273  301                  POP   B         ;GET C BACK
014274  074                  INR   A         ;DELIMITER FOUND?
014275  312 070 032          JZ    ER8       ;NO, ERROR
014300  053          SLOAD:  DCX   H         ;MOVE BACK,
014301  305                  PUSH  B         ;SAVE C,
014302  116                  MOV   C,M       ;GET VAR. INTO
014303  053                  DCX   H         ;B,C FOR FSYM
014304  106                  MOV   B,M
014305  043                  INX   H
014306  043                  INX   H
014307  353          SCALR:  XCHG            ;SAVE H,L IN D,E
014310  315 015 026          CALL  FSYM      ;GET PNTR TO VALUE
014313  353                  XCHG            ;RESTORE H,L PNTR TO DE
014314  301                  POP   B         ;GET C REG BACK
014315  067                  STC             ;SET CY,RET
014316  311                  RET
014317  176          ARCK:   MOV   A,M       ;ARRAY CHEK, GET CHARACTER
014320  376 250              CPI   250Q      ;IS IT (?
014322  312 343 030          JZ    ARYES     ;YES,ITS AN ARRAY
014325  076 002              MVI   A,2       ;NO-CHEK FOR LEGAL DELIM.
014327  305                  PUSH  B         ;SAVE C
014330  315 367 022          CALL  SYMSRT
014333  301                  POP   B         ;RESTORE C
014334  074                  INR   A         ;DELIMITER FOUND?
014335  312 070 032          JZ    ER8
014340  303 214 030          JMP   SC1       ;1 CHAR. SCALAR VAR.
014343  053          ARYES:  DCX   H         ;YES-WE HAVE ARRAY
014344  176                  MOV   A,M       ;GET VAR.
014345  043                  INX   H
014346  365                  PUSH  PSW       ;SAVE VAR.
014347  315 373 025          CALL  ICP8      ;BUMP PNTR'S
014352  315 335 031          CALL  EVAL      ;EVALUATE SUBSCRIPT
014355  345                  PUSH  H         ;SAVE REG H,L
014356  041 174 060          LXI   H,FREG1
014361  315 271 027          CALL  COPDH     ;COPY IT
014364  353                  XCHG
014365  341                  POP   H         ;RESTORE H,L
014366  315 053 031          CALL  FIX       ;FIX VALUE
014371  076 251              MVI   A,251Q    ;CHECK FOR )
014373  276                  CMP   M
014374  302 070 032          JNZ   ER8
014377  043                  INX   H
014400  015                  DCR   C         ;BUMP PNTR'S
014401  023                  INX   D         ;PNT TO LOWER 2 BYTES
014402  023                  INX   D
014403  032                  LDAX  D
014404  107                  MOV   B,A       ;H-BYTE TO B
014405  023                  INX   D         ;PNT TO LOW BYTE
014406  032                  LDAX  D         ;LOW BYTE TO A
014407  267                  ORA   A         ;KILL CY
014410  027                  RAL             ;START MULT OF OFFSET
014411  137                  MOV   E,A       ;BY 4(BYTES/FLTPT =)
014412  170                  MOV   A,B       ;GET H BYTE
014413  027                  RAL
014414  127                  MOV   D,A       ;DE IS OFFSET*2
014415  173                  MOV   A,E       ;GET LOW
014416  267                  ORA   A         ;KILL CARRY
014417  027                  RAL
014420  137                  MOV   E,A
014421  172                  MOV   A,D
014422  027                  RAL
014423  127                  MOV   D,A
014424  361                  POP   PSW       ;DE CONTAIN OFFSET*4
014425  305                  PUSH  B         ;GET VAR., SAVE C
014426  117                  MOV   C,A
014427  006 000              MVI   B,0       ;SETUP TO CALL FSYM
```

```
014431  345                     PUSH    H               ;SAVE H,L
014432  315 015 026             CALL    FSYM            ;GET START ADD.
014435  332 045 031             JC      AFOND
014440  076 022                 MVI     A,12H           ;ERROR 12
014442  303 276 025             JMP     ERROR           ;ARRAY REF. NOT DIM'ED.
014445  031             AFOND:  DAD     D               ;H,L NOW PNT TO START OF
014446  353                     XCHG                    ;ARRAY, ADD OFFSET, EXCHG
014447  341                     POP     H               ;RESTORE PNTR'S AND RET.
014450  301                     POP     B
014451  067                     STC                     ;SET CY
014452  311                     RET
                        ;ROUTINE TO FIX FLOATING POINT
                        ;NUMBERS, ALL REG'S BUT A ARE
                        ;MAINTAINED. DE PNT TO 4 BYTES
                        ;OF = TO BE FIXED
014453  305             FIX:    PUSH    B
014454  345                     PUSH    H
014455  325                     PUSH    D               ;SAVE REG'S
014456  023                     INX     D
014457  023                     INX     D
014460  023                     INX     D               ;PNT TO 4TH BYTE
014461  032                     LDAX    D
014462  365                     PUSH    PSW             ;SAVE CHAR. (FOR SIGN)
014463  346 177                 ANI     177Q
014465  027                     RAL                     ;CHEK IF EXP SIGN IS -
014466  027                     RAL
014467  332 106 031             JC      MINSE
014472  037                     RAR
014473  037                     RAR                     ;RESTORE CHAR
014474  376 030                 CPI     30Q             ;IS IT TOO BIG?
014476  332 110 031             JC      GOOD
014501  076 023                 MVI     A,13H           ;ERROR 13
014503  303 276 025             JMP     ERROR           ;FIX = TOO BIG
014506  037             MINSE:  RAR
014507  037                     RAR
014510  022             GOOD:   STAX    D               ;ABSOLUTE VALUE
014511  033                     DCX     D
014512  033                     DCX     D
014513  033                     DCX     D               ;MOV PNTR BACK
014514  041 174 060             LXI     H,FREG1
014517  315 271 027             CALL    COPDH           ;COPY TO FREG1
014522  041 200 060             LXI     H,FREG2         ;STORE .5*2**24 IN
014525  021 205 031             LXI     D,FDAT          ;FREG2
014530  315 271 027             CALL    COPDH           ;COPY IT
014533  041 174 060             LXI     H,FREG1         ;SET UP TO CALL LADD
014536  006 200                 MVI     B,FREG2 AND 377Q
014540  016 146                 MVI     C,SCR AND 377Q
014542  315 320 017             CALL    LADD            ;ADD THEM,RESULT IN FREG1
014545  041 174 060             LXI     H,FREG1
014550  361                     POP     PSW             ;GET SIGN AND ADD.
014551  321                     POP     D
014552  027                     RAL
014553  076 000                 MVI     A,0             ;GET SIGN ONLY
014555  037                     RAR
014556  106                     MOV     B,M             ;GET BYTE1
014557  022                     STAX    D               ;STORE BYTE 1 OF FIX
014560  170                     MOV     A,B
014561  346 177                 ANI     177Q            ;CLEAR HIGH BIT (FROM ADD)
014563  023                     INX     D
014564  043                     INX     H
014565  106                     MOV     B,M             ;GET BYTE 2
014566  022                     STAX    D               ;STORE BYTE 2 OF FIX
014567  023                     INX     D
014570  170                     MOV     A,B
014571  043                     INX     H
014572  106                     MOV     B,M             ;GET BYTE 3
014573  022                     STAX    D               ;STORE BYTE 3 OF FIX
014574  170                     MOV     A,B
014575  023                     INX     D
014576  022                     STAX    D               ;STORE BYTE 4 OF FIX
014577  033                     DCX     D               ;FIX D PNTR
014600  033                     DCX     D
014601  033                     DCX     D
014602  341                     POP     H
014603  301                     POP     B
014604  311                     RET
014605  200 000 000 030 FDAT:   DB      200Q,0,0,30Q
                        ;INP SAVES ALL REG'S
                        ;SERVES AS BUFFER BETWEEN FINPT AND
                        ;DATA INPUT. IF MODE=0, DATA COMES FROM TTY
                        ;IF MODE=1 DATA COMES FROM SOURCE STMTS.
                        ;IN ALL CASES HL,C ARE UPDATED FROM HLINP, AND
                        ;CREG AND RETURNED TO THOSE LOCATIONS
014611  345             INP:    PUSH    H               ;SAVE ALL REG'S
014612  325                     PUSH    D
014613  305                     PUSH    B
014614  052 206 060             LHLD    HLINP           ;GET PNTR'S
014617  072 204 060             LDA     CREG
014622  117                     MOV     C,A
014623  267                     ORA     A               ;CHECK FOR EOL
014624  302 235 031             JNZ     CHKMD           ;NO CHECK MODE
014627  076 240         SPACE:  MVI     A,240Q          ;SEND A SPACE
014631  301             IDONE:  POP     B               ;RESTORE REG'S
```

```
014632  321                     POP     D
014633  341                     POP     H
014634  311                     RET                     ;AND RETURN
014635  072 205 060     CHKMD:  LDA     MODE            ;GET MODE
014640  075                     DCR     A               ;CHECK IT
014641  312 255 031             JZ      MODE1           ;MODE IS 1
014644  176                     MOV     A,M             ;MODE 0, GET CHAR.
014645  376 254                 CPI     ',' OR 200Q     ;IS IT A ','?
014647  312 227 031             JZ      SPACE           ;YES - SEND A SPACE
014652  303 321 031             JMP     BMPTR           ;NO - SEND IT
014655  315 024 022     MODE1:  CALL    NUMB            ;NUMBER? (ALSO LOADS IT TO A)
014660  332 321 031             JC      BMPTR           ;YES - SEND IT AND BUMP PNTR'S
014663  376 256                 CPI     256Q            ;DEC. PNT.?
014665  312 321 031             JZ      BMPTR
014670  376 305                 CPI     305Q            ;E?
014672  312 321 031             JZ      BMPTR
014675  376 253                 CPI     253Q            ;+?
014677  312 307 031             JZ      CHEKE
014702  376 255                 CPI     255Q            ;-?
014704  302 227 031             JNZ     SPACE           ;SEND A SPACE
014707  107             CHEKE:  MOV     B,A             ;CHEK IF E PRECEDES +,-
014710  053                     DCX     H               ;BACK UP AND GET PRE-
014711  176                     MOV     A,M             ;CEDING CHARACTER
014712  376 305                 CPI     305Q            ;IS IT E?
014714  302 227 031             JNZ     SPACE           ;NO, +OR- WAS DELIMITTER
014717  170                     MOV     A,B             ;YES,GET + OR -
014720  043                     INX     H               ;RESTORE H,L
014721  043             BMPTR:  INX     H               ;BUMP AND STORE PNTR'S
014722  015                     DCR     C
014723  042 206 060             SHLD    HLINP
014726  041 204 060             LXI     H,CREG
014731  161                     MOV     M,C
014732  303 231 031             JMP     IDONE           ;RESTORE REG'S AND RETURN
                        ;THIS ROUTINE WILL EVALUATE UNARY AND/OR
                        ;BINARY EXPRESIONS CALLED WITH H AND L
                        ;POINTING TO FIRST CHAR. OF EXP., C CONTAINS
                        ;NUMBER OF CHAR'S LEFT IN LINE. RETURNS
                        ;D(H) AND E(L) POINTING TO THE ANSWER
                        ;THIS ROUTINE CALLS ITSELF RECURSIVELY
                        ;IN ORDER TO EVALUATE SUBSCRIPT
                        ;EXPRESIONS.  REG A,B DESTROYED
                        ;C,H,L ARE UPDATED
014735  076 255         EVAL:   MVI     A,255Q          ;IS IT UNARY -
014737  276                     CMP     M               ;Z=1 => YES
014740  365                     PUSH    PSW             ;Z=0 => NO
014741  302 347 031             JNZ     ECAV
014744  315 373 025             CALL    ICP8            ;BUMP POINTER
014747  315 324 027     ECAV:   CALL    VALUE           ;GET PNTR. TO VALUE
014752  345                     PUSH    H               ;GET VALUE TO FREG1
014753  041 174 060             LXI     H,FREG1
014756  315 271 027             CALL    COPDH
014761  353                     XCHG
014762  341                     POP     H
014763  361                     POP     PSW             ;GET SIGN
014764  302 002 032             JNZ     DOL             ;SHALL WE NEGATE?
014767  023                     INX     D               ;YES, POINT TO CHAR.
014770  023                     INX     D
014771  023                     INX     D
014772  032                     LDAX    D               ;AND LOAD TO A
014773  027                     RAL                     ;ROTATE SIGN TO CY
014774  077                     CMC                     ;COMPLEMENT IT
014775  037                     RAR                     ;ROTATE BACK
014776  022                     STAX    D               ;STORE AWAY
014777  033                     DCX     D               ;AND FIX PNTR.
015000  033                     DCX     D
015001  033                     DCX     D
015002  171             DOL:    MOV     A,C             ;IS THIS END OF LINE?
015003  267                     ORA     A
015004  310                     RZ                      ;YES-RETURN
015005  305                     PUSH    B               ;SAVE C
015006  076 002                 MVI     A,2             ;NO SET UP TO CALL
015010  315 367 022             CALL    SYMSRT          ;SYMSRT AND CALL
015013  301                     POP     B               ;RESTORE C
015014  074                     INR     A               ;DELIMITER FOUND?
015015  312 070 032             JZ      ER8             ;NO, ERROR
015020  326 012         EOK:    SUI     10              ;CHECK FOR EXPRESSION
015022  330                     RC                      ;DELIMITER
015023  365                     PUSH    PSW             ;SAVE OVERATION
015024  315 373 025             CALL    ICP8            ;BUMP PNTR'S
015027  267                     ORA     A               ;CLEAR CY
015030  345             AGA:    PUSH    H               ;GET BYTES OF NUMBER
015031  032                     LDAX    D               ;AND PLACE ON STACK
015032  157                     MOV     L,A
015033  023                     INX     D
015034  032                     LDAX    D
015035  023                     INX     D
015036  147                     MOV     H,A             ;2 BYTES TO H,L
015037  343                     XTHL                    ;XCHANGE, RESTORES H,L
015040  077                     CMC
015041  332 030 032             JC      AGA             ;ANOTHER PASS?
015044  315 324 027             CALL    VALUE           ;GET 2ND VALUE
015047  171                     MOV     A,C             ;CHECK FOR END OF LINE
015050  267                     ORA     A               ;IF SO => WELL FORMED
```

```
015051  312 075 032             JZ      WFOR
015054  305                     PUSH    B               ;SAVE C
015055  076 002                 MVI     A,2             ;ELSE CALL SYMSRT TO
015057  315 367 022             CALL    SYMSRT          ;CHEK FOR EXP. DEL.
015062  301                     POP     B               ;RECOVER IT
015063  376 012                 CPI     10
015065  332 075 032             JC      WFOR            ;YES, WELL FORMED
015070  076 010         ER8:    MVI     A,8             ;ILL-FORMED EXP.
015072  303 276 025             JMP     ERROR
015075  305             WFOR:   PUSH    B               ;SAVE C, AND H,L
015076  345                     PUSH    H
015077  041 200 060             LXI     H,FREG2         ;COPY 2ND VALUE TO
015102  315 271 027             CALL    COPDH           ;FREG2
015105  321                     POP     D               ;GET BYTES FROM STACK
015106  301                     POP     B
015107  341                     POP     H               ;INTO FREG1+2
015110  042 176 060             SHLD    FREG1+2
015113  341                     POP     H               ;AND NEXT 2 BYTES
015114  042 174 060             SHLD    FREG1           ;FROM STACK TO FREG1
015117  353                     XCHG
015120  361                     POP     PSW             ;GET OPERATION
                        ;THIS ROUTINE PERFORMS BINARY OPERATIONS ON OPERANDS IN FREG1 AND
                        ;B,C,H,L ARE LEFT UNDISTURBED. A IS DESTROYED                FREG2
                        ;D,E PNT TO RESULT
                        ;OPERATIONS ARE SPECIFIED BY A REGISTER AS FOLLOWS:
                        ;
                        ;       A=0     =>              FREG1 * FREG2
                        ;       A=1     =>              FREG1 / FREG2
                        ;       A=2     =>              FREG1 + FREG2
                        ;       A=3     =>              FREG1 - FREG2
                        ;
                        ;IN CASE OF ARITHMETIC ERROR A MESSAGE IS SENT TO USER.
                        ;IF A CONTAINS ILLEGAL OPERATION REQUEST ERROR IS SENT TO USER
                        ;(ERROR 8) AND THE INTERPRETER IS ABORTED.
015121  305             BINOP:  PUSH    B               ;SAVE REG'S
015122  345                     PUSH    H
015123  041 174 060             LXI     H,FREG1         ;SET UP PNTR'S TO
015126  006 200                 MVI     B,FREG2 AND 377Q ;FREG'S AND SCR AREA
015130  016 146                 MVI     C,SCR AND 377Q  ;AND DO OPERATION
015132  075                     DCR     A
015133  372 235 032             JM      FMULT           ;0,1=>* OR /
015136  312 243 032             JZ      DIV             ;2,3=>+ OR -
015141  075                     DCR     A
015142  312 154 032             JZ      ADDD
015145  075                     DCR     A
015146  312 227 032             JZ      SUBB
015151  303 070 032             JMP     ER8             ;ILLEGAL OPER.
015154  315 320 017     ADDD:   CALL    LADD            ;DO ADDITION
015157  124             ASBC:   MOV     D,H             ;FIX PNTR'S FOR RET.
015160  135                     MOV     E,L
015161  267             FPERR:  ORA     A               ;SET FLAGS
015162  312 224 032             JZ      NFPER           ;NO ERROR
015165  325                     PUSH    D               ;SAVE DE
015166  365                     PUSH    PSW             ;SAVE A
015167  315 320 022             CALL    WRIT            ;DUMP BUFFER
015172  361                     POP     PSW             ;GET A BACK
015173  041 215 032             LXI     H,WFPER         ;RETURN ADDRESS
015176  345                     PUSH    H               ;SAVE ON STACK
015177  041 132 024             LXI     H,ODATA         ;MESSAGE TABLE
015202  027                     RAL                     ;UNDERFLOW?
015203  332 077 024             JC      FOR12           ;YES
015206  027                     RAL                     ;OVERFLOW?
015207  332 100 024             JC      FOR11           ;YES
015212  303 101 024             JMP     FOR10           ;NO - ITS ZERODIVIDE
015215  041 132 024     WFPER:  LXI     H,ODATA         ;MESSAGE TABLE
015220  315 337 025             CALL    ERLN            ;PRINT 'IN LINE --' (USE PART OF
015223  321                     POP     D               ;RESTORE REG'S              ERROR
015224  341             NFPER:  POP     H
015225  301                     POP     B
015226  311                     RET
015227  315 331 017     SUBB:   CALL    LSUB            ;DO SUBTRACTION
015232  303 157 032             JMP     ASBC
015235  315 323 017     FMULT:  CALL    LMUL            ;DO MULT.
015240  303 246 032             JMP     MDBC
015243  315 326 017     DIV:    CALL    LDIV            ;DO DIV.
015246  124             MDBC:   MOV     D,H             ;AND FIX PNTR'S FOR RET.
015247  131                     MOV     E,C
015250  303 161 032             JMP     FPERR           ;CHECK FOR ERROR
                        ;PRINT PROCESSOR
015253  052 133 060     PRI:    LHLD    CPNT
015256  043                     INX     H               ;INCR. PAST KEYWORD
015257  043                     INX     H
015260  043                     INX     H
015261  315 366 025             CALL    ICP7
015264  043                     INX     H               ;BUMP PNTRS
015265  015                     DCR     C
015266  006 000                 MVI     B,0             ;SET CHAR CNT
015270  302 304 032             JNZ     PLOOP           ;CONTINUE IF MORE
015273  004                     INR     B               ;NOTHING MORE, PAD A NULL
015274  076 000                 MVI     A,0
015276  315 255 022             CALL    PAD
015301  303 134 033             JMP     PEND            ;WRITE IT AND CONTINUE
015304  176             PLOOP:  MOV     A,M             ;GET CHARACTER
015305  376 242                 CPI     '"'+200Q        ;IS IT "?
```

```
015307  302 354 032          JNZ   EXPRE             ;NO
015312  315 366 025  QUOTE:   CALL  ICP7              ;GET CHARACTER TO A
015315  176                   MOV   A,M
015316  376 242               CPI   '"'+200Q          ;IS IT "?
015320  312 336 032          JZ    QCHEK
015323  004            QOTOK: INR   B                 ;INCREMENT CNT
015324  120                   MOV   D,B               ;SAVE IN D
015325  006 001               MVI   B,1               ;PAD ONCE
015327  315 255 022          CALL  PAD
015332  102                   MOV   B,D               ;RESTORE CNT
015333  303 312 032          JMP   QUOTE             ;AGAIN
015336  043            QCHEK: INX   H                 ;BUMP PNTRS
015337  015                   DCR   C
015340  312 134 033          JZ    PEND              ;EOL
015343  176                   MOV   A,M
015344  376 242               CPI   '"'+200Q          ;ANOTHER "?
015346  312 323 032          JZ    QOTOK
015351  303 035 033          JMP   SCOLN
015354  315 042 022  EXPRE:   CALL  ALPHA             ;IS IT A LETTER
015357  332 003 033          JC    PRTIT             ;YES, EVALUATE AND PRINT
015362  315 024 022          CALL  NUMB              ;IS IT A NUMB?
015365  332 003 033          JC    PRTIT             ;YES, EVALUATE AND PRINT
015370  176                   MOV   A,M
015371  376 256               CPI   '.'+200Q          ;IS IT A DECIMAL PNT?
015373  312 003 033          JZ    PRTIT             ;YES EVALUATE, PRINT
015376  376 255               CPI   '-'+200Q          ;IS IT A -?
015400  302 035 033          JNZ   SCOLN             ;NO, CHECK FOR ;
015403  305            PRTIT: PUSH  B                 ;SAVE CNT
015404  315 335 031          CALL  EVAL              ;EVALUATE EXPRESION
015407  305                   PUSH  B                 ;SAVE C,H,L
015410  345                   PUSH  H
015411  353                   XCHG                    ;DE TO HL
015412  016 146               MVI   C,SCR AND 377Q    ;SET UP, CONVERT
015414  315 345 017          CALL  CONV
015417  341                   POP   H                 ;RESTORE REG'S
015420  301                   POP   B
015421  171                   MOV   A,C
015422  301                   POP   B
015423  117                   MOV   C,A
015424  267                   ORA   A                 ;CHECK EOL
015425  312 134 033          JZ    PEND
015430  076 013               MVI   A,11              ;UPDATE CNTR
015432  200                   ADD   B
015433  107                   MOV   B,A
015434  176                   MOV   A,M               ;GET CHAR.
015435  376 273        SCOLN: CPI   ';'+200Q          ;IS IT ;?
015437  312 117 033          JZ    SONWD             ;YES
015442  376 254               CPI   ','+200Q          ;IS IT ,?
015444  302 072 027          JNZ   ER6               ;NO-UNEXPECTED CHAR.
015447  257                   XRA   A                 ;ZERO A
015450  306 015        ADFLD: ADI   13                ;ADD FIELD LENGTH
015452  270                   CMP   B                 ;COMPARE TO CNT
015453  312 061 033          JZ    $+6
015456  322 103 033          JNC   FLDFD
015461  376 064               CPI   52                ;LAST FLD?
015463  302 050 033          JNZ   ADFLD
015466  315 220 022          CALL  WRIT              ;YES-WRITE LINE
015471  006 000               MVI   B,0               ;RESET CNT
015473  043            ONWD:  INX   H                 ;BUMP PNTRS
015474  015                   DCR   C
015475  312 134 033          JZ    PEND
015500  303 304 032          JMP   PLOOP
015503  220            FLDFD: SUB   B                 ;FOUND FIELD
015504  120                   MOV   D,B               ;DETERMIN   OF SPACES TO PAD
015505  137                   MOV   E,A               ;SET UP TO CALL PAD
015506  107                   MOV   B,A
015507  076 240               MVI   A,240Q
015511  315 255 022          CALL  PAD               ;PAD SPACES
015514  172                   MOV   A,D
015515  203                   ADD   E                 ;NEW CNT
015516  107                   MOV   B,A               ;SAVE IN B
015517  043            SONWD: INX   H                 ;CHECK EOL
015520  015                   DCR   C
015521  302 304 032          JNZ   PLOOP
015524  026 001               MVI   D,1               ;SUPPRESS CR/LF
015526  315 322 022          CALL  WRIT1
015531  303 137 033          JMP   $+6
015534  315 320 022  PEND:    CALL  WRIT              ;DUMP BUFFER, CONTINUE
015537  303 242 033          JMP   IEND
                       ;INPUT PROCESSOR - READS VALUES FROM TTY
                       ;THEY MUST BE DELIMITED BY COMMAS ONLY
015542  171          INPUT:    MOV   A,C             ;IN CASE OF ERROR
015543  062 137 060           STA   PL6               ;SAVE
015546  052 133 060   INPER:   LHLD  CPNT            ;INPUT LINE (V-STRING) PNTR
015551  043                    INX   H                ;ADJUST PNTR'S
015552  043                    INX   H
015553  043                    INX   H
015554  315 366 025           CALL  ICP7
015557  315 366 025           CALL  ICP7
015562  305            PRMPT:  PUSH  B                ;SAVE PNTR'S
015563  345                    PUSH  H
015564  006 001                MVI   B,1              ;SEND PROMPT
015565  076 072                MVI   A,':'
```

```
015570  120                     MOV     D,B             ;TO SUPPRESS CR/LF
015571  315 255 022             CALL    PAD             ;PAD IT
015574  315 322 022             CALL    WRIT1           ;WRITE IT
015577  041 001 060             LXI     H,IBUF          ;ADD. OF INPUT BUFFER
015602  315 311 023             CALL    TTYIN           ;READ A LINE
015605  353                     XCHG                    ;ADD. OF K-STRING TO DE
015606  341                     POP     H               ;ADD. OF V-STRING
015607  301                     POP     B               ;V-STRING CNT TO C
015610  107                     MOV     B,A             ;K-STRING CNT TO B
015611  315 250 033             CALL    STRIN           ;TRANSFER CONSTANTS TO VARIBLES
015614  312 237 033             JZ      INPOK           ;NO ERROR
015617  041 132 024             LXI     H,ODATA         ;SEND ERROR MESSAGE
015622  315 102 024             CALL    FORM9
015625  315 320 022             CALL    WRIT
015630  072 137 060             LDA     PL6             ;GET V-STRING CNT
015633  117                     MOV     C,A
015634  303 146 033             JMP     INPER           ;START AGAIN
015637  332 162 033     INPOK:  JC      PRMPT           ;NEED MORE CONSTANTS
015642  052 126 060     IEND:   LHLD    KFPNT           ;ALL OK - GET NEW PNTR.
015645  303 232 025             JMP     ILOOP           ;CONTINUE
                        ;THIS ROUTINE TRANSFERS THE FLOATING POINT VALUES
                        ;OF AN ASCII STRING OF CONSTANTS TO THE LOCATIONS
                        ;SPECIFIED BY AN ASCII STRING OF VARIBLES
                        ;POINTER AND LINE CNT OF VAR. STRING ARE IN HL,C
                        ;POINTER AND LINE CNT OF CONST. STRING ARE IN DE,B
                        ;ON RETURN:
                        ;           Z=0 AND CY=0    ALL OK
                        ;           Z=0 AND CY=1    NEED MORE CONSTANTS
                        ;           Z=1             ERROR IN CONVERSION
                        ;ALL POINTERS AND LINE CNT'S ARE RETURNED UPDATED
015650  171             STRIN:  MOV     A,C             ;GET V-STRING CNT
015651  267                     ORA     A               ;TEST FOR EOL
015652  310                     RZ                      ;DONE, CY=0 => ALL OK
015653  176                     MOV     A,M             ;GET CHAR.
015654  376 254                 CPI     ',' OR 200Q     ;IS IT A ,?
015656  302 266 033             JNZ     STOKV           ;IT'S NOT A ,
015661  043                     INX     H               ;COMMA, BUMP PNTR'S
015662  015                     DCR     C
015663  312 336 033             JZ      ERRET           ;POSSIBLE ERROR (IF EOL)
015666  170             STOKV:  MOV     A,B             ;GET K-STRING LENGTH
015667  267                     ORA     A               ;TEST FOR EOL
015670  067                     STC                     ;IN CASE IT'S EOL
015671  310                     RZ                      ;RET, CY=1 =@ NEED MORE CONSTANTS
015672  032                     LDAX    D               ;GET CHAR
015673  376 254                 CPI     ',' OR 200Q     ;TEST FOR ,
015675  302 305 033             JNZ     STOKK           ;NOT A , - READY TO GO
015700  023                     INX     D               ;BUMP PNTR'S
015701  005                     DCR     B
015702  312 336 033             JZ      ERRET           ;POSSIBLE ERROR (IF EOL)
015705  305             STOKK:  PUSH    B               ;SAVE,K-STRING CNT
015706  325                     PUSH    D               ;SAVE K-STRING PNTR
015707  315 203 030             CALL    VAR             ;ADD. TO VARIBLE TO DE
015712  353                     XCHG                    ;VAR. ADD TO H,L
015713  042 212 060             SHLD    VARAD           ;SAVE
015716  341                     POP     H               ;ADDRESS OF K-STRING
015717  171                     MOV     A,C             ;V-STRING CNT TO A
015720  301                     POP     B               ;K-STRING CNT TO B
015721  110                     MOV     C,B             ;K-STRING CNT TO C
015722  365                     PUSH    PSW             ;SAVE V-STRING CNT
015723  325                     PUSH    D               ;SAVE V-STRING ADD.
015724  076 000                 MVI     A,0             ;A=0 =@ DATA FROM TTY
015726  315 144 030             CALL    RDKON           ;GET CONSTANT TO GREG
015731  322 341 033             JNC     STNER
015734  341                     POP     H               ;EMPTY STACK
015735  341                     POP     H
015736  257             ERRET:  XRA     A               ;ERROR
015737  074                     INR     A
015740  311                     RET
015741  345             STNER:  PUSH    H               ;SAVE K-STRING PNTR.
015742  052 212 060             LHLD    VARAD           ;GET VAR. ADD
015745  021 167 060             LXI     D,GREG          ;ADD. TO CONST.
015750  315 271 027             CALL    COPDH           ;COPY IT TO VARIABLE LOC.
015753  321                     POP     D               ;K-STING PNTR. TO DE
015754  101                     MOV     B,C             ;K-STRING LENGTH TO B
015755  341                     POP     H               ;V-STRING PNTR. TO HL
015756  361                     POP     PSW             ;V-STRING LENGTH TO C
015757  117                     MOV     C,A
015760  303 250 033             JMP     STRIN           ;LOOP
                        ;LET STMT. PROCESSOR
015763  052 133 060     LET:    LHLD    CPNT            ;GET PNTR.
015766  043                     INX     H               ;FIX PNTR.
015767  043                     INX     H
015770  043                     INX     H
015771  171                     MOV     A,C             ;CHECK FOR EOL
015772  267                     ORA     A
015773  302 003 034             JNZ     LOK
015776  076 007             ER7:    MVI     A,7
016000  303 276 025             JMP     ERROR
016003  315 203 030     LOK:    CALL    VAR             ;GET ADDRESS TO VAR.
016006  332 110 034             JC      SAVV            ;IT'S A VARIABLE
016011  076 003                 MVI     A,3             ;NO-CHEK FOR FUNC.
016013  315 367 022             CALL    SYMSRT
016016  376 377                 CPI     377Q
```

```
016020  312 070 032             JZ      ER8             ;DON'T KNOW WHAT IT IS
016023  075                     DCR     A
016024  302 216 034             JNZ     ER10            ;ILLEGAL USE OF FUNC.
016027  043                     INX     H               ;IT'S PUT,UPDATE H,L
016030  043                     INX     H
016031  043                     INX     H
016032  171                     MOV     A,C             ;EOL CHK
016033  207                     ORA     A
016034  312 070 032             JZ      ER8
016037  176                     MOV     A,M             ;CHEK FOR (
016040  376 250                 CPI     250Q
016042  302 070 022             JNZ     ER8
016045  315 373 025             CALL    ICP8            ;BUMP PNTRS
016050  315 335 031             CALL    EVAL            ;EVALUATE AND FIX
016053  345                     PUSH    H               ;SAVE H,L
016054  041 174 060             LXI     H,FREG1
016057  315 271 027             CALL    COPDH           ;COPY IT
016062  353                     XCHG
016063  341                     POP     H
016064  315 053 031             CALL    FIX
016067  023                     INX     D
016070  023                     INX     D
016071  023                     INX     D
016072  032                     LDAX    D               ;GET LOWEST BYTE
016073  365                     PUSH    PSW             ;PORT = IS SAVED
016074  176                     MOV     A,M
016075  376 251                 CPI     251Q            ;CHECK FOR )
016077  302 070 032             JNZ     ER8
016102  315 373 025             CALL    ICP8            ;BUMP PNTR'S
016105  026 377                 MVI     D,377Q
016107  132                     MOV     E,D
016110  325             SAVV:   PUSH    D               ;KEEP ADDRESS
016111  176                     MOV     A,M             ;CHEK FOR =
016112  376 275                 CPI     275Q
016114  302 070 032             JNZ     ER8
016117  315 373 025             CALL    ICP8            ;BUMP PNTRS
016122  315 335 031             CALL    EVAL            ;EVALUATE EXPRESSION
016125  341                     POP     H               ;GET ADDRESS
016126  315 243 022             CALL    CHK1
016131  332 142 034             JC      PTFIN           ;IT WAS A PUT
016134  315 271 027             CALL    COPDH           ;COPY TO ADDRESS
016137  303 242 033             JMP     IEND            ;CONTINUE
016142  041 174 060     PTFIN:  LXI     H,FREG1         ;COPY VALUE TO FREG1
016145  315 271 027             CALL    COPDH
016150  353                     XCHG
016151  315 053 031             CALL    FIX             ;FIX THE VALUE
016154  023                     INX     D
016155  023                     INX     D
016156  023                     INX     D
016157  032                     LDAX    D
016160  117                     MOV     C,A             ;SAVE IN C
016161  041 211 034             LXI     H,PINST         ;ADD OF BYTES TO GO TO
016164  021 167 060             LXI     D,GREG          ;RAM AT GREG
016167  006 005                 MVI     B,5             ;BYTE CNT
016171  176             PRI1:   MOV     A,M             ;STORE PROG. SEG. IN
016172  022                     STAX    D               ;RAM
016173  043                     INX     H
016174  023                     INX     D
016175  005                     DCR     B
016176  302 171 034             JNZ     PRI1
016201  361                     POP     PSW             ;GET PORT =
016202  041 170 060             LXI     H,GREG+1
016205  167                     MOV     M,A             ;STORE
016206  171                     MOV     A,C             ;GET DATA OUT TO A
016207  053                     DCX     H               ;TRANSFER
016210  351                     PCHL
016211  323 000         PINST:  OUT     0               ;RAM INSTRUCTIONS
016213  303 242 033             JMP     IEND
016216  076 020         ER10:   MVI     A,10H
016220  303 276 025             JMP     ERROR
                        ;IF STMT. PROCESSOR
016223  052 133 060     IFRT:   LHLD    CPNT            ;GET PNTR., ADJUST
016226  043                     INX     H
016227  014                     INR     C               ;CHECK EOL
016230  315 366 025             CALL    ICP7
016233  315 335 031             CALL    EVAL            ;EVALUATE EXPRESSION
016236  171                     MOV     A,C
016237  267                     ORA     A               ;CHECK EOL
016240  312 376 033             JZ      ER7
016243  345             IAGA:   PUSH    H               ;SAVE H,L, PUT VALUE ON STK
016244  032                     LDAX    D
016245  023                     INX     D
016246  157                     MOV     L,A
016247  032                     LDAX    D
016250  023                     INX     D
016251  147                     MOV     H,A
016252  343                     XTHL                    ;RESTORE H,L
016253  077                     CMC
016254  332 243 034             JC      IAGA            ;ANOTHER PASS?
016257  076 002                 MVI     A,2
016261  315 367 022             CALL    SYMSRT          ;CHEK TYPE OF RELATION
016264  376 004                 CPI     4               ;WAS IT LEGAL?
016266  332 276 034             JC      III
```

```
016271  076 024          ER14:   MVI     A,14H
016273  303 276 025              JMP     ERROR
016276  376 002          III:    CPI     2               ;WAS IT A ,?
016300  312 271 034              JZ      ER14
016303  074                      INR     A               ;ALL OK, INC,SAVE
016304  365                      PUSH    PSW
016305  014                      INR     C
016306  315 366 025              CALL    ICP7            ;BUMP PNTRS
016311  076 002                  MVI     A,2             ;CALL SYMSRT
016313  315 367 022              CALL    SYMSRT
016316  376 377                  CPI     377Q            ;FOUND ANYTHING?
016320  312 353 034              JZ      RELAT           ;DONE
016323  376 002                  CPI     2
016325  312 271 034              JZ      ER14            ;IT WAS A ,
016330  376 004                  CPI     4
016332  322 271 034              JNC     ER14            ;NOT LEGAL
016335  074                      INR     A
016336  107                      MOV     B,A
016337  014                      INR     C
016340  315 366 025              CALL    ICP7
016343  361                      POP     PSW             ;GET SECOND RELATION
016344  200                      ADD     B               ;ADD THEM
016345  365                      PUSH    PSW             ;AND SAVE
016346  376 010                  CPI     10Q             ;TEST FOR ==
016350  312 271 034              JZ      ER14
                        ;RELATION IS STORED ON TOP OF STACK (PUSH PSW) ACCORDING TO
                        ;THE FOLLOWING
                        ;
                        ;       1 =>    <
                        ;       2 =>    >
                        ;       3 =>    <>
                        ;       4 =>    =
                        ;       5 =>    <=
                        ;       6 =>    >=
                        ;
016353  315 335 031     RELAT:  CALL    EVAL            ;EVALUATE
016356  345                     PUSH    H               ;SAVE H,L
016357  041 200 060             LXI     H,FREG2         ;COPY TO FREG2
016362  315 271 027             CALL    COPDH
016365  341                     POP     H               ;GET H,L
016366  361                     POP     PSW             ;AND RELATION
016367  343                     XTHL                    ;GET 2ND 2 BYTES
016370  042 176 060             SHLD    FREG1+2         ;STORE
016373  341                     POP     H               ;GET 1ST 2 BYTES,STORE
016374  343                     XTHL
016375  042 174 060             SHLD    FREG1
016400  305                     PUSH    B
016401  365                     PUSH    PSW             ;SAVE A,B,C
016402  315 071 035             CALL    FCOMP           ;COMPARE NUMBERS
016405  127                     MOV     D,A             ;SAVE RESULT IN D
016406  361                     POP     PSW             ;GET RELATION,B,C
016407  301                     POP     B
016410  272                     CMP     D               ;SAME?
016411  312 054 035             JZ      TRUE            ;YES
016414  326 004                 SUI     4
016416  362 036 035             JP      NOT3            ;NOT RELATION 3
016421  074                     INR     A               ;IS IT RELATION 3?
016422  302 050 035             JNZ     FALSE           ;NO, ITS FALSE
016425  076 004                 MVI     A,4             ;IT IS, CHECK FOR INEQUALITY
016427  272                     CMP     D
016430  302 054 035             JNZ     TRUE
016433  303 050 035             JMP     FALSE
016436  272             NOT3:   CMP     D               ;RELATION 5,6 TRUE?
016437  312 054 035             JZ      TRUE            ;YES
016442  076 004                 MVI     A,4             ;IT WAS, CHECK FOR EQUALITY
016444  272                     CMP     D
016445  312 054 035             JZ      TRUE
016450  341             FALSE:  POP     H               ;CONTINUE
016451  303 242 033             JMP     IEND
016454  341             TRUE:   POP     H
016455  006 004                 MVI     B,4
016457  315 366 025     THEN:   CALL    ICP7            ;INCREMENT PAST THEN
016462  005                     DCR     B
016463  302 057 035             JNZ     THEN
016466  303 027 027             JMP     GTRA            ;TRANSFER TO GOTO
                        ;ROUTINE FCOMP COMPARES 2 FLOATING POINT #'S.  THEY ARE ASSUMED
                        ;TO BE IN FREG1 AND FREG2.
                        ;ALL REGISTERS ARE DESTROYED.
                        ;THE VALUE RETURNED IN REG A IS RESULT OF COMPARISON.
                        ;RESULTS ARE AS FOLLOWS:
                        ;
                        ;       A=1     =>      FREG1 < FREG2
                        ;       A=2     =>      FREG1 > FREG2
                        ;       A=4     =>      FREG1 = FREG2
                        ;
016471  041 177 060     FCOMP:  LXI     H,FREG1+3       ;PNTS TO CHAR OF 1ST
016474  021 203 060             LXI     D,FREG2+3       ;PNTS TO CHAR OF 2ND
016477  176                     MOV     A,M             ;GET  1 CHAR
016500  006 200                 MVI     B,200Q          ;MASK TO B
016502  240                     ANA     B               ;GET SIGN,  1
016503  117                     MOV     C,A             ;SAVE IN C
016504  032                     LDAX    D               ;GET CHAR  2
016505  240                     ANA     B               ;GET SIGN  2
```

```
016503  251                     XRA     C
016507  312 121 035             JZ      SINEQ           ;SAME SIGNS
016512  171                     MOV     A,C             ;OPPISITE SIGNS,GET  1 SIGN
016513  027                     RAL                     ;ROTATE TO CY
016514  076 001                 MVI     A,1
016516  330                     RC                      ;FREG1 < FREG2 => A=1
016517  074                     INR     A               ;ELSE FREG1 > FREG2
016520  311                     RET                     ;AND A=2
016521  305             SINEQ:  PUSH    B               ;SAVE SIGN
016522  053                     DCX     H               ;PNTR TO  1 IN H,L
016523  053                     DCX     H
016524  053                     DCX     H
016525  103                     MOV     B,E             ;PNTR TO  2 IN B
016526  005                     DCR     B
016527  005                     DCR     B
016530  005                     DCR     B
016531  315 337 017             CALL    LMCM            ;COMPARE MAGNITUDES
                ;AT THIS POINT Z=1 => =, CY=1 => 1<2
016534  301                     POP     B               ;GET SIGN BACK
016535  302 143 035             JNZ     $+6
016540  076 004                 MVI     A,4             ;EQUAL => A=4
016542  311                     RET
016543  171                     MOV     A,C             ;GET SIGN TO A
016544  074                     INR     A               ;SET SIGN BIT
016545  076 001                 MVI     A,1
016547  372 155 035             JM      $+6             ;SIGN IS NEGATIVE
016552  330                     RC                      ;SIGN=+ AND ABS(FREG1)<ABS(FREG2)
016553  074                     INR     A               ;ABS(FREG1)>ABS(FREG2)
016554  311                     RET
016555  320                     RNC                     ;SIGN=- AND ABS(FREG1)>ABS(FREG2)
016556  074                     INR     A               ;ABS(FREG1)<ABS(FREG2)
016557  311                     RET
                ;CALL PROCESSOR
016560  041 242 033     CALLP:  LXI     H,IEND          ;INIT RETURN ADDRESS
016563  345                     PUSH    H
016564  052 133 060             LHLD    CPNT            ;INIT POINTERS
016567  043                     INX     H
016570  043                     INX     H
016571  043                     INX     H
016572  315 366 025             CALL    ICP7
016575  176                     MOV     A,M             ;GET CHAR
016576  376 250                 CPI     '('+200Q        ;IS IT A (?
016600  302 376 033             JNZ     ER7             ;BAD
016603  315 366 025             CALL    ICP7            ;BUMP PNTRS
016606  315 052 022             CALL    CVB             ;GET SUB
016611  205                     ADD     L               ;UPDATA H,L
016612  157                     MOV     L,A
016613  076 000                 MVI     A,0
016615  214                     ADC     H
016616  147                     MOV     H,A             ;D NOW CONTAINS SUB
016617  345                     PUSH    H               ;SAVE HL
016620  041 322 037             LXI     H,SUBS          ;GET START OF SUB TABLE
016623  176             NUSUB:  MOV     A,M             ;GET ENTRY
016624  272                     CMP     D               ;COMPARE
016625  312 244 033             JZ      FNDSB           ;FOUND IT
016630  043                     INX     H               ;PNT TO NEXT ENTRY
016631  043                     INX     H
016632  043                     INX     H
016633  074                     INR     A               ;CHECK TO SEE IF LAST WAS 377Q
016634  302 223 035             JNZ     NUSUB
016637  076 025                 MVI     A,15H           ;ER 15 - NO SUB BY THIS #
016641  303 276 025             JMP     ERROR
016644  043             FNDSB:  INX     H               ;FOUND IT,GET STARTING ADD.
016645  136                     MOV     E,M
016646  043                     INX     H
016647  146                     MOV     H,M
016650  153                     MOV     L,E             ;AND SAVE IT
016651  042 135 060             SHLD    SBSAV
016654  052 131 060             LHLD    NXTSP           ;INIT MEMORY SCRATCH AREA
016657  042 210 060             SHLD    MESCR
016662  341                     POP     H               ;GET SOURCE PNTR BACK
016663  176             PARLP:  MOV     A,M             ;GET CHAR
016664  376 251                 CPI     ')'+200Q        ;IS IT )?
016666  312 342 035             JZ      CLSUB           ;YES - GO CALL SUB
016671  376 254                 CPI     ','+200Q        ;DO WE HAVE A ,?
016673  302 072 027             JNZ     ER6             ;UEXPECTED CHARACTER
016676  315 366 025             CALL    ICP7            ;BUMP PNTRS
016701  315 203 030             CALL    VAR             ;DO WE HAVE A VARIABLE
016704  322 313 035             JNC     PREXP           ;NO
016707  325                     PUSH    D               ;YES - SAVE ADDRESS
016710  303 263 035             JMP     PARLP           ;CONTINUE
016713  315 335 031     PREXP:  CALL    EVAL            ;EVALUATE EXPRESSION
016716  345                     PUSH    H               ;SAVE H,L
016717  052 210 060             LHLD    MESCR           ;GET SCRATCH AREA
016722  315 271 027             CALL    COPDH           ;AND COPY TO IT
016725  321                     POP     D               ;HL TO DE
016726  345                     PUSH    H               ;SAVE ADDRESS
016727  043                     INX     H               ;UPDATE MESCR
016730  043                     INX     H
016731  043                     INX     H
016732  043                     INX     H
016733  042 210 060             SHLD    MESCR           ;SAVE IT
016736  353                     XCHG                    ;GET H,L BACK
```

```
016737  303 263 035             JMP     PARLP           ;CONTINUE
016742  052 135 060     CLSUB:  LHLD    SBSAV           ;START OF ROUTINE
016745  351                     PCHL                    ;TRANSFER
                        ;GOSUB PROCESSOR
016746  041 232 026     GOSUB:  LXI     H,ILOOP         ;FOR RETURN STMT.
016751  345                     PUSH    H               ;TO STACK
016752  052 126 060             LHLD    KFPNT           ;PNTR. TO NEXT STMT.
016755  345                     PUSH    H               ;SAVE ON STACK
016756  052 131 060             LHLD    NXTSP           ;CHECK MEMORY
016761  315 024 025             CALL    MEMFUL
016764  052 133 060             LHLD    CPNT            ;GET CHAR. PNTR
016767  043                     INX     H
016770  303 021 027             JMP     GSENT           ;PART OF GOTO TO FINISH
                        ;RETURN STMT. PROCESSOR
016773  341             RETRN:  POP     H               ;GET RETURN ADD. FROM STACK
016774  311                     RET                     ;CONTINUE
                        ;FOR STATEMENT PROCESSOR
016775  052 133 060     FOR:    LHLD    CPNT            ;FIX PNTRS
017000  014                     INR     C
017001  043                     INX     H
017002  043                     INX     H
017003  315 366 025             CALL    ICP7
017006  315 042 022             CALL    ALPHA           ;LETTER?
017011  322 253 037             JNC     ER21            ;NO
017014  106                     MOV     B,M             ;GET IT TO B
017015  315 366 025             CALL    ICP7            ;BUMP PNTR'S
017020  121                     MOV     D,C             ;SAVE C
017021  016 000                 MVI     C,0             ;INIT C TO 0
017023  315 024 022             CALL    NUMB            ;NUMBER?
017026  322 037 035             JNC     $+9             ;NO
017031  116                     MOV     C,M             ;YES, GET IT
017032  043                     INX     H               ;BUMP PNTR'S
017033  025                     DCR     D
017034  312 376 033             JZ      ER7             ;PRENATURE EOL
017037  345                     PUSH    H               ;SAVE H,L
017040  315 015 026             CALL    FSYM            ;GET VAR. LOCATION
017043  343                     XTHL                    ;PUT ON STACK, GET H,L
017044  131                     MOV     E,C             ;VARIABLE TO D,E
017045  112                     MOV     C,D             ;RESTORE C
017046  120                     MOV     D,B
017047  353                     XCHG                    ;SAVE VAR NAME
017050  042 214 060             SHLD    VNAME
017053  353                     XCHG                    ;RESTORE H,L
017054  176                     MOV     A,M             ;LOOK FOR =
017055  376 275                 CPI     '=' OR 200Q
017057  302 222 037             JNZ     ER16
017062  315 366 025             CALL    ICP7            ;BUMP PNTR'S
017065  315 335 031             CALL    EVAL            ;EVALUATE EXPRESSION
017070  343                     XTHL                    ;VARIABLE LOCATION
017071  315 271 027             CALL    COPDH           ;WRITE VALUE
017074  042 216 060             SHLD    VLOC            ;SAVE PNTR TO VARIABLE LOCATION
017077  341                     POP     H               ;GET H,L BACK
017100  171                     MOV     A,C             ;CHECK EOL
017101  267                     ORA     A
017102  312 376 033             JZ      ER7
017105  076 002                 MVI     A,2             ;CHECK FOR 'TO'
017107  315 367 022             CALL    SYMSRT
017112  376 007                 CPI     7
017114  302 227 037             JNZ     ER17
017117  043                     INX     H               ;BUMB PNTR'S
017120  043                     INX     H
017121  171                     MOV     A,C             ;CHECK EOL
017122  267                     ORA     A
017123  312 376 033             JZ      ER7
017126  315 335 031             CALL    EVAL            ;EVALUATE LIMIT
017131  345                     PUSH    H               ;SAVE H,L
017132  041 220 060             LXI     H,FLIMT         ;SAVE LIMIT VALUE
017135  315 271 027             CALL    COPDH
017140  171                     MOV     A,C             ;CHECK EOL
017141  267                     ORA     A
017142  302 154 036             JNZ     STP
017145  021 353 036             LXI     D,FONE          ;DEFAULT STEP=1
017150  341                     POP     H               ;RESTORE H,L
017151  303 201 036             JMP     FBILD
017154  341             STP:    POP     H               ;GET H,L
017155  076 002                 MVI     A,2             ;LOOK FOR 'STEP'
017157  315 367 022             CALL    SYMSRT
017162  376 010                 CPI     8
017164  302 227 037             JNZ     ER17
017167  043                     INX     H               ;FIX H,L
017170  043                     INX     H
017171  043                     INX     H
017172  014                     INR     C               ;CHECK EOL
017173  315 366 025             CALL    ICP7
017176  315 335 031             CALL    EVAL            ;GET STEP SIZE
                        ;AT THIS POINT:
                        ;VARIABLE NAME IS IN LOCATION VNAME
                        ;VARIABLE ADDRESS IS IN LOCATION VLOC
                        ;VARIBLE HAS BEEN INITIALIZED
                        ;LIMIT IS IN 4 BYTE LOCATION FLIMT
                        ;STEP IS POINTED TO BY D,E
                        ;H,L,C ARE POINTER, COUNTER AS USUAL
017201  325             FBILD:  PUSH    D               ;SAVE PNTR TO STEP
```

```
017202  052 214 060             LHLD    VNAME           ;GET VARIABLE NAME
017205  076 077                 MVI     A,77Q           ;MASK
017207  244                     ANA     H               ;MASK OFF TOP 2 BITS
017210  107                     MOV     B,A             ;SET UP TO CALL FSYM
017211  115                     MOV     C,L
017212  315 015 026             CALL    FSYM            ;FIND ENTRY
017215  332 243 036             JC      FEXST           ;IT WAS THERE
017220  345                     PUSH    H               ;IT WASN'T, SAVE H,L
017221  052 131 060             LHLD    NXTSP           ;UPDATE NXTSP
017224  076 010                 MVI     A,8             ;ADD 8 TO H,L
017226  205                     ADD     L
017227  157                     MOV     L,A
017230  076 000                 MVI     A,0
017232  214                     ADC     H
017233  147                     MOV     H,A
017234  042 131 060             SHLD    NXTSP           ;NEW VALUE OF NXTSP
017237  315 024 025             CALL    MEMFUL          ;CHECK MEMORY
017242  341                     POP     H               ;GET ADD. IN DATA BLOCK
017243  321             FEXST:  POP     D               ;ADDRESS OF STEP SIZE
017244  315 271 027             CALL    COPDH           ;STORE IT
017247  043                     INX     H               ;PNT TO WHERE VAR. PNTR GOES
017250  043                     INX     H
017251  043                     INX     H
017252  043                     INX     H
017253  072 216 060             LDA     VLOC            ;FIRST BYTE
017256  167                     MOV     M,A             ;STORE IT
017257  043                     INX     H
017260  072 217 060             LDA     VLOC+1          ;SECOND BYTE
017263  167                     MOV     M,A
017264  043                     INX     H               ;PNT TO WHERE LIMIT GOES
017265  021 220 060             LXI     D,FLIMT         ;WHERE IT IS NOW
017270  315 271 027             CALL    COPDH           ;COPY IT
017273  043                     INX     H               ;PNT TO WHERE KFPNT GOES
017274  043                     INX     H
017275  043                     INX     H
017276  043                     INX     H
017277  072 126 060             LDA     KFPNT           ;1ST BYTE
017302  167                     MOV     M,A
017303  043                     INX     H
017304  072 127 060             LDA     KFPNT+1         ;2ND BYTE
017307  167                     MOV     M,A
                        ;PUT CURRENT VNAME ON NESTING STACK
017310  041 000 000             LXI     H,0             ;GET STACK-POINTER
017313  071                     DAD     SP
017314  042 216 060             SHLD    VLOC            ;SAVE IT
017317  052 224 060             LHLD    NEST            ;GET NEST SP
017322  175                     MOV     A,L             ;COMPARE WITH STACK LIMIT
017323  376 226                 CPI     TOPNS AND 377Q  ;NEED ONLY COMPARE PAGE LOCATION
017325  312 234 037             JZ      ER18            ;FOR'S NEXTED TOO DEEPLY
017330  371             NSTOK:  SPHL                    ;LOAD NEW SP
017331  353                     XCHG                    ;SAVE NEST SP
017332  052 214 060             LHLD    VNAME           ;GET INDEX NAME
017335  345                     PUSH    H               ;SAVE IT
017336  033                     DCX     D               ;UPDATE NEST SP
017337  033                     DCX     D
017340  353                     XCHG                    ;SAVE IT
017341  042 224 060             SHLD    NEST
017344  052 216 060             LHLD    VLOC            ;RESTORE OLD SP
017347  371                     SPHL
017350  303 242 033             JMP     IEND            ;ALL DONE
017353  200 000 000 001 FONE:    DB      2000,0,0,001Q   ;FLOATING PNT ONE
                        ;NEXT STATEMENT PROCESSOR
017357  052 133 060     NEXT:   LHLD    CPNT            ;FIX PNTR'S
017362  043                     INX     H
017363  043                     INX     H
017364  043                     INX     H
017365  014                     INR     C
017366  315 366 025             CALL    ICP7
017371  315 042 022             CALL    ALPHA           ;LETTER?
017374  322 253 037             JNC     ER21            ;NO, ERROR
017377  106                     MOV     B,M             ;YES, GET IT
017400  121                     MOV     D,C             ;SAVE C
017401  016 000                 MVI     C,0             ;INIT C TO 0
017403  043                     INX     H               ;BUMP PNTR'S
017404  025                     DCR     D
017405  312 023 037             JZ      NEXT1
017410  315 024 022             CALL    NUMB            ;NUMBER?
017413  322 253 037             JNC     ER21            ;NO, ERROR
017416  116                     MOV     C,M             ;YES, GET IT
017417  025                     DCR     D               ;SHOULD BE EOL
017420  302 253 037             JNZ     ER21
017423  041 000 000     NEXT1:  LXI     H,0             ;GET SP
017426  071                     DAD     SP
017427  042 216 060             SHLD    VLOC            ;SAVE IT
017432  052 224 060             LHLD    NEST            ;GET NEST SP
017435  175                     MOV     A,L             ;COMPARE WITH BOTTOM
017436  376 252                 CPI     BOTNS AND 377Q
017440  312 241 037             JZ      ER19            ;NEXT BEFORE FOR
017443  371                     SPHL                    ;LOAD SP
017444  341                     POP     H               ;GET LAST INDEX
017445  170                     MOV     A,B             ;COMPARE TO CURRENT
017446  274                     CMP     H
017447  302 246 037             JNZ     ER20            ;NESTING ERROR
```

```
017452  171                     MOV     A,C
017453  275                     CMP     L
017454  302 246 037             JNZ     ER20
017457  052 216 060             LHLD    VLOC            ;ALL OK, RESTORE OLD SP
017462  371                     SPHL
017463  076 077                 MVI     A,77Q           ;MASK
017465  240                     ANA     B               ;MASK OUT TOP 2 BITS
017466  107                     MOV     B,A
017467  315 015 026             CALL    FSYM            ;FIND SYMBOL
017472  353                     XCHG                    ;ADDRESS TO D,E
017473  041 174 060             LXI     H,FREG1         ;COPY STEP TO FREG1
017476  315 271 027             CALL    COPDH
017501  023                     INX     D               ;PNT TO CHARACTERISTIC OF STEP
017502  023                     INX     D
017503  023                     INX     D
017504  032                     LDAX    D               ;GET IT
017505  346 200                 ANI     200Q            ;GET SIGN
017507  027                     RAL                     ;ROTATE IT INTO CARRY
017510  077                     CMC                     ;COMPLEMENT IT
017511  076 000                 MVI     A,0             ;MAKE SURE A=0
017513  027                     RAL                     ;ROTATE TO LSB
017514  074                     INR     A               ;BUMP BY ONE
017515  062 216 060             STA     VLOC            ;SAVE IT, ITS =1 IF - STEP,ELSE= 2
017520  023                     INX     D               ;PNT TO VARIABLE PNTR
017521  353                     XCHG                    ;GET IT TO DE
017522  136                     MOV     E,M
017523  043                     INX     H
017524  126                     MOV     D,M
017525  043                     INX     H
017526  345                     PUSH    H               ;SAVE DATA BLOCK PNTR.
017527  041 200 060             LXI     H,FREG2         ;COPY VARIBLE VALUE TO FREG2
017532  315 271 027             CALL    COPDH           ;SAVE VARIABLE LOCATION IN H,L
017535  353                     XCHG
017536  076 002                 MVI     A,2             ;SET UP TO ADD
017540  315 121 032             CALL    BINOP           ;AND DO IT
017543  315 271 027             CALL    COPDH           ;COPY TO VARIABLE
017546  041 174 060             LXI     H,FREG1         ;AND TO FREG1 FOR COMPARE
017551  315 271 027             CALL    COPDH
017554  321                     POP     D               ;PNT TO LIMIT
017555  041 200 060             LXI     H,FREG2         ;COPY TO FREG2
017560  315 271 027             CALL    COPDH
017563  325                     PUSH    D               ;SAVE DATA BLOCK PNTR
017564  315 071 035             CALL    FCOMP           ;COMPARE
017567  041 216 060             LXI     H,VLOC          ;COMPARE WITH STEP TYPE
017572  276                     CMP     M
017573  341                     POP     H               ;GET DATA BLOCK PNTR.
017574  312 212 037             JZ      NXTDN           ;YES => LOOP DONE
017577  043                     INX     H               ;LOOP NOT DONE
017600  043                     INX     H               ;PNT TO TRANSFER ADD.
017601  043                     INX     H
017602  043                     INX     H
017603  136                     MOV     E,M             ;GET IT TO H,L
017604  043                     INX     H
017605  126                     MOV     D,M
017606  353                     XCHG
017607  303 232 026             JMP     ILOOP
017612  041 224 060     NXTDN:  LXI     H,NEST          ;POP NEST STACK
017615  064                     INR     M
017616  064                     INR     M
017617  303 242 033             JMP     IEND            ;CONTINUE
017622  076 026         ER16:   MVI     A,16H           ;'=' EXPECTED(NOTE: NO ARRAY
017624  303 276 025             JMP     ERROR           ;FOR INDICES)           ELEMENTS
017627  076 027         ER17:   MVI     A,17H           ;BAD SYNTAX NEAR 'TO' OR 'STEP'
017631  303 276 025             JMP     ERROR           ;IN FOR STATEMENT
017634  076 030         ER18:   MVI     A,18H           ;FOR'S NESTED TOO DEEPLY
017636  303 276 025             JMP     ERROR
017641  076 031         ER19:   MVI     A,19H           ;'NEXT' EXECUTED BEFORE A 'FOR'
017643  303 276 025             JMP     ERROR
017646  076 040         ER20:   MVI     A,20H           ;NESTING ERROR, 'FOR'-'NEXT'
017650  303 276 025             JMP     ERROR
017653  076 041         ER21:   MVI     A,21H           ;BAD INDEX IN FOR-NEXT
017655  303 276 025             JMP     ERROR

                        ;
                        ;  THIS SUB CHECKS FOR PAGE BOUNDARY CROSSING
                        ;  OF VARIABLE STORAGE BEFORE UPDATING
                        ;  FORWARD POINTER
                        ;     D-E  POINT TO CURRENT LOCATION OF NEXT VARIABLE
                        ;     H-L  POINT TO PREVIOUS VARIABLE LOCATION
                        ;
                        ;  MODIFY D-E ( IF NECESSARY ) SO VARIABLE WILL NOT CROSS PAGE BOUNDARY
                        ;
017660                  CHKLC:
017660  365                     PUSH    PSW
017661  325                     PUSH    D               ; SEE IF CURRENT VARIABLE
017662  076 007                 MVI     A,7             ; STORAGE 8 WORD BLOCK
017664  203                     ADD     E               ; WILL CROSS PAGE BOUNDARY
017665  332 273 037             JC      CHOVL
                        ;  OK   -  DOES NOT CROSS PAGE
017670  321                     POP     D
017671  361                     POP     PSW
017672  311                     RET
                        ;  PAGE BOUNDARY CROSSED  -  SET D-E TO START OF NEXT PAGE
017673                  CHOVL:
```

```
017673  321                     POP     D
017674  024                     INR     D
017675  036 000                 MVI     E,0
017677  361                     POP     PSW
017700  311                     RET

                        ;
                        ;  THIS SUB IS CALLED FROM 'DIM' PROCESSOR
                        ;  REGS. 'D-E' POINT TO NEXT AVAILABLE WORD OF VARIABLE STORAGE
                        ;  THIS SUB MAKES SURE THAT STORAGE STARTS ON A 4-WORD
                        ;  BOUNDARY SO FLT. PT. NUMBER WILL NOT CROSS PAGE
                        ;
017701                  CKDIM:
017701  173                     MOV     A,E
017702  346 003                 ANI     3
017704  310                     RZ
017705  173                     MOV     A,E
017706  346 374                 ANI     374Q
017710  306 004                 ADI     4
017712  137                     MOV     E,A
017713  172                     MOV     A,D
017714  316 000                 ACI     0
017716  127                     MOV     D,A
017717  311                     RET

                        ; CALL ROUTINES

017720  252 060         FWAM:   DW VEND                 ;DEFINE FWAM POINTER

                        ;ENTRIES TO SUBTABLE

017722                  SUBS:

017722  377                     DB 377Q                 ;END OF TABLE
                        ;       ////FLOATING POINT PACKAGE FOR THE MCS8
                        ;       ////BY DAVID MEAD
                        ;       ////MODIFIED BY HAL BRAND 9/6/74
                        ;       ////MODIFIED FOR 24 BIT MANTISSAS**********
                        ;       ////PLUS ADDED I/O CONVERSION ROUTINES
                        ;       ////NEW ROUTINE COMMENTS
                        ;       ////ARE PRECEEDED BY /
                        ;       ////OTHER CHANGES ARE NOTED BY **
                        ;       ////MODIFIED BY FRANK OLKEN 6/28/75
                        ;
004000                  ;       ORG 4000Q
                        ;
007775                  OUTR    EQU     7775Q           ;LINK TO BASIC
013711                  OUTL    EQU     13711Q
014606                  INL     EQU     14606Q
007772                  INP     EQU     7772Q           ;LINK TO BASIC
000300                  MINCH   EQU     300Q        ;MINIMUM CHARACTERISTIC WITH SIGN
                                                            EXTENDED
000077                  MAXCH   EQU     077Q        ;MAXIMUM CHARACTERISTIC WITH SIGN
                                                            EXTENDED
                        ;*****************************************************
                        ;       //// DIVIDE SUBROUTINE
                        ;*****************************************************
                        ;
004000  315 151 013     LDIV:   CALL    CSIGN           ;COMPUTE SIGN OF RESULT
004003  315 332 011             CALL    ZCHK            ;CHECK IF DIVIDEND = ZERO
004006  302 022 010             JNZ     DTST2           ;IF DIVIDEND .NE. 0 CHECK DIVISOR
004011  315 342 011             CALL    BCHK            ;CHECK FOR ZERO/ZERO
004014  312 250 012             JZ      INDFC           ;ZERO/ZERO = INDEFINITE
004017  303 257 012             JMP     WZERC           ;ZERO/NONZERO = ZERO
004022  315 342 011     DTST2:  CALL    BCHK            ;COME HERE IF DIVIDEND .NE. 0
004025  312 133 013             JZ      OFLWC           ;NONZERO/ZERO = OVERFLOW
                                                        ;IF WE GET HERE, THINGS LOOK OKAY
004030  135                     MOV     E,L             ;SAVE BASE IN E
004031  151                     MOV L,C             ;BASE\6 TO L
004032  315 035 012             CALL DCLR           ;CLEAR QUOTIENT MANTISSA SLOT
004035  153                     MOV L,E             ;RESTORE BASE IN L
004036  315 020 013             CALL ENT1           ;DO FIRST CYCLE
004041  151                     MOV L,C             ;BASE \6 TO L
004042  315 351 011             CALL DLST           ;MOVE QUOTIENT OVER ONE PLACE
004045  026 027                 MVI D,23            ;NUMBER OF ITERATIONS TO D
004047  153             REP3:   MOV L,E
004050  315 012 013             CALL ENT2
004053  025                     DCR D               ;DEC D
004054  312 073 010             JZ  GOON
004057  175                     MOV A,L
004060  151                     MOV L,C             ;BASE\6 TO L
004061  117                     MOV C,A
004062  315 351 011             CALL DLST           ;MOVE QUOTIENT MANT OVER
004065  175                     MOV A,L             ;CPTR TO A
004066  131                     MOV E,C             ;LPTR TO E
004067  117                     MOV C,A             ;CPTR TO C
004070  303 047 010             JMP REP3
                        ;
004073  315 341 012     GOON:    CALL   AORS                ;CHECK IF RESULT IS NORMALIZED
004076  372 115 010             JM  CRIN
004101  175                     MOV A,L             ;LPTR TO A
```

```
004102  151                     MOV L,C         ;CPTR TO L
004103  117                     MOV C,A         ;LPTR TO C
004104  315 351 011             CALL DLST       ;SHIFT QUOTIENT LEFT
004107  115                     MOV C,L
004110  153                     MOV L,E
004111  315 071 013             CALL    LDCP      ;COMPUTE THE CHARACTERISTIC OF RESULT
004114  311                     RET
                        ;
004115  315 114 012     CRIN:    CALL    CFCHE          ;GET A=CHAR(H,L), E=CHAR(H,B)
004120  223                     SUB     E       ;NEW CHAR = CHAR(DIVIDEND) - CHAR
004121  376 177                 CPI     177Q    ;CHECK MAX POSITIVE NUMBER   (DVISIOR)
004123  312 133 013             JZ      OFLWC   ;JUMP ON OVERFLOW
004126  306 001                 ADI     1       ;ADD 1 SINCE WE DID NOT LEFTSHIFT
004130  315 104 013             CALL    CCHK    ;CHECK AND STORE CHARACTERISTIC
004133  311                     RET             ;RETURN
                        ;
                        ;
                        ;*****************************************************
                        ;       //// ADDITION SUBROUTINE
                        ;*****************************************************
                        ;
004134  257             LADD:   XRA A           ;/***SET UP TO ADD
004135  303 142 010             JMP LADS        ;/NOW DO IT
                        ;
                        ;*****************************************************
                        ;       //// SUBTRACTION SUBROUTINE
                        ;*****************************************************
                        ;
004140  076 200         LSUB:   MVI A,200Q      ;/****SET UP TO SUBTRACT
                                                SUBROUTINE LADS
                                                FLOATING POINT ADD OR SUB
                                                A[128 ON ENTRY[SUB
                                                A[0 ON ENTRY[ADD
                                                F-S[F,FIRST OPER DESTROYED
                                                BASE \11 USED FOR SCRATCH
004142  315 357 012     LADS:   CALL ACPR       ;SAVE ENTRY PNT AT BASE \6
004145  315 342 011             CALL    BCHK    ;CHECK ADDEND/SUBTRAHEND = ZERO
004150  310                     RZ              ;IF SO, RESULT=ARG SO RETURN
                                                ;THIS WILL PREVENT UNDERFLOW INDICATION ON
                                                ;ZERO + OR - ZERO
004151  315 133 012             CALL CCMP
004154  312 234 010             JZ  EQ02        ;IF EQUAL, GO ON
004157  127                     MOV D,A         ;SAVE LPTR CHAR IN D
004160  332 177 010             JC  LLTB
004163  223                     SUB E           ;L.GT.B IF HERE
004164  346 177                 ANI 127
004166  127                     MOV D,A         ;DIFFERENCE TO D
004167  135                     MOV E,L         ;SAVE BASE IN E
004170  151                     MOV L,C         ;C PTR TO L
004171  054                     INR L           ;C PTR\1 TO L
004172  163                     MOV M,E         ;SAVE BASE IN C PTR\1
004173  150                     MOV L,B         ;B PTR TO L
004174  303 204 010             JMP NCHK
004177  173             LLTB:   MOV A,E         ;L.LT.B IF HERE,BPTR TO A
004200  222                     SUB D           ;SUBTRACT LPTR CHAR FROM BPTR CHAR
004201  346 177                 ANI 127
004203  127                     MOV D,A         ;DIFFERENCE TO D
004204  076 030         NCHK:   MVI A,24
004206  272                     CMP D
004207  322 214 010             JNC SH10
004212  026 030                 MVI D,24
004214  267             SH10:   ORA A
004215  315 370 011             CALL DRST
004220  025                     DCR D
004221  302 214 010             JNZ SH10
004224  175             EQUL:   MOV A,L
004225  270                     CMP B
004226  302 234 010             JNZ EQ02        ;F.GT.S IF L.NE.B
004231  151                     MOV L,C         ;C PTR TO L
004232  054                     INR L           ;C PTR\1 TO L
004233  156                     MOV L,M         ;RESTORE L
004234  315 002 011     EQ02:   CALL LASD       ;CHECK WHAT TO
004237  315 357 012             CALL ACPR       ;SAVE ANSWER
004242  376 002                 CPI 2           ;TEST FOR ZERO ANSWER
004244  302 252 010             JNZ NOTO
004247  303 215 012             JMP     WZER      ;WRITE FLOATING ZERO AND RETURN
                        ;
004252  026 001         NOTO:   MVI D,1         ;WILL TEST FOR SUB
004254  242                     ANA D
004255  312 326 010             JZ  ADDZ        ;LSB[1 INPLIES SUB
004260  315 347 012             CALL TSTR       ;CHECK NORMAL/REVERSE
004263  312 271 010             JZ  SUBZ        ;IF NORMAL,GO SUBZ
004266  175                     MOV A,L         ;OTHERWISE REVERSE
004267  150                     MOV L,B         ;ROLES
004270  107                     MOV B,A         ;OF L AND B
                        ;
004271  315 046 012     SUBZ:   CALL    DSUB      ;SUBTRACT SMALLER FROM BIGGER
004274  315 357 010             CALL    MANT    ;SET UP SIGN OF RESULT
004277  315 347 012             CALL    TSTR    ;SEE IF WE NEED TO INTERCHANGE
```

```
004302  312 255 011                    JZ      NORM            ;BPTR AND LPTR
                                                                ;NO INTERCHANGE NECESSARY,SO NORMALIZE
                                                                ;AND RETURN
004305  175                            MOV A,L             ;INTERCHANGE
004306  150                            MOV L,B             ;L
004307  107                            MOV B,A             ;AND B
004310  171                            MOV A,C             ;CPTR  TO A
004311  110                            MOV C,B             ;BPTR TO C
004312  135                            MOV E,L             ;LPTR TO E
004313  107                            MOV B,A             ;CPTR TO B
004314  315 044 013                    CALL LXFR           ;MOVE_BPTR> TO _LPTR>
004317  170                            MOV A,B
004320  101                            MOV B,C
004321  117                            MOV C,A
004322  153                            MOV L,E
004323  303 255 011                    JMP     NORM        ;NORMALIZE RESULT AND RETURN

                        ;   COPY THE LARGER CHARACTERISTIC TO THE RESULT

004326  315 133 012     ADDZ:   CALL    CCMP            ;COMPARE THE CHARACTERISTICS
004331  322 337 010             JNC     ADD2            ;IF CHAR(H,L) .GE. CHAR(H,B) CONTINUE
004334  315 215 013             CALL    BCTL            ;IF CHAR(H,L) .LT. CHAR(H,B) THE COPY
                                                        ;CHAR(H,B) TO CHAR(H,L)
004337  315 357 010     ADD2:   CALL    MANT            ;COMPUTE SIGN OF RESULT
004342  315 006 012             CALL    DADD            ;ADD MANTISSAS
004345  322 352 012              JNC    SCCFG               ;IF THERE IS NO OVFLW - DONE
004350  315 370 011             CALL    DRST            ;IF OVERFLOW SHIFT RIGHT
004353  315 266 012             CALL    INCR            ;AND INCREMENT CHARACTERISTIC
004356  311                     RET                     ;ALL DONE, SO RETURN

                        ;   THIS ROUTINE STORES THE MANTISSA SIGN IN THE RESULT
                        ;   THE SIGN HAS PREVIOUSLY BEEN COMPUTED BY LASD.

004357  135             MANT:   MOV E,L             ;SAVE L PTR
004360  151                     MOV L,C             ;C PTR TO L
004361  176                     MOV A,M             ;LOAD INDEX WORD
004362  346 200                 ANI 128             ;SCARF SIGN
004364  153                     MOV L,E             ;RESTORE L PTR
004365  054                     INR L               ;L PTR\2
004366  054                     INR L
004367  054                     INR L               ;TO L
004370  137                     MOV E,A             ;SAVE SIGN IN E
004371  176                     MOV A,M
004372  346 177                 ANI 127             ;SCARF CHAR
004374  203                     ADD E               ;ADD SIGN
004375  167                     MOV M,A             ;STORE IT
004376  055                     DCR L               ;RESTORE
004377  055                     DCR L
004400  055                     DCR L               ;L PTR
004401  311                     RET

                        ;                       SUBROUTINE LASD
                        ;                       UTILITY ROUTINE FOR LADS
                        ;                       CALCULATES TRUE OPER AND SGN
                        ;                       RETURNS ANSWER IN

004402  315 171 013     LASD:   CALL MSFH           ;FETCH MANT SIGNS, F IN A,D
004405  273                     CMP E               ;COMPARE SIGNS
004406  332 064 011             JC  ABCH            ;F\,S- MEANS GO TO A BRANCH
004411  302 075 011             JNZ BBCH            ;F- ,S\ MEANS GO TO B BRANCH
004414  203                     ADD E               ;SAME SIGN IF HERE, ADD SIGNS
004415  332 042 011             JC  BMIN            ;IF BOTH MINUS, WILL OVERFLOW
004420  315 341 012             CALL AORS           ;BOTH POS IF HERE
004423  362 106 011             JP  L000            ;IF AN ADD, LOAD 0
004426  315 364 012     COM1:   CALL DCMP           ;COMPARE F WITH S
004431  332 124 011             JC  L131            ;S.GT.F,SO LOAD 131
004434  302 110 011             JNZ L001            ;F.GT.S,SO LOAD 1
004437  076 002         L002:   MVI A,2             ;ERROR CONDITION, ZERO ANSWER
004441  311                     RET
004442  315 341 012     BMIN:   CALL AORS           ;CHECK FOR ADD OR SUB
004445  362 116 011             JP  L128            ;ADD, SO LOAD 128
004450  315 364 012     COM2:   CALL DCMP           ;COMPARE F WITH S
004453  332 113 011             JC  L003            ;S.GT.F,SO LOAD 3
004456  302 121 011             JNZ L129            ;FGT.S.SO LOAD 129
004461  303 037 011             JMP L002            ;ERROR
004464  315 341 012     ABCH:   CALL AORS           ;FT,S- SO TEST FOR A/S
004467  372 106 011             JM  L000            ;SUBTRACT, SO LOAD 0
004472  303 026 011             JMP COM1            ;ADD, SO GO TO DCMP
004475  315 341 012     BBCH:   CALL AORS           ;F-,S\,SO TEST FOR A/S
004500  372 116 011             JM  L128            ;SUB
004503  303 050 011             JMP COM2            ;ADD
004506  257             L000:   XRA A
004507  311                     RET
004510  076 001         L001:   MVI A,1
004512  311                     RET
004513  076 003         L003:   MVI A,3
004515  311                     RET
004516  076 200         L128:   MVI A,128
004520  311                     RET
004521  076 201         L129:   MVI A,129
004523  311                     RET
004524  076 203         L131:   MVI A,131
004526  311                     RET
```

```
                                ;
                                ;                       SUBROUTINE LMCM
                                ;                       COMPARES THE MAGNITUDE OF
                                ;                       TWO FLOATING PNT NUMBERS
                                ;                       Z[1 IF [,C[1 IF F.LT.S.
004527  315 133 012    LMCM:     CALL CCMP               ;CHECK CHARS
004532  300                       RNZ                     ;RETURN IF NOT EQUAL
004533  315 364 012              CALL DCMP               ;IF EQUAL, CHECK MANTS
004536  311                       RET
                                ;
                                ;
                                ;*************************************************
                                ;       //// MULTIPLY SUBROUTINE
                                ;*************************************************
                                ;
                                ;                       SUBROUTINE LMUL
                                ;                       FLOATING POINT MULTIPLY
                                ;                       L PTR X B PTR TO C PTR
                                ;
004537  315 151 013    LMUL:     CALL    CSIGN           ;COMPUTE SIGN OF RESULT AND STORE IT
004542  315 332 011              CALL    ZCHK            ;CHECK FIRST OPERAND FOR ZERO
004545  312 257 012              JZ      WZERC           ;ZERO * ANYTHING = ZERO
004550  315 342 011              CALL    BCHK            ;CHECK SECOND OPERAND FOR ZERO
004553  312 257 012              JZ      WZERC           ;ANYTHING * ZERO = ZERO
004556  135                       MOV E,L               ;SAVE L PTR
004557  151                       MOV L,C               ;C PTR TO L
004560  315 035 012              CALL DCLR             ;CLR PRODUCT MANT LOCS
004563  153                       MOV L,E               ;L PTR TO L
004564  026 030                  MVI D,24              ;LOAD NUMBER ITERATIONS
004566  315 370 011    KPGO:     CALL DRST             ;SHIFT L PTR RIGHT
004571  332 244 011              JC  MADD              ;WILL ADD B PTR IF C[1
004574  175                       MOV A,L               ;INTERCHANGE
004575  151                       MOV L,C               ;L AND
004576  117                       MOV C,A               ;C PTRS
004577  315 370 011    INTR:     CALL DRST             ;SHIFT PRODUCT OVER
004502  175                       MOV A,L               ;INTERCHANGE
004603  151                       MOV L,C               ;L AND C PTRS_BACK TO
004604  117                       MOV C,A               ;ORIGINAL>
004605  025                       DCR D
004606  302 166 011              JNZ KPGO              ;MORE CYCLES IF Z[0
004611  315 341 012              CALL    AORS            ;TEST IF RESULT IS NORMALIZED
004614  372 100 013              JM      LMCP            ;IF NORMALIZED GO COMPUTE CHAR
004617  135                       MOV     E,L           ;SAVE LPTR IN E
004620  151                       MOV     L,C           ;SET L=CPTR
004621  315 351 011              CALL    DLST          ;LEFT SHIFT RESULT TO NORMALIZE
004624  153                       MOV     L,E           ;RESTORE LPTR
004625  315 114 012              CALL    CFCHE         ;OTHERWISE SET A=CHAR(H,L), E=CHAR(H,B)
004630  203                       ADD     E             ;CHAR(RESULT) = CHAR(H,L) + CHAR(H,B)
004631  376 200                   CPI     200Q          ;CHECK FOR SMALLEST NEGATIVE NUMBER
004633  312 142 013              JZ      UFLWC         ;IF SO THEN UNDERFLOW
004636  326 001                   SUI     1             ;SUBTRACT 1 TO COMPENSATE FOR NORMALIZE
004640  315 104 013              CALL    CCHK          ;CHECK CHARACTERISTIC AND STORE IT
004643  311                       RET                   ;RETURN
                                ;
004644  175            MADD:     MOV A,L               ;INTERCHANGE
004645  151                       MOV L,C               ;L AND
004646  117                       MOV C,A               ;C PTRS
004647  315 006 012              CALL DADD             ;ACCUMULATE PRODUCT
004652  303 177 011              JMP INTR
                                ;
                                ;   SUBROUTINE NORM
                                ;
                                ;       THIS SUBROUTINE WILL NORMALIZE A FLOATING POINT
                                ;       NUMBER, PRESERVING ITS ORIGINAL SIGN.
                                ;       WE CHECK FOR UNDERFLOW AND SET THE CONDITION
                                ;       FLAG APPROPRIATELY.  (SEE ERROR RETURNS).
                                ;       THER IS AN ENTRY POINT TO FLOAT A SIGNED INTEGER
                                ;       (FLOAT) AND AN ENTRY POINT TO FLOAT AN UNSIGNED
                                ;       INTEGER.
                                ;
                                ;   ENTRY POINTS:
                                ;
                                ;       NORM  - NORMALIZE FLOATING PT NUMBER AT (H,L)
                                ;       FLOAT - FLOAT TRIPLE PRECISION INTEGER AT (H,L)
                                ;               PRESERVING SIGN BIT IN (H,L)+3
                                ;       DFXL  - FLOAT UNSIGNED (POSITIVE) TRIPLE PRECISION
                                ;               AT (H,L)
                                ;
                                ;REGISTERS ON EXIT:
                                ;
                                ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                                ;       D,E = GARBAGE
                                ;       B,C,H,L = SAME AS ON ENTRY
                                ;
004655  135            NORM:     MOV     E,L           ;SAVE L IN E
004656  315 101 012    NORM1:    CALL    GCHAR         ;GET CHAR(H,L) IN A WITH SIGN EXTENDED
004661  127                       MOV     D,A           ;SAVE CHAR IN D
004662  153            FXL1:     MOV     L,E           ;RESTORE L
004663  315 332 011    FXL2:     CALL    ZMCHK         ;CHECK FOR ZERO MANTISSA
004666  312 215 012              JZ      WZER          ;IF ZERO MANTISSA THEN ZERO RESULT
004671  176            REP6:     MOV     A,M           ;GET MOST SIGNIFICANT BYTE OF
                                                        ;MANTISSA
```

```
004672  267                     ORA     A               ;SET FLAGS
004673  372 313 011             JM      SCHAR           ;IF MOST SIGNFICANT BIT = 1 THEN
                                                        ;NUMBER IS NORMALIZED AND WE GO TO
                                                        ;STORE THE CHARACTERISTIC
004676  172                     MOV     A,D             ;OTHERWISE CHECK FOR UNDERFLOW
004677  376 300                 CPI     MINCH           ;COMPARE WITH MINIMUM CHAR
004701  312 143 012             JZ      WUND            ;IF EQUAL THEN UNDERFLOW
004704  315 351 011             CALL    DLST            ;SHIFT MANTISSA LEFT
004707  025                     DCR     D               ;DECREMENT CHARACTERSTIC
004710  303 271 011             JMP     REP6            ;LOOP AN TEST NEXT BIT
004713  303 303 012     SCHAR:  JMP     INCR3           ;STORE THE CHARACTERISTIC USING
                                                        ;THE SAME CODE AS THE INCREMENT
                        ;
004716  135             DFXL:   MOV     E,L             ;ENTER HERE TO FLOAT UNSIGNED
                                                        ;INTEGER
                                                        ;FIRT SAVE L IN E
004717  054                     INR     L               ;MAKE (H,L) POINT TO CHAR
004720  054                     INR     L               ;MAKE (H,L) POINT TO CHAR
004721  054                     INR     L               ;MAKE (H,L) POINT TO CHAR
004722  257                     XRA     A               ;ZERO ACCUMULATOR
004723  167                     MOV     M,A             ;STORE A PLUS (+) SIGN
004724  153                     MOV     L,E             ;RESTORE L
004725  026 030         FLOAT:  MVI     D,24            ;ENTER HERE TO FLOAT INTEGER
                                                        ;PRESERVING ORIGINAL SIGN IN (H,L)+3
                                                        ;SET UP CHARACTERISTIC
004727  303 263 011             JMP     FXL2            ;GO FLOAT THE NUMBER
                        ;
                        ;
                        ;   SUBROUTINE ZCHK
                        ;
                        ;       THIS ROUTINE SETS THE ZERO FLAG IF IT DETECTS
                        ;       A FLOATING ZERO AT (H,L).
                        ;
                        ;   SUBROUTINE ZMCHK
                        ;
                        ;       THIS ROUTINE SETS THE ZERO FLAG IF IT DETECTS A
                        ;       ZERO MANTISSA AT (H,L)
                        ;
004732                  ZCHK:
004732  054             ZMCHK:  INR     L               ;SET L TO POINT LAST BYTE OF MANTISSA
004733  054                     INR     L               ;SET L TO POINT TO LAST BYTEOF MANTISSA
004734  176                     MOV     A,M             ;LOAD LEAST SIGNIFICANT BYTE
004735  055                     DCR     L               ;L POINTS TO MIDDLE BYTE
004736  266                     ORA     M               ;OR WITH LEAST SIGNFICANT BYTE
004737  055                     DCR     L               ;L POINTS TO MOST SIGNFICANT BYTE
                                                        ;OF MANTISSA (ORIGINAL VALUE)
004740  266                     ORA     M               ;OR IN MOST SIGNFICANT BYTE
004741  311                     RET                     ;RETURNS WITH ZERO FLAG SET
                                                                        APPROPRIATELY
                        ;   SUBROUTINE BCHK
                        ;
                        ;       THIS ROUTINE CHECKS (H,B) FOR FLOATING PT ZERO
004742  135             BCHK:   MOV     E,L             ;SAVE LPTR IN E
004743  150                     MOV     L,B             ;SET L=BPTR
004744  315 332 011             CALL    ZCHK            ;CHECK FOR ZERO
004747  153                     MOV     L,E             ;RESTORE L=LPTR
004750  311                     RET                     ;RETURN
                        ;
                        ;
                        ;                               SUBROUTINE DLST
                        ;                               SHIFTS DBL WORD ONE PLACE LF
004751  054             DLST:   INR L
004752  054                     INR L
004753  176                     MOV A,M                 ;/***TP
004754  267                     ORA A                   ;LOAD IT
004755  027                     RAL                     ;KILL CARRY
004756  167                     MOV M,A                 ;SHIFT IT LEFT
004757  055                     DCR L                   ;STORE IT
004760  176                     MOV A,M
004761  027                     RAL                     ;LOAD IT
                        ;                               ;SHIFT IT LEFT
                        ;                               IF CARRY SET BY FIRST SHIFT
                        ;                               IT WILL BE IN LSB OF SECOND
004762  167                     MOV M,A
004763  055                     DCR L                   ;/***TP EXTENSION
004764  176                     MOV A,M
004765  027                     RAL
004766  167                     MOV M,A                 ;/***ALL DONE TP
004767  311                     RET
                        ;                               SUBROUTINE DRST
                        ;                               SHIFTS DOUBLE WORD ONE PLACE
                        ;                               TO THE RIGHT
                        ;                               DOES NOT AFFECT D
004770  135             DRST:   MOV E,L                 ;/***TP MODIFIED RIGHT SHIFT TP
004771  176                     MOV A,M                 ;LOAD FIRST WORD
004772  037                     RAR                     ;ROTATE IT RIGHT
004773  167                     MOV M,A                 ;STORE IT
004774  054                     INR L                   ;/*** TP
004775  176                     MOV A,M                 ;LOAD SECOND WORD
004776  037                     RAR                     ;SHIFT IT RIGHT
004777  167                     MOV M,A                 ;STORE IT
```

```
005000  054                     INR L                   ;/*** TP EXTENSION
005001  176                     MOV A,M
005002  037                     RAR
005003  167                     MOV M,A
005004  153                     MOV L,E                 ;/***TP - ALL DONE TP
005005  311                     RET
                        ;                               SUBROUTINE DADD
                        ;                               ADDS TWO DOUBLE PRECISION
                        ;                               WORDS, CY=1 IF THERE IS OVRFLW
005006  135            DADD:    MOV E,L                 ;SAVE BASE IN E
005007  150                     MOV L,B                 ;BASE \3 TO L
005010  054                     INR L                   ;BASE \4 TO L
005011  054                     INR L                   ;/***TP
005012  176                     MOV A,M                 ;LOAD S MANTB
005013  153                     MOV L,E                 ;BASE TO L
005014  054                     INR L                   ;BASE \1 TO L
005015  054                     INR L                   ;/***TP
005016  206                     ADD M                   ;ADD TWO MANTBJS
005017  167                     MOV M,A                 ;STORE ANSWER
005020  150                     MOV L,B                 ;/***TP EXTENSION
005021  054                     INR L
005022  176                     MOV A,M
005023  153                     MOV L,E
005024  054                     INR L
005025  216                     ADC M
005026  167                     MOV M,A                 ;/***TP - ALL DONE
005027  150                     MOV L,B                 ;BASE \3 TO L
005030  176                     MOV A,M                 ;MANTA OF S TO A
005031  153                     MOV L,E                 ;BASE TO L
005032  216                     ADC M                   ;ADD WITH CARRY
005033  167                     MOV M,A                 ;STORE ANSWER
005034  311                     RET
                        ;                               SUBROUTINE DCLR
                        ;                               CLEARS TWO SUCCESSIVE
                        ;                               LOCATIONS OF MEMORY
005035  257            DCLR:    XRA A
005036  167                     MOV M,A
005037  054                     INR L
005040  167                     MOV M,A
005041  054                     INR L                   ;/***TP EXTENSION
005042  167                     MOV M,A                 ;/***TP ZERO 3
005043  055                     DCR L                   ;/***TP - ALL DONE
005044  055                     DCR L
005045  311                     RET
                        ;                               /*****ALL NEW DSUB - SHORTER***
                        ;                               SUBROUTINE DSUB
                        ;                               DOUBLE PRECISION SUBTRACT
005046  135            DSUB:    MOV E,L                 ;SAVE BASE IN E
005047  054                     INR L                   ;/***TP EXTENSION
005050  054                     INR L                   ;/START WITH LOWS
005051  176                     MOV A,M                 ;/GET ARG
005052  150                     MOV L,B                 ;/NOW SET UP TO SUB
005053  054                     INR L
005054  054                     INR L
005055  226                     SUB M                   ;/NOW DO IT
005056  153                     MOV L,E                 ;/NOW MUST PUT IT BACK
005057  054                     INR L
005060  054                     INR L
005061  167                     MOV M,A                 ;/PUT BACK
005062  055                     DCR L                   ;/***TP - ALL DONE
005063  176                     MOV A,M                 ;/GET LOW OF LOP
005064  150                     MOV L,B                 ;/SET TO BOP
005065  054                     INR L                   ;/SET TO BOP LOW
005066  236                     SBB M                   ;/GET DIFF. OF LOWS
005067  153                     MOV L,E                 ;/SAVE IN LOP LOW
005070  054                     INR L                   ;/TO LOP LOW
005071  167                     MOV M,A                 ;/INTO RAM
005072  055                     DCR L                   ;/BACK UP TO LOP HIGH
005073  176                     MOV A,M                 ;/GET LOP HIGH
005074  150                     MOV L,B                 ;/SET TO BOP HIGH
005075  236                     SBB M                   ;/SUB. WITH CARRY
005076  153                     MOV L,E                 ;/SAVE IN LOP HIGH
005077  167                     MOV M,A                 ;/INTO RAM
005100  311                     RET                     ;/ALL DONE - MUCH SHORTER
                        ;
                        ;   SUBROUTINE GCHAR
                        ;
                        ;       THIS SUBROUTINE RETURNS THE CHARACTERISTIC OF
                        ;       THE FLOATING POINT NUMBER POINTED TO BY (H,L)
                        ;       IN THE A REGISTER WITH ITS SIGN EXTENDED INTO THE
                        ;       LEFTMOST BIT.
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CHARACTERISTIC OF (H,L) WITH SIGN EXTENDED
                        ;       L = (ORIGINAL L) + 3
                        ;       B,C,D,E,H = SAME AS ON ENTRY
005101  054            GCHAR:   INR     L              ;MAKE (H,L) POINT TO CHAR
005102  054                     INR     L              ;MAKE (H,L) POINT TO CHAR
005103  054                     INR     L              ;MAKE (H,L) POINT TO CHAR
005104  176                     MOV     A,M            ;SET A=CHAR + MANTISSA SIGN
005105  346 177                 ANI     177Q           ;GET RID OF MANTISSA SIGN BIT
```

```
005107  306 100                 ADI     100Q        ;PROPAGATE CHAR SIGN INTO LEFTMOST BIT
005111  356 100                 XRI     100Q        ;RESTORE ORIGINAL CHAR SIGN BIT
005113  311                     RET                 ;RETURN WITH (H,L) POINTING TO THE
                                                    ;CHAR = ORIGINAL (H,L)+3
                                                    ;SOMEONE ELSE WILL CLEAN UP
                        ;
                        ;   SUBROUTINE CFCHE
                        ;
                        ;       THIS SUBROUTINE RETURNS THE CHARACTERISTICS OF THE
                        ;       FLOATING POINT NUMBERS POINTED TO BY (H,L) AND
                        ;       (H,B) IN THE A AND E REGISTERS RESPECTIVELY,
                        ;       WITH THEIR SIGNS EXTENDED INTO THE LEFTMOST BIT.
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CHARACTERISTIC OF (H,L) WITH SIGN EXTENDED
                        ;       E = CHARACTERISTIC OF (H,B) WITH SIGN EXTENDED
                        ;       B,C,H,L = SAME AS ON ENTRY
                        ;       D = A
                        ;
005114  135             CFCHE:  MOV     E,L         ;SAVE LPTR IN E
005115  150                     MOV     L,B         ;SET L = BPTR
005116  315 101 012            CALL    GCHAR       ;GET CHAR(H,B) WITH SIGN EXTENDED IN A
005121  153                     MOV     L,E         ;RESTORE L = LPTR
005122  137                     MOV     E,A         ;SET E=CHAR(H,B) WITH SIGN EXTENDED
005123  315 101 012            CALL    GCHAR       ;SET A=CHAR(H,L) WITH SIGN EXTENDED
005126  055                     DCR     L           ;RESTORE L = LPTR
005127  055                     DCR     L           ;RESTORE L = LPTR
005130  055                     DCR     L           ;RESTORE L = LPTR
005131  127                     MOV     D,A         ;SET D=A=CHAR(H,L) WITH SIGN EXTENDED
005132  311                     RET
                        ;
                        ;   SUBROUTINE CCMP
                        ;
                        ;       THIS SUBROUTINE COMPARES THE CHARACTERISTICS OF
                        ;       FLOATING POINT NUMBERS POINTED TO BY (H,L) AND (H,B).
                        ;       THE ZERO FLIP-FLOP IS SET IF CHAR(H,L) EQUALS
                        ;       CHAR(H,B).  IF CHAR(H,L) IS LESS THAN CHAR(H,B) THEN
                        ;       THE CARRY BIT WILL BE SET.
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CHARACTERISTIC OF (H,L) WITH SIGN EXTENDED
                        ;       E = CHARACTERISTIC OF (H,B) WITH SIGN EXTENDED
                        ;       D = A
                        ;       B,C,H,L = SAME AS ON ENTRY
005133  315 114 012    CCMP:   CALL    CFCHE       ;FETCH CHARACTERTISTICS WITH SIGN EXTENDED
                                                    ;INTO A (CHAR(H,L)) AND E (CHAR(H,B)) REGISTERS
005136  127                     MOV     D,A         ;SAVE CHAR (H,L)
005137  223                     SUB     E           ;SUBTRACT E (CHAR(H,B))
005140  027                     RAL                 ;ROTATE SIGN BIT INTO CARRY BIT
005141  172                     MOV     A,D         ;RESTORE A=CHAR(H,L)
005142  311                     RET                 ;RETURN
                        ;
                        ;   ERROR RETURNS
                        ;
                        ;       THE FOLLOWING CODE IS USED TO RETURN VARIOUS
                        ;       ERROR CONDITIONS.  IN EACH CASE A FLOATING POINT
                        ;       NUMBER IS STORED IN  THE 4 WORDS POINTED TO BY (H,L)
                        ;       AND A FLAG IS STORED IN THE ACCUMULATOR.
                        ;
                        ;       CONDITION    FLAG   RESULT (+)          RESULT (-)
                        ;
                        ;       UNDERFLOW     377   000 000 000 100     000 000 000 300
                        ;       OVERFLOW      177   377 377 377 077     377 377 377 277
                        ;       INDEFINITE    077   377 377 377 077     377 377 377 277
                        ;       NORMAL        000   XXX XXX XXX XXX     XXX XXX XXX XXX
                        ;       NORMAL ZERO   000   000 000 000 100     (ALWAYS RETURNS +0)
                        ;
                        ;   ENTRY POINTS:
                        ;
                        ;       WUND - WRITE UNDERFLOW
                        ;       WOVR - WRITE OVERFLOW
                        ;       WIND - WRITE INDEFINITE
                        ;       WZER - WRITE NORMAL ZERO
                        ;
      1                 WFLT    MACRO   VMANT,VCHAR,VFLAG,LABEL  ;WRITE FLOATING NUMBER
      1                 ;
      1                         MVI     D,VCHAR     ;LOAD CHARACTERISTIC INTO D REGISTER
      1                         CALL    WCHAR       ;WRITE CHARACTERISTIC
      1                 LABEL:: MVI     A,VMANT     ;LOAD MANTISSA VALUE
      1                                             ;WE ASSUME HERE THAT ALL BYTES OF MANTISSA
      1                                             ;ARE THE SAME
      1                         CALL    WMANT       ;WRITE THE MANTISSA
      1                         MVI     A,VFLAG     ;SET ACCUMULATOR TO FLAG
      1                         ORA     A           ;SET FLAGS PROPERLY
      1                         RET                 ;RETURN (WMANT RESTORED (H,L))
      1                         ENDM
                        ;
005143 1                WUND:   WFLT    0,100Q,377Q,UFLW1  ;WRITE UNDERFLOW
```

```
                1
                +
005143  1  026 100          +           MVI     D,00010H        ;LOAD CHARACTERISTIC INTO D REGISTER
005145  1  315 237 012      +           CALL    WCHAR           ;WRITE CHARACTERISTIC
005150  1  076 000          +UFLW1::    MVI     A,00000H        ;LOAD MANTISSA VALUE
                1                                               ;WE ASSUME HERE THAT ALL BYTES OF
                1                                               ;ARE THE SAME                   MANTISSA
005152  1  315 230 012      +           CALL    WMANT           ;WRITE THE MANTISSA
005155  1  076 377          +           MVI     A,000FFH        ;SET ACCUMULATOR TO FLAG
005157  1  267              +           ORA     A               ;SET FLAGS PROPERLY
005160  1  311              +           RET                     ;RETURN (WMANT RESTORED (H,L))
005161  1                   +WOVR:      WFLT    377Q,77Q,177Q,0FLW1  ;WRITE OVERFLOW
                1                   +
005161  1  026 077          +           MVI     D,0003FH        ;LOAD CHARACTERISTIC INTO D REGISTER
005163  1  315 237 012      +           CALL    WCHAR           ;WRITE CHARACTERISTIC
005166  1  076 377          +OFLW1::    MVI     A,000FFH        ;LOAD MANTISSA VALUE
                1                                               ;WE ASSUME HERE THAT ALL BYTES OF
                1                                               ;ARE THE SAME                   MANTISSA
005170  1  315 230 012      +           CALL    WMANT           ;WRITE THE MANTISSA
005173  1  076 177          +           MVI     A,000/FH        ;SET ACCUMULATOR TO FLAG
005175  1  267              +           ORA     A               ;SET FLAGS PROPERLY
005176  1  311              +           RET                     ;RETURN (WMANT RESTORED (H,L))
005177  1                   +WIND:      WFLT    377Q,77Q,77Q,1NDF1  ;WRITE INDEFINITE
                1                   +
005177  1  026 077          +           MVI     D,0003FH        ;LOAD CHARACTERISTIC INTO D REGISTER
005201  1  315 237 012      +           CALL    WCHAR           ;WRITE CHARACTERISTIC
005204  1  076 377          +INDF1::    MVI     A,000FFH        ;LOAD MANTISSA VALUE
                1                                               ;WE ASSUME HERE THAT ALL BYTES OF
                1                                               ;ARE THE SAME                   MANTISSA
005206  1  315 230 012      +           CALL    WMANT           ;WRITE THE MANTISSA
005211  1  076 077          +           MVI     A,0003FH        ;SET ACCUMULATOR TO FLAG
005213  1  267              +           ORA     A               ;SET FLAGS PROPERLY
005214  1  311              +           RET                     ;RETURN (WMANT RESTORED (H,L))

005215     054               WZER:       INR     L               ;WRITE NORMAL ZERO
005216     054                           INR     L               ;
005217     054                           INR     L               ;
005220     066 100                       MVI     M,100Q          ;STORE CHARACTERISTIC FOR ZERO
005222     257                           XRA     A               ;ZERO ACCUMULATOR
005223     315 230 012                   CALL    WMANT           ;STORE ZERO MANTISSA
005226     267                           ORA     A               ;SET FLAGS PROPERLY
005227     311                           RET                     ;RETURN

                        ;  ROUTINE TO WRITE MANTISSA FOR ERROR RETURNS
                        ;
005230     055               WMANT:      DCR     L               ;POINT LEAST SIGNIFICANT BYTE
                                                                 ;OF MANTISSA
005231     167                           MOV     M,A             ;STORE LSBYTE OF MANTISSA
005232     055                           DCR     L               ;POINT TO NEXT LEAST SIGNIFICANT BYTE
                                                                 ;OF MANTISSA
005233     167                           MOV     M,A             ;STORE NLSBYTE OF MANTISSA
005234     055                           DCR     L               ;POINT TO MOST SIGNIFICANT BYTE
                                                                 ;OF MANTISSA
005235     167                           MOV     M,A             ;STORE MSBYTE OF MANTISSA
005236     311                           RET                     ;RETURN (H,L) POINTS TO BEGINNING OF
                                                                 ;FLOATING POINT RESULT
                        ;  ROUTINE TO WRITE CHARACTERTIC FOR ERROR RETURNS
                        ;  NOTE:  WE PRESERVE ORIGINAL MANTISSA SIGN
                        ;  ON ENTRY D CONTAINS NEW CHARACTERTISTIC TO BE STORED.
                        ;
005237     054               WCHAR:      INR     L               ;SET (H,L) TO POINT TO CHARACTERISTIC
005240     054                           INR     L               ;PART OF ABOVE
005241     054                           INR     L               ;PART OF ABOVE
005242     176                           MOV     A,M             ;LOAD CHARACTERISTIC A
                                                                 ;AND MANTISSA SIGN
005243     346 200                       ANI     200Q            ;JUST KEEP MANTISSA SIGN
005245     262                           ORA     D               ;OR IN NEW CHARACTERISTIC
005246     167                           MOV     M,A             ;STORE IT BACK
005247     311                           RET                     ;RETURN WITH (H,L) POINT TO
                                                                 ;OF RESULT                CHARACTERISTIC
                                                                 ;SOMEONE ELSE WILL FIX UP (H,L)

                        ;    SUBROUTINE INDFC
                        ;
                        ;        THIS ROUTINE WRITES A FLOATING INDEFINITE, SETS
                        ;        THIS WRITES WRITES A FLOATING POINT INDEFINITE
                        ;        AT (H,C), SETS THE CONDITION FLAG AND RETURNS
                        ;
                        ;
005250     135               INDFC:      MOV     E,L             ;SAVE LPTR IN E
005251     151                           MOV     L,C             ;SET L=CPTR SO (H,L)-ADDR OF RESULT
005252     315 177 012                   CALL    WIND            ;WRITE INDEFINITE
005255     153                           MOV     L,E             ;RESTORE L=LPTR
005256     311                           RET                     ;RETURN
                        ;
                        ;    SUBROUTINE WZERC
                        ;
                        ;        THIS ROUTINE WRITES A NORMAL FLAOTING POINT ZERO
                        ;        AT (H,C), SETS THE CONDITION FLAG AND RETURNS
                        ;
005257     135               WZERC:      MOV     E,L             ;SAVE LPTR IN E
005260     151                           MOV     L,C             ;SETL=CPTR SO (H,L)=ADDR OF RESULT
005261     315 215 012                   CALL    WZER            ;WRITE NORMAL ZERO
005264     153                           MOV     L,E             ;RESTORE L=LPTR
                                         RET                     ;RETURN
```

```
005265  311                     RET                     ;RETURN
                        ;
                        ;   SUBROUTINE INCR
                        ;
                        ;       THIS SUBROUTINE INCREMENTS THE CHARACTERISTIC
                        ;       OF THE FLOATING POINT NUMBER POINTED TO BY (H,L).
                        ;       WE TEST FOR OVERFLOW AND SET APPROPRIATE FLAG.
                        ;       (SEE ERRROR RETURNS).
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                        ;       D = CLOBBERED
                        ;       B,C,H,L = SAME AS ON ENTRY
                        ;
005266  315 101 012     INCR:   CALL    GCHAR           ;GET CHAR WITH SIGN EXTENDED
005271  376 077                 CPI     MAXCH           ;COMPARE WITH MAX CHAR PERMITTED
005273  312 106 012             JZ      OFLW1           ;INCREMENT WOULD CAUSE OVERFLOW
005276  127                     MOV D,A                 ;/SAVE IT IN D
005277  024                     INR D                   ;/INCREMENT IT
005300  303 306 012             JMP     INCR2           ;JUMP AROUND ALTERNATE ENTRY POINT
005303  054             INCR3:  INR     L               ;COME HERE TO STORE CHARACTERISTIC
005304  054                     INR     L               ;POINT (H,L) TO CHAR
005305  054                     INR     L               ;POINT (H,L) TO CHAR
005306  076 177         INCR2:  MVI A,177Q
005310  242                     ANA D                   ;/KILL SIGN BIT
005311  127                     MOV D,A                 ;/BACK TO D
005312  176                     MOV A,M                 ;/NOW SIGN IT
005313  346 200                 ANI 200Q                ;/GET MANTISSA SIGN
005315  262                     ORA D                   ;/PUT TOGETHER
005316  167                     MOV M,A                 ;/STORE IT BACK
005317  055                     DCR L                   ;/NOW BACK TO BASE
005320  055                     DCR L                   ;/***TP
005321  055                     DCR L
005322  257             SCCFG:   XRA    A                       ;SET SUCCESS FLAG
005323  311                     RET
                        ;
                        ;   SUBROUTINE DECR
                        ;
                        ;       THIS SUBROUTINE DECREMENTS THE CHARACTERISTIC
                        ;       OF THE FLOATING POINT NUMBER POINTED TO BY (H,L).
                        ;       WE TEST FOR UNDERFLOW AND SET APPROPRIATE FLAG.
                        ;       (SEE ERRROR RETURNS).
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                        ;       D = CLOBBERED
                        ;       B,C,H,L = SAME AS ON ENTRY
                        ;
005324  315 101 012     DECR:   CALL    GCHAR           ;GET CHAR WITH SIGN EXTENDED
005327  376 300                 CPI     MINCH           ;COMPARE WITH MIN CHAR PERMITTED
005331  312 150 012             JZ      UFLW1           ;DECREMENT WOULD CAUSE UNDERFLOW
005334  127                     MOV     D,A             ;SAVE CHARACTERSTIC IN D
005335  025                     DCR     D               ;DECREMENT CHARACTERISTIC
005336  303 306 012             JMP     INCR2           ;GO STORE IT BACK
                        ;
                        ;                               SUBROUTINE AORS
                        ;                               RETURN SC1 IF BASE \6
                        ;                               HAS A 1 IN MSB
005341  135             AORS:   MOV E,L                 ;SAVE BASE
005342  151                     MOV L,C                 ;BASE \6 TO L
005343  176                     MOV A,M                 ;LOAD IT
005344  267                     ORA A                   ;SET FLAGS
005345  153                     MOV L,E                 ;RESTORE BASE
005346  311                     RET
                        ;                               SUBROUTINE TSTR
                        ;                               CHECKS C PTR TO SEE IF
                        ;                               NLSB[1
                        ;                               RETURNS Z[1 IF NOT
                        ;                               DESTROYS E,D
005347  135             TSTR:   MOV E,L                 ;SAVE BASE
005350  151                     MOV L,C                 ;C PTR TO L
005351  026 002                 MVI D,2                 ;MASK TO D
005353  176                     MOV A,M                 ;LOAD VALUE
005354  153                     MOV L,E                 ;RESTORE BASE
005355  242                     ANA D                   ;AND VALUE WITH MASK
005356  311                     RET
                        ;                               SUBROUTINE ACPR
                        ;                               STORES A IN LOCATION OF CPTR
                        ;                               LPTR IN E
005357  135             ACPR:   MOV E,L                 ;SAVE LPTR
005360  151                     MOV L,C                 ;CPTR TO L
005361  167                     MOV M,A                 ;STORE A
005362  153                     MOV L,E                 ;RESTORE BASE
005363  311                     RET
                        ;                               SUBROUTINE DCMP
                        ;                               COMPARES TWO DOUBLE LENGTH
                        ;                               WORDS
005364  176             DCMP:   MOV A,M                 ;NUM MANTA TO A
005365  135                     MOV E,L                 ;SAVE BASE IN E
005366  150                     MOV L,B                 ;BASE\3 TO L
005367  276                     CMP M                   ;COMPARE WITH DEN MANTA
```

```
005370  153                     MOV L,E         ;RETURN BASE TO L
005371  300                     RNZ             ;RETURN IF NOT THE SAME
005372  054                     INR L           ;L TO NUM MANTB
005373  176                     MOV A,M         ;LOAD IT
005374  150                     MOV L,B         ;DEN MANTB ADD TO L
005375  054                     INR L           ;BASE\ 4 TO L
005376  276                     CMP M
005377  153                     MOV L,E
005400  300                     RNZ             ;/***TP EXTENSION
005401  054                     INR L           ;/NOW CHECK BYTE 3
005402  054                     INR L
005403  176                     MOV A,M         ;/GET FOR COMPARE
005404  150                     MOV L,B
005405  054                     INR L
005406  054                     INR L           ;/BYTE 3 NOW
005407  276                     CMP M           ;/COMPARE
005410  153                     MOV L,E         ;/***TP - ALL DONE
005411  311                     RET
                        ;                       SUBROUTINE DIVC
                        ;                       PERFORMS ONE CYCLE OF DOUBLE
                        ;                       PRECISION FLOATING PT DIVIDE
                        ;                       ENTER AT ENT1 ON FIRST CYCLE
                        ;                       ENTER AT ENT2 ALL THEREAFTER
005412  315 351 011     ENT2:   CALL DLST       ;SHIFT MOVING DIVIDEND
005415  332 027 013             JC  OVER        ;IF CARRY[1,NUM.GT.D
005420  315 364 012     ENT1:   CALL DCMP       ;COMPARE NUM WITH DEN
005423  322 027 013             JNC OVER        ;IF CARRY NOT SET,NUM.GE.DEN
005426  311                     RET
005427  315 046 012     OVER:   CALL DSUB       ;CALL DOUBLE SUBTRACT
005432  135                     MOV E,L         ;SAVE BASE IN E
005433  151                     MOV L,C         ;BASE \6 TO L
005434  054                     INR L           ;BASE \7 TO L
005435  054                     INR L           ;/***TP
005436  176                     MOV A,M
005437  306 001                 ADI 1           ;ADD 1
005441  167                     MOV M,A         ;PUT IT BACK
005442  153                     MOV L,E         ;RESTORE BASE TO L
005443  311                     RET
                        ;                       SUBROUTINE LXFR
                        ;                       MOVES CPTR TO EPTR
                        ;                       MOVES 3 WORDS IF ENTER AT LXFR
005444  026 004         LXFR:   MVI D,4         ;/MOVE 4 WORDS
005446  151             REP5:   MOV L,C         ;CPTR TO L
005447  176                     MOV A,M         ;_CPTR> TO A
005450  153                     MOV L,E         ;EPTR TO L
005451  167                     MOV M,A
005452  014                     INR C           ;/INCREMENT C
005453  034                     INR E           ;/INCREMENT E TO NEXT
005454  025                     DCR D           ;/TEST FOR DONE
005455  302 046 013             JNZ REP5        ;/GO FOR FOR TILL D=0
005460  173                     MOV A,E         ;/NOW RESET C AND E
005461  326 004                 SUI 4           ;/RESET BACK BY 4
005463  137                     MOV E,A         ;/PUT BACK IN E
005464  171                     MOV A,C         ;/NOW RESET C
005465  326 004                 SUI 4           ;/BY 4
005467  117                     MOV C,A         ;/BACK TO C
005470  311                     RET             ;/DONE
                        ;
                        ;   SUBROUTINE LDCP
                        ;
                        ;       THIS SUBROUTINE COMPUTES THE CHARACTERISTIC
                        ;       FOR THE FLOATING DIVIDE ROUTINE
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                        ;       D,E = GARBAGE
                        ;       B,C,H,L = SAME AS ON ENTRY
                        ;
                        ;   REGISTERS ON ENTRY:
                        ;
                        ;       (H,B) = ADDRESS OFF DIVISOR
                        ;       (H,C) = ADDRESS OF QUOTIENT
                        ;       (H,L) = ADDRESS OF DIVIDEND
005471  315 114 012     LDCP:   CALL    CFCHE   ;SET E=CHAR(H,B), A=CHAR(H,L)
005474  223                     SUB     E       ;SUBTRACT TO GET NEW CHARACTERISTIC
005475  303 104 013             JMP     CCHK    ;GO CHECK FOR OVER/UNDERFLOW
                                                ;AND STORE CHARACTERTISTIC
                        ;
                        ;   SUBROUTINE LMCP
                        ;
                        ;       THIS SUBROUTINE COMPUTES THE CHARACTERISTIC
                        ;       FOR THE FLOATING MULTIPLY ROUTINE.
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                        ;       D,E = GARBAGE
                        ;       B,C,H,L = SAME AS ON ENTRY
```

```
                        ;   REGISTERS ON ENTRY:
                        ;
                        ;       (H,B) = ADDRESS OFF MULTIPLICAND
                        ;       (H,C) = ADDRESS OF PRODUCT
                        ;       (H,L) = ADDRESS OF MULTIPLIER
005500  315 114 012     LMCP:   CALL    CFCHE           ;SET E=CHAR(H,B), A=CHAR(H,L)
005503  203                     ADD     E               ;ADD TO GET NEW CHARACTERISTIC
                                                        ;NOW FALL INTO THE ROUTINE
                                                        ;WHICH CHECKS FOR OVER/UNDERFLOW
                                                        ;AND STORE CHARACTERTISTIC
                        ;
                        ;   SBUROUTINE CCHK
                        ;
                        ;       THIS SUBROUTINE CHECKS A CHARACTERISTIC IN
                        ;       THE ACCUMULATOR FOR OVERFLOW OR UNDERFLOW.
                        ;       IT THEN STORES THE CHARACTERISTIC, PRESERVING
                        ;       THE PREVIOUSLY COMPUTED MANTISSA SIGN.
                        ;
                        ;   REGISTERS ON ENTRY:
                        ;
                        ;       (H,L) = ADDRESS OF ONE OPERAND
                        ;       (H,B) = ADDRESS OF OTHER OPERAND
                        ;       (H,C) = ADDRESS OF RESULT
                        ;       A     = NEW CHARACTERISTIC OF  RESULT
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A = CONDITION FLAG (SEE ERROR RETURNS)
                        ;       D,E = GARBAGE
                        ;       B,C,H,L = SAME AS ON ENTRY
005504                  CCHK:                           ;ENTER HERE TO CHECK CHARACTERISTIC
005504  376 100                 CPI     100Q            ;CHECK FOR 0 TO +63
005506  332 123 013             JC      STORC           ;JUMP IF OKAY
005511  376 200                 CPI     200Q            ;CHECK FOR +64 TO +127
005513  332 133 013             JC      OFLWC           ;JUMP IF OVERFLOW
005516  376 300                 CPI     300Q            ;CHECK FOR -128 TO -65
005520  332 142 013             JC      UFLWC           ;JUMP IF UNDERFLOW
005523  135             STORC:  MOV     E,L             ;SAVE L IN E
005524  151                     MOV     L,C             ;LET L POINT TO RESULT
005525  127                     MOV     D,A             ;SAVE CHARACTERISTIC IN D
005526  315 303 012             CALL    INCR3           ;STORE CHARACTERISTIC
005531  153                     MOV     L,E             ;RESTORE L
005532  311                     RET                     ;RETURN
                        ;
                        ;   SUBROUTINE OFLWC
                        ;
                        ;       THIS ROUTINE WRITES A FLOATING POINT OVERFLOW AT (H,C)
                        ;       SETS THE CONDITION FLAG, AND RETURNS.
                        ;
005533  135             OFLWC:  MOV     E,L             ;SAVE L IN E
005534  151                     MOV     L,C             ;SET L=CPTR, SO (H,L)=ADDR OF RESULT
005535  315 161 012             CALL    WOVR            ;WRITE OUT OVERFLOW
005540  153                     MOV     L,E             ;RESTORE L
005541  311                     RET                     ;RETURN
                        ;
                        ;   SUBROUTINE UFLWC
                        ;
                        ;       THIS ROUTINE WRITES A FLOATING POINT UNDERFLOW AT (H,C)
                        ;       SETS THE CONDITION FLAG, AND RETURNS.
                        ;
005542  135             UFLWC:  MOV     E,L             ;SAVE L IN E
005543  151                     MOV     L,C             ;SET L=CPTR, SO (H,L)=ADDR OF RESULT
005544  315 143 012             CALL    WUND            ;WRITE OUT UNDEFLOW
005547  153                     MOV     L,E             ;RESTORE L
005550  311                     RET                     ;RETURN
                        ;
                        ;   SUBROUTINE CSIGN
                        ;
                        ;       THIS SUBROUTINE COMPUTES AND STORE THE MANTISSA
                        ;       SIGN FOR THE FLOATING MULTIPLY AND DIVIDE ROUTINES
                        ;
                        ;   REGISTERS ON ENTRY:
                        ;
                        ;       (H,L) = ADDRESS OF ONE OPERAND
                        ;       (H,B) = ADDRESS OF OTHER OPERAND
                        ;       (H,C) = ADDRESS OF RESULT
                        ;
                        ;   REGISTERS ON EXIT:
                        ;
                        ;       A,D,E = GARBAGE
                        ;       B,C,H,L = SAME AS ON ENTRY
                        ;
005551  315 171 013     CSIGN:  CALL    MSFH            ;SET A=SIGN(H,L), E=SIGN(H,B)
005554  253                     XRA     E               ;EXCLUSIVE OR SIGNS TO GET NEW SIGN
005555  315 161 013             CALL    CSTR            ;STORE SIGN INTO RESULT
005560  311                     RET                     ;RETURN
                        ;
```

```
                                ;                               SUBROUTINE CSTR
                                ;                               STORES VALUE IN A IN
                                ;                               CPTR\2
                                ;                               PUTS LPTR IN E
005561  135                     CSTR:   MOV E,L                 ;SAVE LPTR IN E
005562  151                             MOV L,C                 ;CPTR TO L
005563  054                             INR L                   ;CPTR\2
005564  054                             INR L                   ;TO L
005565  054                             INR L                   ;/***TP
005566  167                             MOV M,A                 ;STORE ANSWER
005567  153                             MOV L,E                 ;LPTR BACK TO L
005570  311                             RET
                                ;   SUBROUTINE MSFH
                                ;       THIS SUBROUTINE FETCHES THE SIGNS OF THE MANTISSAS
                                ;       OF THE FLOATING POINT NUMBERS POINTED TO BY (H,L)
                                ;       AND (H,B) INTO THE A AND E REGISTERS RESPECTIVELY.
                                ;   REGISTERS ON EXIT:
                                ;       A = SIGN  OF MANTISSA OF (H,L)
                                ;       E = SIGN OF MANTISSA OF (H,B)
                                ;       B,C,D,H,L = SAME AS ON ENTRY
005571  135                     MSFH:   MOV E,L                 ;SAVE LPTR
005572  150                             MOV L,B                 ;BPTR TO L
005573  054                             INR L                   ;BPTR\2
005574  054                             INR L                   ;/***TP
005575  054                             INR L                   ;TO L
005576  176                             MOV A,M                 ;_BPTR\2>TO A
005577  346 200                         ANI 128                 ;SAVE MANT SIGN
005601  153                             MOV L,E                 ;LPTR BACK TO L
005602  137                             MOV E,A                 ;STORE BPTR MANT SIGN
005603  054                             INR L                   ;LPTR\2
005604  054                             INR L                   ;/***TP
005605  054                             INR L                   ;TO L
005606  176                             MOV A,M                 ;_LPTR\2>TO A
005607  346 200                         ANI 128                 ;SAVE LPTR MANT SIGN
005611  055                             DCR L                   ;LPTR BACK
005612  055                             DCR L                   ;TO L
005613  055                             DCR L                   ;/***TP
005614  311                             RET
                                ;                               SUBROUTINE BCTL
                                ;                               MOVES BPTR CHAR TO LPTR CHAR
                                ;                               DESTROYSE
005615  135                     BCTL:   MOV E,L                 ;LPTR TO E
005616  150                             MOV L,B                 ;BPTR TO L
005617  054                             INR L                   ;BPTR \2
005620  054                             INR L                   ;/***TP
005621  054                             INR L                   ;TO L
005622  176                             MOV A,M                 ;BPTR CHAR TO A
005623  153                             MOV L,E                 ;LPTR TO L
005624  054                             INR L                   ;LPTR \2
005625  054                             INR L                   ;TO L
005626  054                             INR L                   ;/***TP
005627  167                             MOV M,A                 ;STORE BPTR CHAR IN LPTR CHAR
005630  153                             MOV L,E                 ;LPTR TO L
005631  311                             RET
                                ;*****************************************************
                                ;       //// 5 DIGIT FLOATING PT. OUTPUT
                                ;*****************************************************
                                ;       *******ROUTINE TO CONVERT FLOATING PT.
                                ;       ***NUMBERS TO ASCII AND OUTPUT THEM VIA A SUBROUTINE
                                ;       ***CALLED OUTR  -  NOTE:  THIS IS CURRENTLY SET
                                ;       ***TO ODT'S OUTPUT ROUTINE
005632  315 332 011     CVRT:     CALL    ZCHK                  ;CHECK FOR NEW ZERO
005635  302 257 013               JNZ     NNZRO                 ;NOT ZERO
005640  014                       INR     C                     ;IT WAS, OFFSET C BY 2
005641  014                       INR     C
005642  151                       MOV     L,C
005643  315 215 012               CALL    WZER                  ;WRITE ZERO
005646  054                       INR     L                     ;PNT TO DECIMAL EXPONENT
005647  054                       INR     L
005650  054                       INR     L
005651  054                       INR     L
005652  257                       XRA     A                     ;SET IT TO ZERO
005653  167                       MOV     M,A
005654  303 016 014               JMP     MDSKP                 ;OUTPUT IT
005657  126               NNZRO:  MOV     D,M                   ;/GET THE NUMBER TO CONVERT
005660  054                     INR L
005661  106                     MOV B,M
005662  054                     INR L
005663  136                     MOV E,M
005664  054                     INR L                   ;/4 WORD***TP
```

```
005665  176                     MOV A,M         ;/***TP
005666  014                     INR C           ;/OFFSET SCRATCH POINTER BY 2
005667  014                     INR C
005670  151                     MOV L,C         ;/L NOT NEEDED ANY MORE
005671  162                     MOV M,D         ;/SAVE NUMBER IN SCRATCH
005672  054                     INR L
005673  160                     MOV M,B
005674  054                     INR L
005675  163                     MOV M,E         ;/***TP
005676  054                     INR L           ;/***TP
005677  107                     MOV B,A         ;/SAVE COPY OF CHAR & SIGN
005700  346 177                 ANI 177Q        ;GET ONLY CHAR.
005702  167                     MOV M,A         ;/SAVE ABS(NUMBER)
005703  376 100                 CPI 100Q        ;CK FOR ZERO
005705  312 314 013             JZ  NZRO
005710  326 001                 SUI 1           ;/GET SIGN OF DEC. EXP
005712  346 100                 ANI 100Q        ;/GET SIGN OF CHAR.
005714  007             NZRO:   RLC             ;MOVE IT TO SIGN POSITION
005715  054                     INR L           ;/MOVE TO DECIMAL EXP.
005716  167                     MOV M,A         ;/SAVE SIGN OF EXP.
005717  170                     MOV A,B         ;/GET MANT. SIGH BACK
005720  315 220 014             CALL SIGN       ;/OUTPUT SIGN
005723  056 024                 MVI L,(TEN5 AND 377Q)  ;/TRY MULT. OR DIV. BY 100000 FIRST
005725  315 361 014             CALL COPT       ;/MAKE A COPY IN RAM
005730  315 235 014     TST8:   CALL GCHR       ;/GET CHAR. OF NUMBER
005733  107                     MOV B,A         ;/SAVE A COPY
005734  346 100                 ANI 100Q        ;/GET ABSOLUTE VALUE OF CHAR
005736  170                     MOV A,B         ;/INCASE PLUS
005737  312 345 013             JZ  GOTV        ;/ALREADY PLUS
005742  076 200                 MVI A,200Q      ;/MAKE MINUS INTO PLUS
005744  220                     SUB B           ;/PLUS=200B-CHAR
005745  376 022         GOTV:   CPI 22Q         ;/TEST FOR USE OF 100000
005747  372 363 013             JM  TRY1        ;/WONT GO
005752  315 243 014             CALL MORD       ;/WILL GO SO DO IT
005755  306 005                 ADI 5           ;/INCREMENT DEC. EXPONENT BY 5
005757  167                     MOV M,A         ;/UPDATE MEM
005760  303 330 013             JMP TST8        ;/GO TRY AGAIN
005763  056 030         TRY1:   MVI L,(TEN AND 377Q)  ;/NOW USE JUST TEN
005765  315 361 014             CALL COPT       ;/PUT IT IN RAM
005770  315 235 014     TST1:   CALL GCHR       ;/GET CHARACTERISTIC
005773  376 001                 CPI 1           ;/MUST GET IN RANGE 1 TO 6
005775  362 011 014             JP  OK1         ;/ATLEAST ITS 1 OR BIGGER
006000  315 243 014     MDGN:   CALL MORD       ;/MUST MUL OF DIV BY 10
006003  306 001                 ADI 1           ;/INCREMENT DECIMAL EXP.
006005  167                     MOV M,A         ;/UPDATE MEM
006006  303 370 013             JMP TST1        ;/NOW TRY AGAIN
006011  376 007         OK1:    CPI 7           ;/TEST FOR LESS THAN 7
006013  362 000 014             JP  MDGN        ;/NOPE - 7 OR GREATER
006016  151             MDSKP:   MOV    L,C             ;/SET UP DIGIT COUNT
006017  055                     DCR L
006020  055                     DCR L           ;/IN 1ST WORD OF SCRATCH
006021  066 005                 MVI M,5         ;/5 DIGITS
006023  137                     MOV E,A         ;/SAVE CHAR. AS LEFT SHIFT COUNT
006024  315 166 014             CALL LSFT       ;/SHIFT LEFT PROPER NUMBER
006027  376 012                 CPI 12Q         ;/TEST FOR 2 DIGITS HERE
006031  362 311 014             JP  TWOD        ;/JMP IF 2 DIGITS TO OUTPUT
006034  315 072 014             CALL DIGO       ;/OUTPUT FIRST DIGIT
006037  315 115 014     POPD:   CALL MULTT      ;/MULTIPLY THE NUMBER BY 10
006042  315 072 014     INPOP:  CALL DIGO       ;/PRINT DIGIT IN A
006045  302 037 014             JNZ POPD        ;/MORE DIGITS?
006050  076 305                 MVI A,305Q      ;/NO SO PRINT E
006052  315 375 017             CALL OUTR       ;/BASIC CALL TO OUTPUT
006055  315 275 014             CALL GETEX      ;/GET DECIMAL EXP
006060  107                     MOV B,A         ;/SAVE A COPY
006061  315 220 014             CALL SIGN       ;/OUTPUT SIGN
006064  170                     MOV A,B         ;/GET EXP BACK
006065  346 077                 ANI 77Q         ;/GET GOOD BITS
006067  315 340 014             CALL CTWO       ;/GO CONVERT 2 DIGITS
006072  306 260         DIGO:   ADI 260Q        ;/MAKE A INTO ASCII
006074  315 375 017             CALL OUTR       ;/OUTPUT DIGIT
006077  151                     MOV L,C         ;/GET DIGIT COUNT
006100  055                     DCR L           ;/BACK UP TO DIGIT COUNT
006101  055                     DCR L
006102  176                     MOV A,M         ;/TEST FOR DECIMAL PT
006103  376 005                 CPI 5           ;/PRINT . AFTER 1ST DIGIT
006105  076 256                 MVI A,256Q      ;/JUST IN CASE
006107  314 375 017             CZ  OUTR        ;/OUTPUT . IF 1ST DIGIT
006112  126                     MOV D,M         ;/NOW DECREMENT DIGIT COUNT
006113  025                     DCR D
006114  162                     MOV M,D         ;/UPDATE MEM AND LEAVE FLOPS SET
006115  311                     RET             ;/SERVES AS TERM FOR DIGO & CVRT
006116  036 001         MULTT:  MVI E,1         ;/MULT. BY 10 (START WITH X2)
006120  315 166 014             CALL LSFT       ;/LEFT SHIFT 1 = X2
006123  151                     MOV L,C         ;/SAVE X2 IN "RESULT"
006124  055                     DCR L           ;/SET TO TOP OF NUMBER
006125  171                     MOV A,C         ;/SET C TO RESULT
006126  306 011                 ADI 11Q
006130  117                     MOV C,A         ;/NOW C SET RIGHT
006131  174                     MOV A,H         ;/SHOW RAM TO RAM TRANSFER
006132  315 377 014             CALL COPY       ;/SAVE X2 FINALLY
006135  171                     MOV A,C         ;/MUST RESET C
006136  326 011                 SUI 11Q         ;/BACK TO NORMAL
006140  117                     MOV C,A
```

```
006141  036 002             MVI E,2         ;/NOW GET (X2)X4=X8
006143  151                 MOV L,C         ;/BUT MUST SAVE OVERFLOW
006144  055                 DCR L
006145  315 172 014         CALL TLP2       ;/GET X8
006150  151                 MOV L,C         ;/SET UP TO CALL DADD
006151  171                 MOV A,C         ;/SET B TO X2
006152  306 012             ADI 12Q         ;/TO X2
006154  107                 MOV B,A
006155  315 006 012         CALL DADD       ;/ADD TWO LOW WORDS
006160  055                 DCR L           ;/BACK UP TO OVERFLOW
006161  176                 MOV A,M         ;/GET IT
006162  150                 MOV L,B         ;/NOW SET TO X2 OVERFLOW
006163  055                 DCR L           ;/ITS AT B-1
006164  216                 ADC M           ;/ADD WITH CARRY - CARRY WAS PRESERVED
006165  311                 RET             ;/ALL DONE, RETURN OVERFLOW IN A
006166  151         LSFT:   MOV L,C         ;/SET PTR FOR LEFT SHIFT OF NUMBER
006167  055                 DCR L           ;/BACK UP TO OVERFLOW
006170  257                 XRA A           ;/OVERFLOW=0 1ST TIME
006171  167         TLOOP:  MOV M,A         ;/SAVE OVERFLOW
006172  035         TLP2:   DCR E           ;/TEST FOR DONE
006173  370                 RM              ;/DONE WHEN E MINUS
006174  054                 INR L           ;/MOVE TO LOW
006175  054                 INR L
006176  054                 INR L           ;/***TP EXTENSION
006177  176                 MOV A,M         ;/SHIFT LEFT 4 BYTES
006200  027                 RAL
006201  167                 MOV M,A         ;/PUT BACK
006202  055                 DCR L           ;/***TP - ALL DONE
006203  176                 MOV A,M         ;/GET LOW
006204  027                 RAL             ;/SHIFT LEFT 1
006205  167                 MOV M,A         ;/RESTORE IT
006206  055                 DCR L           ;/BACK UP TO HIGH
006207  176                 MOV A,M         ;/GET HIGH
006210  027                 RAL             ;/SHIFT IT LEFT WITH CARRY
006211  167                 MOV M,A         ;/PUT IT BACK
006212  055                 DCR L           ;/BACK UP TO OVERFLOW
006213  176                 MOV A,M         ;/GET OVERFLOW
006214  027                 RAL             ;/SHIFT IT LEFT
006215  303 171 014         JMP TLOOP       ;/GO FOR MORE
006220  346 200         SIGN:   ANI 200Q    ;/GET SIGN BIT
006222  076 240             MVI A,240Q      ;/SPACE INSTEAD OF PLUS
006224  312 231 014         JZ  PLSV        ;/TEST FOR +
006227  076 255             MVI A,255Q      ;/NEGATIVE
006231  315 375 017     PLSV:   CALL OUTR   ;/OUTPUT SIGN
006234  311                 RET
006235  151         GCHR:   MOV L,C         ;/GET CHARCTERISTIC
006236  054         GETA:   INR L           ;/MOVE TO IT
006237  054                 INR L
006240  054                 INR L           ;/***TP
006241  176                 MOV A,M         ;/FETCH INTO A
006242  311                 RET             ;/DONE
006243  315 276 014     MORD:   CALL GETEX  ;/MUL OR DIV DEPENDING ON EXP
006246  137                 MOV E,A         ;/SAVE DECIMAL EXP
006247  105                 MOV B,L         ;/SET UP TO MULT OR DIV
006250  004                 INR B           ;/NOW BOP POINTER SET
006251  151                 MOV L,C         ;/L POINTS TO NUMBER TO CONVERT
006252  171                 MOV A,C         ;/POINT C AT "RESULT" AREA
006253  306 011             ADI 11Q         ;/IN SCRATCH
006255  117                 MOV C,A         ;/NOW C SET RIGHT
006256  173                 MOV A,E         ;/NOW TEST FOR MUL
006257  346 200             ANI 200Q        ;/TEST NEGATIVE DEC. EXP.
006261  312 303 014         JZ  DIVIT       ;/IF EXP IS + THEN DIVIDE
006264  315 137 011         CALL LMUL       ;/MULT.
006267  171         FINUP:  MOV A,C         ;/SAVE LOC. OF RESULT
006270  115                 MOV C,L         ;/C=LOC OF NUMBER (IT WAS DESTROYED)
006271  157                 MOV L,A         ;/SET L TO LOC. OF RESUTL
006272  174                 MOV A,H         ;/SHOW RAM TO RAM TRANSFER
006273  315 377 014         CALL COPY       ;/MOVE RESULT TO NUMBER
006276  151         GETEX:  MOV L,C         ;/NOW GET DECIMAL EXP
006277  054                 INR L
006300  303 236 014         JMP GETA        ;/USE PART OF GCHR
006303  315 000 010     DIVIT:  CALL LDIV   ;/DIVIDE
006306  303 267 014         JMP FINUP
006311  315 340 014     TWOD:   CALL CTWO   ;/CONVERT TO 2 DIGITS
006314  107                 MOV B,A         ;/SAVE ONES DIGIT
006315  315 276 014         CALL GETEX      ;/GET DECIMAL EXP
006320  137                 MOV E,A         ;/SAVE A COPY
006321  346 200             ANI 200Q        ;/TEST FOR NEGATIVE
006323  312 334 014         JZ  ADD1        ;/BUMP EXP BY 1 SINCE 2 DIGITS
006326  035                 DCR E           ;/DECREMENT NEGATIVE EXP SINCE 2 DIGITS
006327  163         FINIT:  MOV M,E         ;/RESTORE EXP WITH NEW VALUE
006330  170                 MOV A,B         ;/NOW DO 2ND DIGIT
006331  303 042 014         JMP INPOP       ;/GO OUT 2ND AND REST FO DIGITS
006334  034         ADD1:   INR E           ;/COMPENSATE FOR 2 DIGITS
006335  303 327 014         JMP FINIT
006340  036 377         CTWO:   MVI E,377Q  ;/CONVERT 2 DIGIT BIN TO BCD
006342  034         LOOP:   INR E           ;/ADD UP TENS DIGIT
006343  326 012             SUI 12Q         ;/SUBTRACT 10
006345  362 342 014         JP  LOOP        ;/TIIL NEGATIVE RESULT
006350  306 012             ADI 12Q         ;/RESTORE ONES DIGIT
006352  107                 MOV B,A         ;/SAVE ONES DIGIT
006353  173                 MOV A,E         ;/GET TENS DIGIT
006354  315 072 014         CALL DIGO       ;/OUTPUT IT
```

```
006357  170                     MOV A,B             ;/SET A TO 2ND DIGIT
006360  311                     RET
006361  171             COPT:   MOV A,C             ;/COPY FROM 10^N TO RAM
006362  306 005                 ADI 5
006364  117                     MOV C,A             ;/SET C TO PLACE TO PUT
006365  076 015                 MVI A,(TEN5/256).
006367  315 377 014             CALL COPY           ;/COPY IT
006372  171                     MOV A,C             ;/NOW RESET C
006373  326 005                 SUI 5
006375  117                     MOV C,A             ;/ITS RESET
006376  311                     RET
006377  104             COPY:   MOV B,H             ;/SAVE RAM H
006400  147                     MOV H,A             ;/SET TO SOURCE H
006401  176                     MOV A,M             ;/GET 4 WORDS INTO THE REGS.
006402  054                     INR L
006403  126                     MOV D,M
006404  054                     INR L
006405  136                     MOV E,M
006406  054                     INR L
006407  156                     MOV L,M             ;/LAST ONE ERASES L
006410  140                     MOV H,B             ;/SET TO DESTINATION RAM
006411  105                     MOV B,L             ;/SAVE 4TH WORD IN B
006412  151                     MOV L,C             ;/SET TO DESTINATION
006413  167                     MOV M,A             ;/SAVE FIRST WORD
006414  054                     INR L
006415  176                     MOV A,M             ;/SAVE THIS WORD IN A(INPUT SAVES C HERE
006416  162                     MOV M,D             ;/NOW PUT 2ND WORD
006417  054                     INR L
006420  163                     MOV M,E
006421  054                     INR L
006422  160                     MOV M,B             ;/ALL 4  COPIED NOW
006423  311                     RET                 ;/ALL DONE

006424  303 120 000 021 TEN5:  DB 303Q,120Q,0Q,21Q  ;/303240(8) = 100000.
006430  240 000 000 004 TEN:   DB 240Q,0Q,0Q,4Q    ;/12(8) = 10

                        SCRATCH MAP FOR I/O CONVERSION ROUTINES

                        RELATIVE TO (C+2)USE
                        C-2             DIGIT COUNT
                        C-1             OVERFLOW
                        C               HIGH NUMBER - MANTISSA
                        C+1             LOW NUMBER
                        C+2             CHARACTERISTIC
                        C+3             DECIMAL EXPONEXT (SIGN & MAG.)
                        C+4             TEN**N
                        C+5             TEN**N
                        C+6             TEN**N
                        C+7             RESULT OF MULT & DIV
                        C+8             AND TEMP FOR X2
                        C+9             "
                        C+10            L FOR NUMBER TO GO INTO (INPUT ONLY)
                        C+11            DIGIT JUST INPUT (INPUT ONLY)

                                        /*****BEGIN INPUT*************

006434  067             ERR:    STC                 ;ERROR FLAG
006435  311                     RET                 ;AND RETURN

                        ;*****************************************************
                        ;       //// 4 1/2 DIGIT INPUT ROUTINE
                        ;*****************************************************

                                                    /L POINTS TO WHERE TO PUT INPUT NUMBER
                                                    /C POINTS TO 13(10) WORDS OF SCRATCH

006436  105             INPUT:  MOV B,L             ;/SAVE ADDRESS WHERE DATA IS TO GO
006437  171                     MOV A,C             ;/IN SCRATCH
006440  306 017                 ADI 17Q             ;/COMPUTE LOC. IN SCRATCH
006442  157                     MOV L,A
006443  160                     MOV M,B             ;/PUT IT
006444  014                     INR C               ;/OFFSET SCRATCH POINTER
006445  014                     INR C               ;/BY 2
006446  315 055 013             CALL ZROIT          ;/ZERO NUMBER
006451  054                     INR L               ;/AND ZERO
006452  167                     MOV M,A             ;/DECIMAL EXPONENT
006453  315 311 015             CALL GNUM           ;/GET INTEGER PART OF NUM
006456  376 376                 CPI 376Q            ;/TERM=.?
006460  312 203 013             JZ  DECPT           ;/YES
006463  376 025         TSTEX:  CPI 25Q             ;/TEST FOR E
006465  312 230 015             JZ  INEXP           ;/YES - HANDLE EXP
006470  376 360                 CPI 360Q            ;/TEST FOR SPACE TERM (240B-260B)
006472  302 034 013             JNZ ERR             ;/NOT LEGAL TERM
006475  315 022 013             CALL FLTSGN         ;/FLOAT # AND SIGN IT
006500  315 276 014     SCALE:  CALL GETEX          ;/GET DECIMAL EXP
006503  346 177                 ANI 177Q            ;/GET GOOD BITS
006505  137                     MOV E,A             ;/SAVE COPY
006506  346 100                 ANI 100Q            ;/GET SIGN OF EXP
006510  007                     RLC                 ;/INTO SIGN BIT
```

```
006511  267                     ORA A           ;/SET FLOPS
006512  107                     MOV B,A         ;/SAVE SIGN
006513  173                     MOV A,E         ;/GET EXP BACK
006514  312 122 015            JZ  APLS        ;/JMP IS +
006517  076 200                 MVI A,200Q      ;/MAKE MINUS +
006521  223                     SUB E           ;/NOW ITS +
006522  200             APLS:   ADD B           ;/SIGN NUMBER
006523  167                     MOV M,A         ;/SAVE EXP (SIGN & MAG.)
006524  056 024                 MVI L,(TEN5 AND 377Q)  ;/TRY MORD WITH 10**5 FIRST
006526  315 361 014             CALL COPT       ;/TRANSFER TO RAM
006531  315 276 014             CALL GETEX      ;/GET DECIMAL EXP
006534  346 077         INT5:   ANI 77Q         ;/GET MAG. OF EXP
006536  376 005                 CPI 5Q          ;/TEST FOR USE OF 10**5
006540  372 154 015             JM  TRYTN       ;/WONT GO - TRY 10
006543  315 243 014             CALL MORD       ;/WILL GO SO DO IT
006546  326 005                 SUI 5Q          ;/MAG = MAG -5
006550  167                     MOV M,A         ;/UPDATE DEC. EXP IN MEM
006551  303 134 015             JMP INT5        ;/GO TRY AGAIN
006554  056 030         TRYTN:  MVI L,(TEN AND 377Q)  ;/PUT TEN IN RAM
006556  315 361 014             CALL COPT
006561  315 276 014             CALL GETEX      ;/SET UP FOR LOOP
006564  346 077         INT1:   ANI 77Q         ;/GET MAGNITUDE
006566  267                     ORA A           ;/TEST FOR 0
006567  312 026 016             JZ  SAVEN       ;/DONE, MOVE NUM OUT AND GET OUT
006572  315 243 014             CALL MORD       ;/NOT DONE - DO 10
006575  326 001                 SUI 1Q          ;/EXP = EXP -1
006577  167                     MOV M,A         ;/UPDATE MEM
006600  303 164 015             JMP INT1        ;/TRY AGAIN
006603  151             DECPT:  MOV L,C         ;/ZERO DIGIT COUNT
006604  055                     DCR L           ;/SINCE ITS NECESSARY
006605  055                     DCR L           ;/TO COMPUTE EXP.
006606  066 000                 MVI M,0         ;/ZEROED
006610  315 014 016             CALL EP1        ;/GNUM IN MIDDLE
006613  137                     MOV E,A         ;/SAVE TERMINATOR
006614  151                     MOV L,C         ;/MOVE DIGIT COUNT TO EXP
006615  055                     DCR L           ;/BACK UP TO DIGIT COUNT
006616  055                     DCR L
006617  106                     MOV B,M         ;/GOT DIGIT COUNT
006620  315 276 014             CALL GETEX      ;/SET L TO DEC. EXP
006623  160                     MOV M,B         ;/PUT EXP
006624  173                     MOV A,E         ;/TERM BACK TO A
006625  303 063 015             JMP TSTEX       ;/TEST FOR E+OR-XX
006630  315 022 015     INEXP:  CALL FLTSGN     ;/FLOAT AND SIGN NUMBER
006633  315 026 016             CALL SAVEN      ;/SAVE NUMBER IN (L) TEMP
006636  315 055 015             CALL ZROIT      ;/ZERO OUT NUM. FOR INPUTTING EXP
006641  315 311 015             CALL GNUM       ;/NOW INPUT EXPONENT
006644  376 360                 CPI 360Q        ;/TEST FOR SPACE TERM.
006646  302 034 015             JNZ ERR         ;/NOT LEGAL - TRY AGAIN
006651  151                     MOV L,C         ;/GET EXP OUT OF MEM
006652  054                     INR L           ;/***TP
006653  054                     INR L           ;/EXP LIMITED TO 5 BITS
006654  176                     MOV A,M         ;/GET LOWEST 8 BITS
006655  346 037                 ANI 37Q         ;/GET GOOD BITS
006657  107                     MOV B,A         ;/SAVE THEM
006660  054                     INR L           ;/GET SIGN OF EXP
006661  176                     MOV A,M         ;/INTO A
006662  267                     ORA A           ;/SET FLOPS
006663  170                     MOV A,B         ;/INCASE NOTHING TO DO
006664  372 272 015             JM  USEIT       ;/IF NEG. USE AS +
006667  076 000                 MVI A,0Q        ;/IF + MAKE -
006671  220                     SUB B           ;/0-X = -X
006672  054             USEIT:  INR L           ;/POINT AT EXP
006673  206                     ADD M           ;/GET REAL DEC. EXP
006674  167                     MOV M,A         ;/PUT IN MEM
006675  171                     MOV A,C         ;/NOW GET NUMBER BACK
006676  306 015                 ADI 15Q         ;/GET ADD OF L
006700  157                     MOV L,A         ;/L POINTS TO L OF NUMBER
006701  156                     MOV L,M         ;/NOW L POINTS TO NUMBER
006702  174                     MOV A,H         ;/RAM TO RAM COPY
006703  315 377 014             CALL COPY       ;/COPY IT BACK
006706  303 100 015             JMP SCALE       ;/NOW ADJUST FOR EXP
006711  315 372 017     GNUM:   CALL INP        ;/GET A CHAR
006714  376 240                 CPI 240Q        ;/IGNORE LEADING SPACES
006716  312 311 015             JZ  GNUM
006721  376 255                 CPI 255Q        ;/TEST FOR -
006723  302 337 015             JNZ TRYP        ;/NOT MINUS
006726  151                     MOV L,C         ;/MINUS SO SET SIGN
006727  054                     INR L           ;/IN CHAR LOC.
006730  054                     INR L           ;/***TP
006731  054                     INR L
006732  066 200                 MVI M,200Q      ;/SET - SIGN
006734  303 311 015             JMP GNUM
006737  376 253         TRYP:   CPI 253Q        ;/IGNORE +
006741  312 311 015             JZ  GNUM
006744  326 260         TSTN:   SUI 260Q        ;/STRIP ASCII
006746  370                     RM              ;/RETURN IF TERM
006747  376 012                 CPI 12Q         ;/TEST FOR NUMBER
006751  360                     RP              ;/ILLEGAL
006752  137                     MOV E,A         ;/SAVE DIGIT
006753  315 047 016             CALL GETN       ;/LOC. OF DIGIT STORAGE TO L
006756  163                     MOV M,E         ;/SAVE DIGIT
006757  315 116 014             CALL MULTT      ;/MULT NUMBER BY 10
006762  267                     ORA A           ;/TEST FOR TOO MANY DIGITS
```

```
006763  300                     RNZ                 ;/TOO MANY DIGITS
006764  315 047 016             CALL GETN           ;/GET DIGIT
006767  151                     MOV L,C             ;/SET L TO NUMBER
006770  054                     INR L
006771  054                     INR L               ;/***TP
006772  206                     ADD M               ;/ADD IN THE DIGIT
006773  167                     MOV M,A             ;/PUT RESULT BACK
006774  055                     DCR L               ;/NOW DO HIGH
006775  176                     MOV A,M             ;/GET HIGH TO ADD IN CARRY
006776  316 000                 ACI 0Q              ;/ADD IN CARRY
007000  167                     MOV M,A             ;/UPDATE HIGH
007001  055                     DCR L               ;/***TP EXTENSION
007002  176                     MOV A,M
007003  316 000                 ACI 0Q              ;/ADD IN CARRY
007005  167                     MOV M,A             ;/***TP ALL DONE
007006  330                     RC                  ;/OVERFLOW ERROR
007007  055                     DCR L               ;/BUMP DIGIT COUNT NOW
007010  055                     DCR L
007011  106                     MOV B,M             ;/GET DIGIT COUNT
007012  004                     INR B               ;/BUMP DIGIT COUNT
007013  160                     MOV M,B             ;/UPDATE DIGIT COUNT
007014  315 372 017     EP1:    CALL INP            ;/GET NEXT CHAR
007017  303 344 015             JMP TSTN            ;/MUST BE NUM. OR TERM
007022  151             FLTSGN: MOV     L,C           ;POINT L AT NUMBER TO FLOAT
007023  303 325 011             JMP     FLOAT       ;GO FLOAT IT
007026  171              SAVEN: MOV A,C             ;/PUT NUMBER IN (L)
007027  306 015                 ADI 15Q             ;/GET ADD OF L
007031  157                     MOV L,A
007032  136                     MOV E,M             ;/GET L OF RESULT
007033  153                     MOV L,E             ;/POINT L AT (L)
007034  054                     INR L               ;/SET TO 2ND WORD TO SAVE C
007035  161                     MOV M,C             ;/SAVE C IN (L) +1 SINCE IT WILL BE
007036  151                     MOV L,C             ;/SET UP TO CALL COPY          DESTROYED
007037  113                     MOV C,E             ;/NOW L&C SET
007040  174                     MOV A,H             ;/RAM TO RAM COPY
007041  315 377 014             CALL COPY           ;/COPY TO L
007044  117                     MOV C,A             ;/(L)+1 RETURNED HERE SO SET AS C
007045  267                      ORA    A                ;MAKE SURE CY=0 (NO ERROR)
007046  311                     RET                 ;/NOW EVERYTHING HUNKY-DORRY
007047  171              GETN:  MOV A,C             ;/GET DIGIT
007050  306 016                 ADI 16Q             ;/LAST LOC. IN SCRATCH
007052  157                     MOV L,A             ;/PUT IN L
007053  176                     MOV A,M             ;/GET DIGIT
007054  311                     RET
007055  151              ZROIT: MOV L,C             ;/ZERO NUMBER
007056  257                     XRA A
007057  167                     MOV M,A             ;/***TP
007060  054                     INR L               ;/***TP
007061  167                     MOV M,A
007062  054                     INR L
007063  167                     MOV M,A
007064  054                     INR L               ;/NOW SET SIGN TO +
007065  167                     MOV M,A
007066  311                     RET                 ;/DONE
000333                  READ      EQU     333Q              ;ODT READ ROUTINE
                        ; CONTAIN LOW BYTE OF TWO BYTE VALUE. RETURNS CY=1 IF
                        ; BC>DE, CY=0 IF BC<DE: Z=1 IF BC=DE.
007067  173             DCOMP:    MOV     A,E
007070  271                       CMP     C
007071  300                       RNZ
007072  172                       MOV     A,D
007073  270                       CMP     B
007074  311                       RET
                        ; ROUTINE TO INPUT CHAR FROM TTY
007075  305             CHAR2:    PUSH    B
007076  315 333 000               CALL    READ              ;INPUT FROM ODT
007101  170                       MOV     A,B               ;GET CHAR TO A REG.
007102  301                       POP     B                 ;RESTORE B,C
007103  311                       RET
                        ; ROUTINE TO ADJUST VALUES OF BIN, FORWARD PNT. AND
                        ; LINE LENGTH OF SOURCE LINE.  PASSED ADD OF TEMP VARIABLE
                        ; CONTAINING ADD OF SOURCE LINE.
007104  365             PTVAL:    PUSH    PSW
007105  325                       PUSH    D
007106  345                       PUSH    H
007107  076 002                   MVI     A,002
007111  136                       MOV     E,M
007112  054                       INR     L
007113  126                       MOV     D,M
007114  054                       INR     L
007115  325                       PUSH    D
007116  343             N1:       XTHL
007117  136                       MOV     E,M
007120  043                       INX     H
007121  126                       MOV     D,M
007122  043                       INX     H
007123  343                       XTHL
007124  163                       MOV     M,E
007125  054                       INR     L
007126  162                       MOV     M,D
007127  054                       INR     L
007130  075                       DCR     A
007131  302 116 016               JNZ     N1
```

```
007134  343                     XTHL
007135  126                     MOV     D,M
007136  341                     POP     H
007137  162                     MOV     M,D
007140  341                     POP     H
007141  321                     POP     D
007142  361                     POP     PSW
007143  311                     RET
                        ; ROUTINE TO CHK FLAGS ON INPUT AND OUTPUT.
                        ; PASSED FLAG VALUE IN REG B.
007144  365             MCHK:   PUSH    PSW
007145  333 003        MCHK1:   IN      3
007147  240                     ANA     B
007150  312 145 016             JZ      MCHK1
007153  361                     POP     PSW
007154  311                     RET
                        ; MULTIPLICATION ROUTINE (ADD. VALUES)
007155  136             MULT:   MOV     E,M
007156  053                     DCX     H
007157  126                     MOV     D,M
007160  066 021                 MVI     M,11H
007162  006 000                 MVI     B,0
007164  110                     MOV     C,B
007165  173             TOP:    MOV     A,E
007166  037                     RAR
007167  137                     MOV     E,A
007170  172                     MOV     A,D
007171  037                     RAR
007172  065                     DCR     M
007173  127                     MOV     D,A
007174  310                     RZ
007175  322 212 016             JNC     SHIFT
007200  053                     DCX     H
007201  053                     DCX     H
007202  170                     MOV     A,B
007203  206                     ADD     M
007204  107                     MOV     B,A
007205  043                     INX     H
007206  171                     MOV     A,C
007207  216                     ADC     M
007210  117                     MOV     C,A
007211  043                     INX     H
007212  171             SHIFT:  MOV     A,C
007213  037                     RAR
007214  117                     MOV     C,A
007215  170                     MOV     A,B
007216  037                     RAR
007217  107                     MOV     B,A
007220  303 165 016             JMP     TOP
                        ;LINKAGES TO FLOATING POINT ROUTINES
007707                          ORG     7707Q
007707  303 255 011             JMP     NORM
007712  303 325 011             JMP     FLOAT
007715  303 215 012             JMP     WZER
007720  303 134 010             JMP     LADD
007723  303 137 011             JMP     LMUL
007726  303 000 010             JMP     LDIV
007731  303 140 010             JMP     LSUB
007734  303 316 011             JMP     DFXL
007737  303 127 011             JMP     LMCM
007742  303 377 014             JMP     COPY
007745  303 232 013             JMP     CVRT
007750  303 036 015             JMP     INPUT
007753  303 155 016             JMP     MULT
007756  303 104 016             JMP     PTVAL
007761  303 067 016             JMP     DCOMP
007764  303 144 016             JMP     MCHK
007767  303 075 016             JMP     CHAR2
007772  303 206 031             JMP     INL
007775  303 311 027             JMP     OUTL
                                END
```

BASIC INTERPRETER SYMBOL TABLE

A	000007		ADDD	015154		ADFLD	015450		AFOND	014445	
AGA	015030		ALPHA	011042		AR	013024		ARCK	014317	
ARYES	014343		ASBC	015157		B	000000		BAC	011040	
BINOP	015121		BMPTR	014721		BND1	012554	*	BND2	012643	
BND3	012646		BOTNS	030252		BOUND	012515		BUPT	014246	*
C	000001		C1	011031		CALLP	016560		CHOVL	017673	
CHAR2	007767		CHARS	012410		CHEKE	014707		CHK1	011243	
CHKLC	017660		CHKMD	014635		CHPAR	003712		CKDIM	017701	
CLSUB	016742		CONT	013614		CONV	007745		COPD1	013677	
COPDH	013671		CPNT	030133		CREG	030204		CVB	011052	
CVB1	011113		CVB2	011150		D	000002		DCOMP	007761	
DEXL	007734		DIM	013456		DIV	015243		DLOOP	013464	
DOL	015002		E	000003		ECAV	014747		ENDD	013400	
ENTRY	013164		EOK	015020	*	ER10	016216		ER14	016271	
ER15	017622		ER17	017627		ER18	017634		ER19	017641	
ER20	017646		ER21	017653		ER6	013472		ER7	015776	
ER8	015070		ER9	014175		ERLN	012737		ERRET	015736	
ERROR	012576		ERRR1	012722		EVAL	014735		EXPRE	015354	
F1	012116		FALSE	016450		FBAC	013171		FBILD	017201	
FCOMP	016471		FDAT	014605		FEXST	017243		FINPT	007750	
FIX	014453		FLOFD	015503		FLIMT	030220		FLOAT	007712	*
FMULT	015235		FNDSB	016644		FONE	017353		FOR	016775	
FOR10	012101		FOR11	012100		FOR12	012077		FORM1	012110	
FORM2	012107		FORM3	012106		FORM4	012105		FORM5	012111	
FORM6	012117		FORM7	012104		FORM8	012103		FORM9	012102	
FOUND	011240		FPERR	015161		FREG1	030174		FREG2	030200	
FSYM	013015		FWAM	017720		GET	013752		GKEY	013324	
GOOD	014510		GOSUB	016745		GOTO	013416		GREG	030167	
GSENT	013421		GTRA	013427		H	000004		HLINP	030206	
HOME	014065		HSRIN	012325		IAGA	016243		IBUF	030001	
ICP2	013005		ICP4	013000		ICP7	012766		ICP8	012773	
IDONE	014631		IEND	015642		IFRT	016223		I11	016276	
ILOOP	013232		INCPT	013007		INP	014611	*	INPER	015546	
INPOK	015637		INPUT	015542		INSER	010217		ISR10	010553	
ISR11	010561		ISR12	010572		ISR1A	010242		ISRT1	010275	
ISRT2	010304		ISRT3	010324		ISRT4	010327		ISRT5	010417	
ISRT6	010441		ISRT7	010504		ISRT8	010510		ISRT9	010522	
JTBL	013344		KASE	030140		KDAT1	011517		KDAT2	011544	
KDAT3	011636		KDAT4	011700		KDATA	011513		KFPNT	030126	
KL2	030124		KL4	030126		KL6	030130	*	KLEN	030130	
KLINE	030122		KONT	014114		L	000005		L1	013274	
L2	011213		LADD	007720		LDIV	007726		LEN	030141	
LENGT	011161		LET	015763		LIS1	010752		LIST	010721	
LMCM	007737		LMUL	007723		LOK	016003		LPNT	030122	
LSUB	007731		LUKON	013047		M	000006		M1	010000	
M1A	010023		M2	010026		M3	010037		M4	010102	
M4A	010107		MCHK	007764		MDBC	015246		MEMEN	037777	
MEMFU	012424		MEMST	030000		MESCR	030210		MINSE	014506	
MODE	030205		MODE1	014655		MORE	014225		MULT	007753	
MULT1	030142		MULT2	030144		NDOU	013555		NEST	030224	
NEXT	017357		NEXT1	017423		NFPER	015224		NL2	030115	
NL4	030117	*	NL5	030121		NLE1	011165		NLE2	011204	
NLINE	030113		NOENT	013106		NOLIN	010621		NOMAT	013073	
NORM	007707	*	NOSYM	013104		NOT3	016436		NSRCH	011210	
NSTOK	017330	*	NUMB	011024		NUSUB	016623		NXTDN	017612	
NXTSP	030131		OBUFF	030000		ODAT0	012313		ODAT1	012144	
ODAT2	012152		ODAT3	012170		ODAT4	012201		ODAT5	012213	
ODAT6	012221		ODAT7	012240		ODAT8	012267		ODAT9	012302	
ODATA	012182		OKK	014130		OKLET	013477		OKN	013443	
OMUD	015473	*	OUTR	013714	*	P1	011270		P2	011302	
PAD	011255		PARLP	016063		PEND	015534		PIN1	012333	
PIN1A	012336		PIN3	012400		PINST	016211		PK1	012030	
PK2	012065		PL2	030133	*	PL4	030135		PL6	030137	
PLINE	030131		PLOOP	015304		PREXP	016713		PRI	015253	
PRI1	016171		PRMPT	015562		PRTIT	015403		PSW	000006	
PTI	010155		PTAPE	010145		PTFIN	016142		PTVAL	007756	
QCHEK	015336		QOTOK	015323		QSTR1	012060		QSTRG	012053	
QTCHK	011016	*	QUITT	011010		QUOTE	015312		RDKON	014144	
RELAT	016353		RETRN	016773		RINST	014107		RUN	013173	
S2	011407	*	S3	011410		S4	011443		S4A	011442	
S5	011477		SAVV	016110		SBSAV	030135		SC1	014214	
SCALR	014307		SCOLN	015435		SCR	030146		SEND	000006	
SFSG	014252		SINEQ	016521		SLOAD	014300		SONWD	015517	
SORCE	013250		SP	000006		SPACE	014627		STAC	030226	
STLIN	030111		STNER	015741		STOKK	015705		STOKV	015666	
STP	017154		STPNT	010613		STRIN	015650		STSIZ	000024	
STSPA	030115		SUBB	015227		SUBS	017722		SYMSR	011367	
TAPE	010652		THEN	016457		TIN1	011714		TIN2	011766	
TIN4	012010		TIN4A	012013		TIN5	012001		TIN5A	012004	
TOPNS	030226		TRUE	016454		TTYIN	011711		V1	014046	
VALUE	013724		VAR	014203		VARAD	030212		VEND	030252	
VLOC	030216		VNAME	030214		W1	011337		W2	011360	
WFOR	015075		WFPER	015215		WHAT	010636		WRIT	011320	
WRIT1	011622		ZROL	007715							

FLOATING POINT PACKAGE SYMBOL TABLE

A	000007	ABCH	004464	ACPR	005357	ADD1	006334
ADD2	004337	ADDZ	004326	ADRS	005341	APLS	006522
B	000000	BBCH	004475	BCHK	004742	BCTL	005615
BMIN	004442	C	000001	CCHK	005504	CCMP	005133
CFCHE	005114	CHAR2	007075	COM1	004426	COM2	004450
COPT	006361	COPY	006377	CRIN	004115	CSIGN	005551
CSTR	005561	CTWO	006340	CVRT	005632	D	000002
DADD	005006	DCLR	005035	DCMP	005364	DCOMP	007067
DECPT	006600	DECR	005324 *	DFXL	004716	DIGO	006072
DIVIT	006303	DLST	004751	DRST	004770	DSUB	005046
DTST2	004022	E	000003	ENT1	005420	ENT2	005412
EP1	007014	EQ02	004224	EQUL	004224 *	ERR	006434
FINIT	006327	FINUP	006267	FLOAT	004725	FLTSG	007022
FXL1	004062	FXL2	004663	GCHAR	005101	GCHR	006235
GETA	006236	GETEX	006276	GETN	007047	GNUM	006711
GOON	004073	GOTV	005745	H	000004	INCR	005266
INCR2	005306	INCR3	005303	INDF1	005204 *	INDFC	005250
INEXP	006630	INL	014606	INP	007772	INPOP	006042
INPUT	006436	INT1	006564	INT5	006534	INTR	004577
KPGO	004566	L	000005	L000	004505	L001	004510
L002	004437	L003	004513	L128	004516	L129	004521
L131	004524	LADD	004134	LADS	004142	LASD	004402
LDCP	005471	LDIV	004000	LLTB	004177	LMCM	004527
LMCP	005500	LMUL	004537	LOOP	006342	LSFT	006166
LSUB	004140	LXFR	005444	M	000006	MADD	004644
MANT	004357	MAXCH	000077	MCHK	007144	MCHK1	007145
MDGN	006000	MDSKP	006016	MINCH	000300	MORD	006243
MSFH	005571	MULT	007155	MULTT	006116	N1	007116
NOHK	004204	NNZRO	005657	NORM	004655	NORM1	004656 *
NOTO	004252	NZRO	005714	OFLW1	005166	OFLWC	005533
OK1	006011	OUTL	013711	OUTR	007775	OVER	005427
PLSV	006231	POPD	006037	PSW	000006	PTVAL	007104
READ	000333	REP3	004047	REP5	005446	REP6	004671
SAVEN	007026	SCALE	006500	SCCFG	005322	SCHAR	004713
SH10	004214	SHIFT	007212	SIGN	006220	SP	000005
STORC	005523	SUBZ	004271	TEN	006430	TEN5	006424
TLOOP	006171	TLP2	006172	TOP	007165	TRY1	005763
TRYP	006737	TRYTN	006554	TST1	005770	TST8	005730
TSTEX	006463	TSTN	006744	TSTR	005347	TWOD	005311
UFLW1	005150	UFLWC	005542	USEIT	006672	WCHAR	005237
WFLT	003557	WIND	005177	WMANT	005230	WOVR	005161
WUND	005143	WZER	005215	WZERC	005257	ZCHK	004732
ZMCHK	004732	ZROIT	007055				

BOOK REVIEW

Computer Accouting Methods
by G. A. Cook, B. J. Wade and C. C. Upton
Petrocelli Books, 384 Fifth Ave., New York, NY 10018

Reviewed by Rich Roth, 1052 Clark, Mountain View, CA 94040

With an ominous title like this, you'd expect a detailed accountant's handbook to computing. However, what you get is an 184-page paperback comprising the computer part of an 'Introduction to Accounting' course. Being oriented toward a beginning accountant who doesn't know computers, it spends the first 3 chapters on computers and BASIC. And then in rapid succession comes the juicy parts¾ general ledger, financial reports, accounts receivable, payroll accounting, depreciation accounting. Each chapter has flowcharts with explanation, variable definitions, operating instructions and BASIC SOURCE code for the topic. The BASIC used in standard Dartmough BASIC, with character string arrays and 'PRINT with IMAGE' added. There is a fair sized database, and the class exercises were set-up for a time-sharing service (probably the ex-IBM Service Bureau Co.rp
Bureau Corp., since two of the authors work for SBC). This allows the programs to use disk files, and to do all activity in memory.

For anyone familiar with BASIC, this book is an excellent starting point to do accounting programming, with the possible addition of a review of a regular introductory accounting text. The only major missing topic is calculation errors. Most mini/microcomputer BASIC's use 32-bit floating point numbers for about 6½ decimal digits precision. For business purposes that limits you to $29,999.99. **[O'course, you could always modify your BASIC to do BCD arithmetic with unlimited precision and *no* round-off . . . as is more appropriate to accounting applications than is floating-point, anyway. — Editor]**

NEW BOOK FOR COMPUTER HOBBYISTS

Howard W. Sams & Co. (4300 W. 62nd St., Indianapolis, IN 46206) has just published a book for the computer hobbyist: *How to Buy & Use Minicomputers & Microcomputers*, by William Barden, Jr. I am sure that all hobbyists will agree that such a book is long overdue. Newcomers to computing as a hobby have had to rely on other hobbyists, periodicals, and manufacturers' literature for their orientation . This book presents a great deal of useful information all in one place, and the hobbyist will surely get his or her $9.95 worth from it.

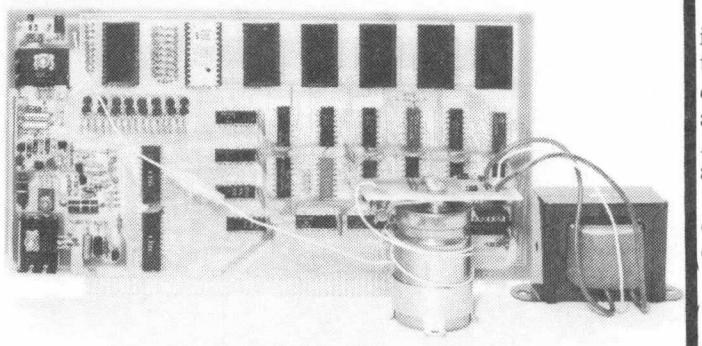
Like most introductory books on computing, this one is written backwards. That is, the basic approach is "bottom-up" instead of "top-down," and the reader must wade through the usual discussions of binary arithmetic, etc. before getting to the good stuff. *C'est la vie.* Chapter 1 is a short introduction. Chapter 2 is devoted to binary arithmetic. Chapter 3 deals with basic ALU operations. Chapter 4 discussed programming languages. Chapter 5 covers peripherals. Chapter 6 explains benchmarking. Chapter 7 is about programming methods. Chapter 8 compares the various microcomputers, and the final chapter compares minicomputers. Ten appendices present useful reference data.

On the whole it's a good book, but one might prefer less space devoted to instruction sets and more devoted to the relative merits of Altair versus IMSAI, etc. Coverage of peripheral boards such as video displays, cassette controllers, etc. is rather skimpy, to say the least, and information on hardware construction and troubleshooting is conspicuous by its absence. Nevertheless, this is the world's greatest book on computing as a hobby. It also seems to be the only one, at present.

Nov. 1, 1976 Jim Day

PRAMMER FROM XYBEK

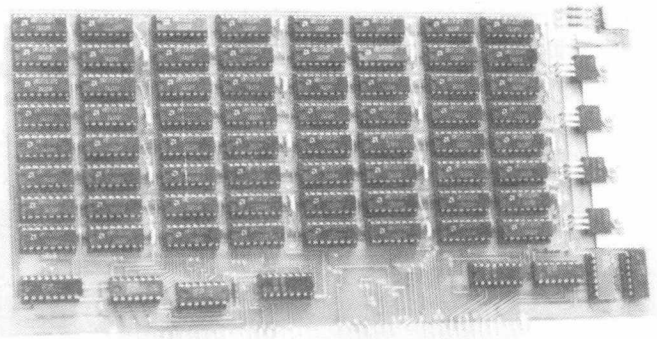
A PROM/RAM/PROM programmer for the Altair 8800, IMSAI 8080 and other S100 bus microcomputers is now available. This 2K memory board contains 256 bytes of RAM and space of 1792 bytes of 1702A EPROM. One of the 1702A sockets doubles as a 1702A programmer. The PRAMMER is not an I/O device, but occupies any 2K slice of system memory. This kit is complete with its own 80v power supply, features onboard timing independent of the CPU clocks and contains its own microprogram for read and write control. No oneshots are used. The 256 bytes of RAM may be used for a stack, for buffers, save areas, etc,, eliminating the need for use of main memory already dedicated to other application programs. Complete stand-alone software for programming and copying 1702A EPROMs is supplied with the PRAMMER kit in a single preprogrammed 1702A. Also included are the complete listings for PRAMSYS, an eleven-function development system designed to reside in the 1792 bytes of EPROM in the fully populated board and to interface with a TTY compatible terminal. Also available is a 3-foot extention kit for bringing any of the 1702A sockets to a Textools zero insertion force external to your system. The introductory price for the PRAMMER kit is $189 and the extension kit is $15. Address inquiries to XYBEK, P.O. Box 1631, Cupertino, CA 95104, (408) 243-0834.



8K S-100 STATIC RAM BOARD KIT FOR $239

Compatible with Altair and Imsai Mothers, this pre-tested 8K static memory board has buffered address and data out lines, memory protect, no wait states, access time of 450 ns, low power consumption of only 1.3 amps, output disable feature, and a DIP switch, located on the upper edge of the board which allows address change without removing the board from the computer. The printed circuit board is of aerospace quality and solder masked on both sides. The 8K baby provides the most desired features in a high quality, low power product with a low price.

Contact VECTOR GRAPHIC, INC., P.O. Box 4784, Thousand Oaks, California 91359, (213) 889-9809.



16 KILOBYTE DYNAMIC RAM KIT FOR $650

WWW Enterprises (Box 548, Harbor City, CA 90710, 213/835-9417) sent us an announcement of their 16Kx8 dynamic RAM units–$550 (kit) or $650 (assembled & tested). It sez they have a 270ns access, 470 ns R/W, are fully socketed, and are Sphere-compatible.

NEW BIPOLAR TTL PROM FROM SIGNETICS FEATURES 40 NANOSECOND ACCESS, REDUCED CHIP SIZE

A field programmable 4096-bit bipolar TTL Read Only Memory with a typical access time of 40 nanoseconds has been introduced by Signetics.

Organized as 1024 4-bit words, the device is one of the fastest 1K X 4 PROMs on the market and also has a significantly smaller chip size of 18,000 square mils, about 30% smaller than current industry standards.

The 1K X 4 PROM is being produced in a space-saving 18-pin ceramic DIP package and is available with either Open Collector outputs (82S136) or Tri-State outputs (82S137) and in both commercial and military temperature ranges.

The 82S136/137 devices are an extension of the Signetics family of PROMs, utilizing the same basic fusible-link programming techniques. They also represent an extension in the firm's line of PROMs with 4-bit word organization, which facilitates system expansion.

The improved speed of these new PROMs makes them ideal for any application requiring a large ROM, according to Ralph Kaplan, Marketing Manager for Memories. The commercial version (N82S136/137) has a specified maximum address access time of 60 nanoseconds, with 40 ns typical. Access time of the military version (S82S136/137) is rated at 80 ns with 60 ns typical.

The devices, which are fully TTL compatible, include on-chip decoding and two chip enable inputs for ease of memory expansion.

Other key specifications include: typical power dissipation of 0.15 milliwatts/bit and input loading of –150 microamperes maximum (military) and –100 microamperes maximum (commercial).

The N82S135/137 devices are priced at $15.50 in quantities of 100 or more. This represents a 35% savings in cost/bit compared with 4K bits of 256 X 4 PROMs now on the market, Kaplan says.

In addition, Kaplan noted that further production economies can result from the reduced package count and board costs made possible by the smaller size of the device.

For further information, contact Signetics, 811 E. Arques Ave., Sunnyvale, CA 94086, (408) 739-7700.

TOSHIBA MARKETS NONVOLATILE RAM AND FIRST JAPANESE MICRO

Toshiba is now offering a 1K-bit static RAM that is nonvolatile, that is, when the power goes away, the data doesn't (at least, not for a year or more). Its price is comparable to that of core memory. It is TTL-compatible, has a maximum write-time of 1 mics, a maximum access-time of 1.5 mics. and draws 600 mw or less.

Toshiba has also introduced Japan's first home-grown microprocessor, the T3444. It comes in a 42-pin pack that includes ALU, 16x8 RAM, 256x24 ROM, and I/O ports. It has only 14 instructions, but runs much faster than many of the U.S. micros (a 1.25 mics. cycle time).

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 12 February 1977 Volume 2, Issue 2

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

STUFF

Free Software? Or – Support Your Local Software Vendor, Tom Pittman 3
Symple Cyphering, David Gruhn 9
Dr. Dobb's Apologizes 10
Game Solutions & Notes on Segmented Memory, Ed Comer 11

CONSUMER NOTES

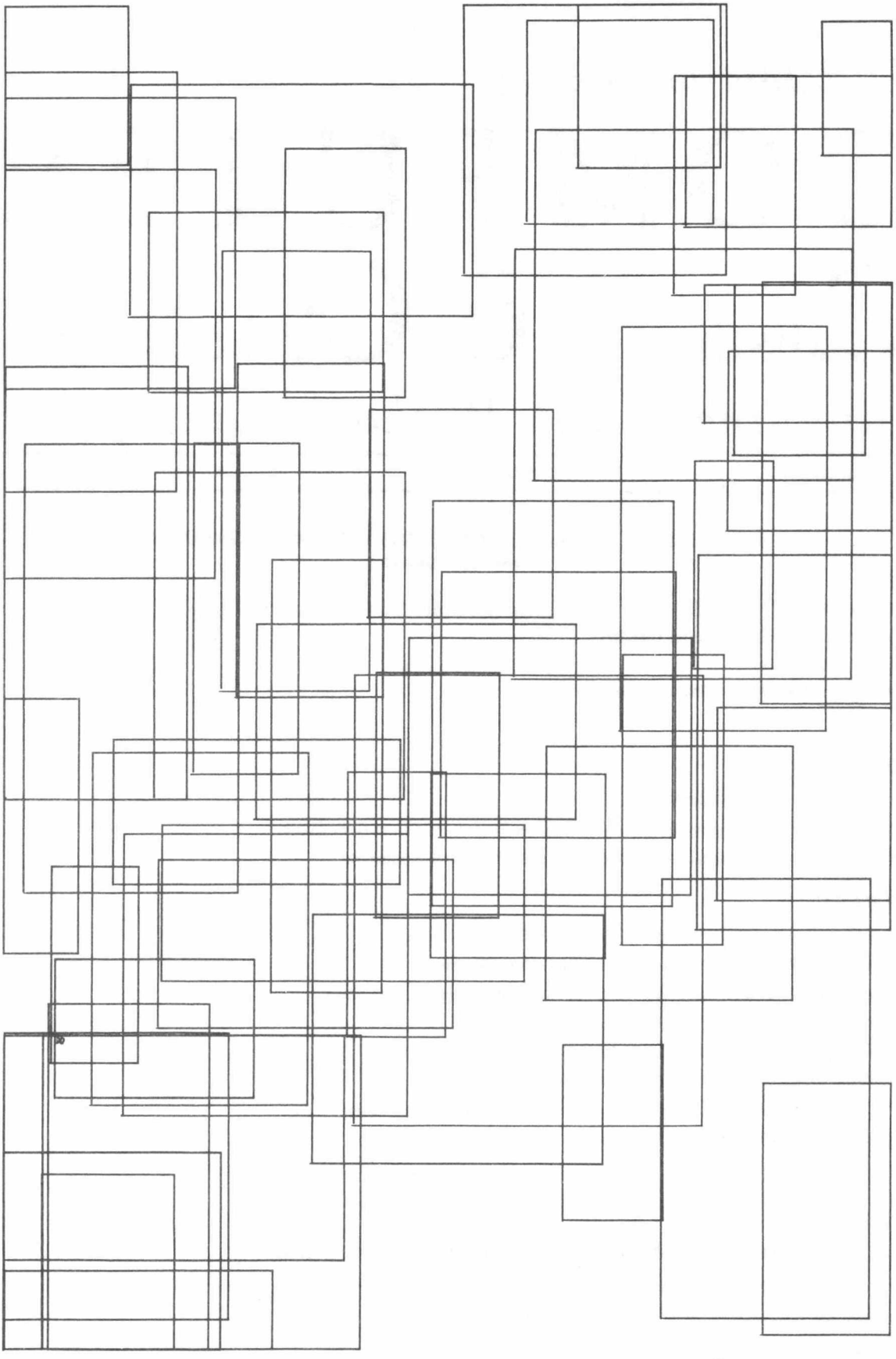
Raskin Misled . . . By *DDJ*?, Jef Raskin 4
Praise For Advanced Microcomputer Products & James Electronics, Stuart R. Fallgatter 45
Comments on James, Advanced Micro-Electronics & SWTPC, W. B. Goldsmith 46

SOFTWARE

String Handling for Pittman's 6800 Tiny BASIC, Henry L. Lee 5
Extensions to Wang's Palo Alto Tiny BASIC, Russ Hayden 6
Pittman 6800 Tiny BASIC Mods for SWTPC PR–40 Printer, Mickey E. Ferguson 7
STRINGOUT Mods for Espinosa's 6502 Routine, H. T. Gordon 8
Modifications to MicroSoft 8K BASIC 3.1 for Cromemco Z–80 20
An 8080 Disassembler, Ward Christensen 30
With Exquisite Documentation & Complete, Assembled Listing
Appendices to CASUAL, Bob Van Valzah 44

HARDWARE

High Speed Interaction Without Interrupt or DMA, Eugene Fisher 12
*Simple Floppy Disc Interface Detailed as an Example (*from *Electronic Design)*
$10 for 64-Characters & Scrolling on a TVT–II, R. Lynn Smith 25
64 Char/Line TVT–II Mods, Requests & Suggestions, Gary Alevy 28



© David Dameron 1979

FREE SOFTWARE? OR – SUPPORT YOUR LOCAL SOFTWARE VENDOR

Dear Jim: January 14, 1977

Enclosed is the manuscript I mentioned to you. Per your request, it is not in the form of a letter, but I suppose you could publish it as "An Open Letter to Manufacturers" which was my original intent.

As you can see, it is not an impartial feature article, but rather an opinionated analysis of the software problem. My big beef is the last point – the diversity of the system configurations in personal computers. If you think it is not clear (you are the editor) I will gladly accept suggestions for improvement.

Tom Pittman
Box 23189
San Jose, CA 95153

I will be honest. What you are about to read is a gripe, directed primarily at hardware manufacturers. I am in the software business (ITTY BITTY COMPUTERS), so you know my own bias.

Among personal computers there is a software vacuum. Tiny Basic was an attempt to start filling that vacuum. About a year ago PCC started publishing articles entitled "Build Your Own Basic", giving details and general architecture for Tiny Basic, a children's language conceived by Bob Albrecht. The idea was to get hundreds of hobbyists across the country working and sharing and producing tons of free software. Instead of hundreds, we actually saw only about a half a dozen successful working programs. Why?

My Tiny is one (or two, depending on your point of view) of that half dozen. I wrote it specifically in response to this software vacuum. It was an experiment, to see if there was any substance to the claim that if software were cheap enough it would not be ripped off like the Altair BASIC. The claim was largely valid; I have reason to believe the ripoff rate is less that 10%. But after nine months of business, I am still not selling anything bigger or better than Tiny Basic. Why?

A quick scan of a couple of hobby magazines shows typically 40 ads offering computer hardware and no software, 10 ads offering both hardware and software (in half of those the software went with the hardware) and only five ads offering software and no hardware (mostly books). Why?

Let me tell you why. Three factors work together to cause these phenomena.

One is that the personal computer market has not matured enough to recognize the importance of software, and there is not the demand for it. Obviously the readers of *DDJ* differ from the rest in this respect (*DDJ* is primarily a software journal), so I will not belabor this point. But even here the emphasis is often on bigger and better systems software instead of considering the system as a tool to achieve other ends.

Second, and more important, is the fact that good software is expensive. It is incredibly complicated, laborious to develop, and impossible to perfect. It is a known fact that the average programmer output is one line of debugged and documented code per hour, so an 8K 8080 program written in assembly language represents two man-years of labor, if done right. Obviously my performance has to exceed that average or I could not stay in business, but the order of magnitude is correct. Somebody has to pay for that cost. The hobbyist software exchange so widely acclaimed in the press (including *DDJ* !) simply has not materialized because the hobbyist is either unwilling or unable to expend the effort and discipline himself to the extent required to produce a program usable beyond his own computer. By comparison, hardware design is a piece of cake. I know – I've done quite a bit of that too. Software is not, and cannot be, free.

The third reason is the object of my gripe. If we want low-cost (to the user) software in abundance, we must find another way to pay for it.

One time-honored way is for the manufacturer to pay for it and bury the cost in his hardware. He offers the software "free" and hopes (or requires) that the customer will buy enough of his slightly higher-priced hardware to recover his investment. As I mentioned earlier, there were some five or six manufacturers doing this. A more subtle example is that of Processor Technology's "free" Software Number One. It was an assembler that ate vast amounts of memory, and guess who sells memory.

Another approach is to produce the software on venture capital and hope to sell enough of the packages to recover the investment. For what we would consider a reasonable price, a very large volume is necessary. More on this later.

The third alternative is no software. Forty or more purveyors of hardware consider this the way to go. Maybe they hope that "free software exchange" will happen. I will analyze some of this reasoning a little later.

Now I will admit that the picture is not all that sharply defined in black and white. For example, MITS charges a stiff price for their software AND requires you to buy their hardware. I call them "the IBM of the hobby computer field." Many of the available program products did not cost the developers as much as they should have, and when you try to use the programs you see why. But I said this was primarily a gripe against hardware manufacturers, so let me analyze exactly what is wrong with each of these three software distribution methods.

The best by far from the user's point of view is where the manufacturer provides the software with the hardware, either "free" or an an extra cost option. The hazards are all from the manufacturer's point of view, probably explains why there are so few doing this. A reasonable cost for 8K BASIC is $25,000. If the manufacturer expects to sell 5,000 computers, he must add $5 to the selling price of each one (that gets marked up to $8 out of the customer's pocket) to recoup his investment. No profits, just break even. All other things being equal, his computer now sells for $8 more than that of his competitor, who offers no software. In a market competing for beer money, price is of prime importance. So the customer accepts his free BASIC and buys the competitor's computer to run it in. Now actually, the BASIC probably only cost $15,000 because it was written by a moon lighter or $5,000 because it was written by a college student as a term project. The other $10,000 or $20,000 will be spent in maintaining the program because the moonlighter got a divorce and went to Venezuela, or the student vanished, leaving no documentation. The economics of the situation prevent very much software from being offered by the manufacturer unless the processor is so unique that the higher price will not be noticed and the software will not be lifted to other machines. One of these that comes to mind is the 16-bit machine offered by Dick Wilcox, who has amassed a tremendous amount of software for it by lifting from previous projects.

Other than these sparkling exceptions, the user should expect to get software from other sources as well, which brings us to the third catagory – the computers with no software (the competitors that the benevolent manufacturers in the first category have to worry about). Many of the buyers of these unpadded (no soft) machines have no use for software. Their only interest is in constructing bigger and (in some mysterious way) better elephants to sit in the attic. I won't worry about them because they represent a relatively dwindling minority of the computer buyers. Of the rest we have a similarly dwindling (though still considerably larger) proportion of hackers who like to build software elephants to metaphorically sit in the attic. They also do not need much supplied software.

The rest of us consider software as tools to be used in the making or doing of other things, be it playing games or writing other programs. Most of us would rather not write a program from scratch if it already exists. This is the market for that second category.

What about the independent software vendor? Let's look at the economics. Suppose he writes that 8K BASIC. That means investing $25,000 on speculation. Or for someone like myself without a lot of overhead, the cash investment is smaller, but we pass up an actual $25,000 job to work on the BASIC instead. In either case the value of the investment is the same. If I can sell it as a package to a manufacturer, that is great: take the money and run. But they are not buying, or at least not for $25,000 up front. They want to offer you "Royalties." On what? Not on the increased value your software gives their computer, but rather on the $5 that they will charge the customer for the package. I get letters with some regularity offering me 15% royalties. Fifteen percent of $5 is $.75/unit; since it is priced separately, at least half of the customers will opt not to buy it, so they have to sell 80,000 computers before I show a profit – if I live so long. Perhaps that is a little exaggerated, but I did have a manufacturer of a $1,000 computer tell me that $5 each was too much to pay me for a customized version of my software.

Along this line, I should point out that the software distribution systems that KILOBAUD and others are trying to set up are unlikely to attract much Grade A software for very much the same economic reasons. But there are other problems.

Well, then, the software vendor must go directly to the public. A recent census has it that there are 15,000 computers out there. If I sell this BASIC at $10 per copy, I might net $5 each, so I need to sell 5,000 copies. Considering that of those 15,000 computers, probably less than half have 8K of memory, and of those with the requisite hardware, less than half will buy the program (actual experience seems to run closer to 10%), there is already no way I can sell enough copies to recoup my investment. OK, so we call it a 5-year product life, and estimate 100,000 computers by 1980.

Let us assume further that many of those 100,000 computers still come with no software (though the trend is away from that), and because this is actually worth its $25,000 (as opposed to $5,000 or $15,000), it is recognized as one of the better products on the marketplace and captures

40% of the market from the other six or eight BASIC interpreters out there, some of which are "free". There still remains the fact that there are at least three different CPU chips going into those computers.

Finally, the Gotcha: Every one of those manufacturers with less than cradle-to-grave software support (i.e. most of them) designs the hardware configuration just a little bit differently, so that none of the programs that work on one computer will work properly and conveniently on the next, EVEN THOUGH THEY USE THE SAME CPU CHIP! I can understand a manufacturer designing in a new bus structure to lock the customers into buying high-profit peripherals from the same source as the low-profit CPU. But considering the costs of software development versus the potential profit, I should think the manufacturers would try to make software portability as easy as possible, so that their computer could be used with someone else's software. An obvious exception here is MITS: their BASIC is generally accepted as one of the better packages, and it is also quite expensive. A little software incompatability (and they did it, too!) might conceivably reduce the chances of BASIC walking to other computers.

So now the software vendor has to support different versions of his package for every different manufacturer. I have seven different versions of Tiny Basic, and there are still at least four more 6800 and 6502 systems that I do not directly support – all different. I have no idea how many different varieties of 8080 and Z-80 systems there are, but I'll bet there are at least six.

What can be done about this mess? First, you manufacturers can adhere to a few simple conventions:

1) Put your monitor in high memory, and leave low memory in RAM.
2) Avoid using memory Page 00 for any system software or firmware, but if you must use some, restrict it to the first and/or last 20 or so bytes.
3) If the monitor needs additional RAM for local storage, or buffers, put it at the far end with the monitor ROM.
4) Start the stack pointer at the highest available memory address, so it has the most freedom to grow without interfering with low memory.
5) Provide monitor subroutines to:
   a) Read a character from the console keyboard.
   b) Display a character on the console output device.
   c) Terminate the current line of display and start a new one.
6) Always use the A register (accumulator) to pass one-byte data to and from monitor routines. Avoid altering the other registers.
7) Use a Jump vector at one end of your monitor to define the entry points to monitor subroutines, so that subsequent revisions will not affect previously developed software packages.
8) Provide at least one standard program loading peripheral such as a paper tape reader or Kansas City audio cassette, even if you do not include a way to generate output media in the same format. There is more need to load package software than there is to produce it. Your Super High Speed Patented Data Recording Technique is fine for local data and program storage.
9) Support the CPU chip manufacturer's conventional hexadecimal program format for that program loader hardware. Again, save the tricky double-checksum reverse binary format for local program and data storage. The software vendor cannot afford to support 23 different media standards, but there is nothing wrong with requiring each buyer of a package program to read it in slowly, then recopy it to his fast local storage peripheral.

Those of you who are not manufacturers, you can help too. Complain to nonconforming manufacturers. Refuse to buy their products. (Remember, you have to interface package software to the non-conforming systems, and eventually the software vendors will only support good-guy systems.)

If we in the software business can ever manage to concentrate on new products that have a reasonable chance at high volume, you might begin to see more and better low-cost software. And when that happens, more people will be willing to buy personal computers, which will make the software market more attractive, which in turn will result in more and better software. It's up to you now.

DESPERATE KIM-I SEARCH

Dear Sirs: January 17, 1977

As a KIM-1 owner, I am desparately searching for a BASIC Interpreter to be run on my KIM-1 System. I would be most appreciative if you would print (I will gladly pay you to do so) the following notice in the next KIM-1/6502 Newsletter:

HELP! Desparately looking for a BASIC Interpreter to run on my KIM-1 System. Will gladly pay! At your mercy!

Edward L. Pavia
127 Sugar Maple Drive
Rochester, NY 14615

RASKIN MISLED . . . BY *DDJ*?

Dear Dr. Dobbs, January 29, 1977

In my position as consumer advocate for the hobbyist I would be remiss to not point out such things as misleading articles and advertisements. Thus I must object to the heading of a recent article in my own beloved *Dr. Dobbs*, due I am sure to misguided enthusiasm rather than intentional deception. The heading to Bob Van Valzah's interesting CASUAL language touted "It's a BASIC ... It's an APL ... It's CASUAL". This letter in no way reflects on the CASUAL language or the article about it, just on the claim that it is in any way like APL.

My heart beat faster when I saw the mention of an APL for the microcomputer, and I felt sorely disappointed when I read the description of the language. Certainly if I had purchased that issue (Vol. 1, No. 10) on the basis of the table of contents I would have felt cheated. APL is a very elegant and powerful language, and its notation and rigor far exceed what the hobbyist who knows only BASIC and/or assembler.

The only APL-like feature to be found in CASUAL is its use of single-character keywords rather than BASIC's many character keywords. There are a number of non-APL languages that share this attribute. The hallmarks of APL, its array handling basis, its strict right-to-left evaluation, its unique and powerful operators, are none of them in CASUAL.

CASUAL is, indeed, very much like BASIC, with some neat improvements to be sure. But that is not the burden of my message which is: Straighten Up and Fly Right, Dr. Dobbs. The secret avenger of consumer annoyances watches even you!

Jef Raskin

Tsk, tsk – was it your enthusiasm, or ours, that misguided you? The "misleading" title implied, equally, that CASUAL was BASIC-like. –Editor

HOW 'BOUT VISIBLE MUSIC VIA YOUR TV?

Dear Jim, Oct. 11, 1976

I agree with Tim Bonham's comment on p. 37 of the October issue of *DDJ*. Even though a language processor may accept free-form input, it is highly desirable to maintain visual alignment of the beginning of each field. This would not preclude the use of variable-width characters but would require the display-generating program to keep track of the current "dot count," so that proper field alignment could be preserved. Numerals would all be assigned a dot width of 5, so that columns of numbers would line up evenly.

Speaking of video displays, since many hobbyists are using an ordinary or modified TV set as a display device, there is no reason why the same TV could not be used as an audio output device at the same time. For example, when a bell character is sent to a TTY, a bell rings. When a bell character is sent to a TV display, nothing happens. Some commercial video terminals have a beeper to simulate a bell, and a similar beeper could be added to a TV display. But why not use the audio section of the TV set instead? An audio oscillator could be added to a video display board or, better yet, an all-purpose D/A audio output port could be connected to the TV and used not only for bell simulation but for speech output, music synthesis, etc. The TV could be used simultaneously for audio and video display in such applications as "visible music," animated cartoons with synchronous sound, educational programs, and all kinds of games. Some TV sets are equipped with an auxiliary phono input jack, but others would have to be modified to facilitate access to the AF amplifier. Safety precautions would have to be taken on "hot-chassis" sets (e.g., an opto-isolator or isolation transformer).

Sincerely,

Jim Day
17042 Gunther St.
Granada Hills, CA 91344

STRING HANDLING FOR PITTMAN'S 6800 TINY BASIC

Dear Jim,

Enclosed are modifications to Pittman's Tiny BASIC version 1.2S. These changes will enable the user to add string handling capabilities to Pittman's Tiny BASIC. The additional options are handled through the USR verb. Although the changes are for a Sphere user with a V3N keyboard, they are easily adaptable to other 6800 systems. The Sphere version is relocated 100 (base 16) higher in address.

Sincerely,

Henry L. Kee
42-24 Colden St.
Flushing, NY 11355

```
SUBROUTINES

ORIGIN ACO    setup ID

DE 33         store 33 - 36
FF OAE0       areas used by TINY BASIC
DE 35         for INPUT verb
FF OAE2
86 24         first character of ID = $
97 33
B6 OAFE       second character of ID
97 34
39            exit

ORIGIN AD4    restore TINY BASIC work areas

FE OAE0
DF 33
FE OAE2
DF 35
39

ORIGIN AF0    set pointers

DF 18         location 18 contains first character of string address
97 FF         location FF contains last character of string address
86 0B
97 FE
97 18
39

INPUT $

ORIGIN A00

BD 0AF0       set pointers
86 3F         emit a ? symbol
BD 0209       put
BD 0206       get
DE 18         load IX
A7 00         store character in string
08            increment IX
DF 18         store IX pointer
81 0D         test for CR
27 0A
81 0B         test for BS
27 0C
96 FF         test for end of string buffer
91 19
26 03
96 19         set value for next
39            exit
7E 0A08       goto next get
DE 18         routine to backspace and blank out
09            use BACK SPACE to backspace
09
DF 18
DE 1C
09
DF 1C
86 20
BD FCBC       change to BD 0209 if CRT is bypassed
DE 1C
09
DF 1C
BD FCBC       change to BD 0209 if CRT is bypassed
DE 1C
09
DF 1C
7E 0A08

modify contents of location 301
change it from 0A to 0C

WRITE $

ORIGIN AA0

BD 0AF0       set pointers
86 2D         set for WRITE
7E 0A78       goto entry point for READ $

PRINT $

ORIGIN A46

BD 0AF0       set pointers
DE 18         load IX
A6 00         pickup character
08            increment IX
DF 18         store IX pointer
BD 0209       put character
81 0D         test for CR
27 06         BEQ
96 FF         test for end of buffer
91 19
26 03
96 19
39            exit
7E 0A49

INITIALIZE

ORIGIN A64

BD 0AF0       set pointers
96 FF         store ACIA in AFF
B7 0AFF
96 19         store ID in AFE
B7 0AFE
39            exit

READ $

ORIGIN A73

BD 0AF0       set pointers
86 91         set for READ
B7 0A98       store READ or WRITE - entry point for WRITE $
DE 18         set start of string block
DF 3C
DE FE         set end of string block
DF 3E
B6 0AFF       set ACIA
97 39
86 F0
97 38
BD 0AC0       set ID
86 FF         set FLAG
97 3A
BD FB00
BD FB91
BD 0AD4       restore TINY BASIC work area
39            exit

PEEK $

ORIGIN AE6

BD 0AF0   set pointers
DE 18     load IX
A6 00     load A
39        exit

POKE $

ORIGIN AAA

BD 0AF0   set pointers
DE 18     load IX
96 FF     load A
A7 00     store in string location
39        exit
```

EXTENSIONS TO WANG'S PALO ALTO TINY BASIC

Dear Jim, Oct. 1, 1976

I have added some extensions to Li-Chen Wang's Palo Alto Tiny BASIC which may be of interest; they include machine language subroutine linkage, string I/O, and memory location examination and modification. These additional statements and functions take approximately 150 bytes; however, if you have installed P.A.T.B. as written (*DDJ*, May 1976, pp. 12-25), your program space is not reduced, as the routines fit into the expansion space designed into the Tiny BASIC program.

Loud cheers here for Mr. Wang's forethought in providing for improvement of his program. This, along with the excellent structure of the execution skeleton, made implementing these commands a lot simpler than it might have been. I think anyone writing a "Tiny" language should provide room for other users to play with it; sometimes this is more fun than writing programs in the language. Credit is also due Erick Mueller for the idea behind the string handling, and MITS BASIC for the form of PEEK and POKE.

There are four added statements and two functions: in the following descriptions, X may be any number, variable or expression permitted by P.A.T.B.

CALL X: calls a machine language subroutine located at X. I suggest passing arguments, if necessary, by placing them in specified variables.

I$ X*: inputs a string from the keyboard to successive memory locations starting at X. The length of the string (including delimiter) is returned in variable Z.

0$ X: outputs a string (no CR) input by the above statement.

POKE X*,Y: places the value of expression Y (modulo 256) in location X.

Functions are:

PEEK (X): returns the ASCII value of a single character. Note that
PRINT "A" will output *A*
PRINT ("A") will output *65*

*note: both the Tiny BASIC program and the user's program space are protected; I$ and POKE will give a "SORRY" message if you attempt to deposit into these locations.

In my listing, I have marked as Mr. Wang did, with @@@@, two locations which may be changed if you are running P.A.T.B. in other than 8K; I have 4K and run a modified version.

Another change to P.A.T.B. I have found to be useful is to suppress the carriage returns it does before and after the the "OK" message. To do this, change the following locations:

00AD	00
00BA	00 00 00
074E	CD 0E 00 C7 (provides CR after CNTRL/C)

This cuts down greatly on wasted paper, especially when doing a number of direct commands.

Yours,

Russ Hayden Box 139 MIT Br. P.O.
Cambridge, MA 02139

```
****ADDITIONS TO WANG'S PALO ALTO TINY BASIC****  10/01/76 Russ Hayden

*Mods to statement table
0181  43414C4C          "CALL"
0185  8752
0187  4F24              "O$"
0189  875B
018B  4924              "I$"
018D  8764
018F  504F4B45          "POKE"
0193  8797
0195        8401

*Mods to function table
01B1  5045454B          "PEEK"
01B5  8790
01B7  22                ' " '
01B8  87AF
01BA  84EC
```

A sample program using the added features

Program output is underlined.

```
LIST
   1 A=3000
   2 PRINT "STRING ONE? ";;I$ A
    3 B=A+Z
   4 PRINT "STRING TWO? ";;I$ B
   5 FOR C=A TO B+Z
   6 IF PEEK(C)="N" POKE C,"M"
   7 NEXT C
   8 PRINT "MODIFIED AND SWAPPED STRINGS"
   9 O$ B;PRINT " ";;O$ A;PRINT
OK>RUN
STRING ONE? NUMBER ONE
STRING TWO? I'M STING TWO
MODIFIED AND SWAPPED STRINGS
I'M STIMG TWO MUMBER OME
OK>RUN
STRING ONE? AAANAAA
STRING TWO? BBBNBBB
MODIFIED AND SWAPPED STRINGS
BBBMBBB AAAMAAA
OK>
```

Input 1st string to locations beginning at 3000

B = A + length of string input to A
Input 2nd string to next available space

Check over entire space (Z is now length of string at B)

Note spacing and CR after strings

```
*Code for new statements and functions
0752  DF              CALL    EXPR            Get address of routine (RST 3)
0753  D5                      PUSH D          Save text pointer
0754  015907                  LXI B HERE      Push the return address
0757  C5                      PUSH B
0758  E9                      PCHL            Jump to routine
0759  D1              HERE    POP D           Return here; restore pointer
075A  F7                      FINISH          Done          (RST 6)
*
075B  DF              0$      EXPR            Get address of string (RST 3)
075C  D5                      PUSH D          Save text pointer
075D  EB                      XCHG            DE is now addr. of string
075E  AF                      XRA A
075F  CD3C06                  CALL PRSTNG         Print until a zero
0762  D1                      POP D           Restore pointer
0763  F7                      FINISH          Done      (RST 6)
*
0764  DF              I$      EXPR            Get addr. to store string  (RST 3)
0765  D5                      PUSH D          Save pointer
0766  EB                      XCHG
0767  2A1308                  LHLD TXTUNF     Get  low limit to DE
076A  EB                      XCHG
076B  E7                      COMP            DO compare [HL-DE]    (RST 4)
076C  DAD005                  JC ASORRY       If HL less than limit, no input
076F  CDD705                  CALL GETLN&1    Input to buffer (no prompt)
0772  44                      MOV B,H
0773      4D                  MOV C,L         BC is addr. to move string to
0774  EB                      XCHG            HL is last chr. in plus one
0775  2B                      DCX H           Now points at last chr.(CR)
0776  11371F                  LXI D BUFFER    DE is first chr.(Note: 8K system)
0779     D5                   PUSH D          Save to compute length
077A  CD8D06                  CALL MVUP       Move (DE) to (BC) until DE equ. HL
077D  AF                      XRA A
077E  02                      STAX B          Store zero instead of CR to end str.
077F  D1                      POP D           Equ.first chr.
0780  23                      INX H           Equ. last chr. plus one
0781  CD6005                  CALL SUBDE      HL equ. HL-DE (length of string)
0784  EB                      XCHG            Put in DE
0785  21341F                  LXI H Z         Store length in variable Z ( Note:
0788  73                      MOV M,E         8K system)
0789     23                      INX H
078A  72                      MOV M,D
078B  D1                      POP D           Restore text pointer
078C  CD9105                  CALL FIN        Go on if was stored statement;
078F  C7                      START           otherwise input zapped direct (RST 0)
                                              command, restart system
*
0790  CDF804            PEEK   CALL PARN       Argument is in parentheses
0793  6E                      MOV L,M         HL equ. peek addr.; get the
0794  2600                    MVI H 0         byte and return it in HL
0796  C9                      RET
*
0797  DF              POKE    EXPR            Get poke address
0798  D5                      PUSH D          Save pointer
0799  EB                      XCHG
079A  2A1308                  LHLD TXTUNF     Get low limit to DE
079D  EB                      XCHG
079E  E7                      COMP            If poke addr. less than limit,
079F  DAD005                  JC ASORRY       no poke
07A2  D1                      POP D           Restore pointer
07A3  E5                      PUSH H          Save poke addr.
07A4  CF                      TSTC ",",PK1    Is next text a comma? If not, skip
07A5  2C                                      to PK1     (RST 1)
07A6  05
07A7  DF                      EXPR            Yes, get expression for poke
07A8  7D                      MOV A,L         Value (mod.    256) to acc.
07A9  E1                      POP H           Get back poke addr.
07AA  77                      MOV M,A         Poke it
07AB  F7                      FINISH          Done
07AC  C3A405            PK1   JMP QWHAT       No comma in text; error
*
07AF  1A                "     LDAX D          Get text chr. after quote
07B0  13                      INX D
07B1  6F                      MOV L,A         Ascii value to HL
07B2  2600                    MVI H 0
07B4  CF                      TSTC '"', ASC1  Is chr. followed by a quote?
07B5  22
07B6  C1
07B7  C9                      RET             Yes, return
07B8  C3A405            ASC1  JMP QWHAT       No; error
```

MUCH-REQUESTED EXTENSIONS TO PALO ALTO BASIC

Dear Jim: **February 7, 1977**

Many readers of *DDJ* have asked me about extending Palo Alto Tiny Basic to include strings. Well, it has been done, and plus a lot more. Russell Hayden of Cambridge, Mass. has extend PATB to call machine subroutines, string I/O, and memory PEEK and POKE. (You gave me a copy of his letter last October, remember?) In a recent letter (to me this time) he also extended strings to assignment statement ex. LET A$ = "HELLO" and TAB(X) and CHR(X) to PRING statement. I am sending you a copy of this extension and a copy of the copy of his early letter. I think many of *DDJ*'s readers would like to see them.

Russell's extensions are very compact and efficient. He utilized a lot of the service routines in PATB and managed to put all the extensions but TAB(X) and CHR(X) into the 2K space as the original PATB. I could not have done it better. Cheers for tiny power.

Sincerely yours,

Li-Chen Wang
150 Tennyson Avenue
Palo Alto, CA 94301

PITTMAN 6800 TINY BASIC MODS FOR SWTPC PR-40 PRINTER

Dear Dr. Dobb, Oct. 7, 1976

My system is all SWTPC:

1. M6800 computer with 16K RAM
2. AC-30 cassette interface
3. PR-40 printer
4. CT-1024 terminal

It has been modified slightly. The computer's clock drives the terminal so the terminal can be operated at any of the baud rates available from the computer by flipping a single switch on the computer! Full details if you want them. [**Sure.**] Am presently modifying AC-30 to allow tapes to be made at speeds up to 1200 baud – more details when completed and debugged [*Do* **tell us the intimate details!**].

Also included is a listing of a routine I have written that allows Tom Pittman's M6800 Tiny BASIC to drive a SWTPC PR-40 printer. My routine is loaded into Mikbug's RAM and one change is made to Tom's Tiny:

Address	Inst		
0109	7E E1D1	Jump to OntEEE	Tom's Tiny
0109	7E A04C	Jump to printer driver	Mod. Tiny

The B and Index Registers are used but saved and restored. All communication from the computer to the control terminal is printed on the printer also. When it is not desired to print simply turn the PR-40 off; turn it on only when you are ready to print. Data in A051 and A052 should be changed changed to the appropriate address for the I/O slot you are using. A064-A074 is SWTPC's Outchr subroutine that is supplied with the PR-40.

Happy computing!

Mickey E. Ferguson
P.O. Box 708
Trenton, GA 30752

```
A04A 00
A04B 00
A04C FF A04A
A04F 37
A050 CE 800C
A053 C6 FF
A055 E7 00
A057 C6 3F
A059 E7 01
A05B 8D 07
A05D 33
A05E FE A04A
A061 7E E1D1
A064 A7 00
A066 C6 37
A068 E7 01
A06A C6 3F
A06C E7 01
A06E 6D 01
A070 2A FC
A072 E6 00
A074 39


ABOVE IS PRINTER ROUTINE
FOR USE WITH SWTPC M6800
AND TOM PITTMAN'S 6800 TINY BASIC
```

STRINGOUT MODS

Dear Dr. Warren, February 2, 1977

In *DDJ* Vol. 1, No. 10 (p. 50), Marcel Meier presented a revision of the Espinosa 6502 program STRINGOUT, cutting its length from 64 to 44 bytes and making it relocatable in RAM. The following revision cuts it to 40 bytes, allows location in ROM (or RAM), is compatible with 6502 systems other than the OSI-400 (e.g., KIK-1), and allows selection of a variety of subroutines to work on the character string.

```
          ORG $O200
YKEEP     EQU $OOE8
AKEEP     EQU $OOE9
LO        EQU $OOEA
HI        EQU $00EB

0200 84 E8     BEGIN STY YKEEP     (save Y index)
0202 85 E9           STA AKEEP     (save accumulator)
0204 68              PLA           (stock pull return LO)
0205 85 EA           STA LO        (store zero-page LO)
0207 68              PLA           (stock pull return HI)
0208 85 EB           STA HI        (store zero-page HI)
020A AO OO           LDY #00       (clear Y index)
020C C8        ALOOP INY           (increment Y)
020D B1 EA           LDA (LO, Y    (get character)
020F FO 06           BEQ EXIT      (if null, exit)
0211 20 00 01        JSR ANYSUB    (selector subroutine)
0214 18              CLC           (clear carry)
0215 90 F5           BCC ALOOPE    (always loops back)
0217 98         EXIT TYA           (start Y offset add)
0218 38              SEC           (offset is Y + 1)
0219 65 EA           ADC LO        (add to return LO)
021B 85 EA           STA LO        (LO = L) + offset)
021D 90 02           BCC NOCAR     (carry clear, no page cross)
021F E6 EB           INC HI        (increment page number)
0221 A5 E9     NOCAR LDA AKEEP     (restore accumulator)
0223 A4 E8           JMP (LO)      (return to program)
```

The selector subroutine ANYSUB is located in RAM, at the bottom of the stack where it is relatively secure.

```
0100 20 XX XX  ANYSUB JSR XXXXXX
0103 60               RTS
```

It allows subroutines in ROM (or RAM) to call any of several ROM (or RAM) subroutines, by prior setting of the address XX XX. Although STRINGOUT was intended for teletype output of a character string, ANYSUB allows it to feed characters to other devices. One possibility is a sequence of digital commands to a control system. For this purpose, the above version is restrictive since it excludes 00. This restriction can be removed by making exit conditional on a sequence of two 00 bytes, by modifying (with 8 more bytes) the ALOOP section as follows:

```
ALOOP INY
      LDA (LO), Y
      BNE OUTPUT
      INY              (test next character)
      LDA (LO), Y
      BEQ EXIT         (exit if also = 00)
      DAY              (if not, output the
                         single-null character)
      LDA (LO), Y
OUTPUT JSR ANYSUB
      CLC
      BCC ALOOP
 EXIT TYA              (same as earlier version)
```

One possible use of the unrestricted STRINGOUT would be to relocate blocks of instructions (up to 256 bytes) in RAM. In fact, this is how I tested the program, since I do not have a teletype. Since the Y index (as well as the character) is available to ANYSUB, STA instructions using the Absolute, Y addressing mode can be used.

Sincerely,
H. T. Gordon
University of California, Berkeley

STRINGOUT MODS – ADDENDA

Dear Mr. Warren: February 4, 1977

(This is an addendum to my letter of February 2.)

While playing with the double-null-terminator version of STRINGOUT, it dawned on me that the restriction that its transfer subroutine pre-exist at 0100 is unnecessary, since it can create it from the first 4 bytes of the string. The following alteration replaces the LDY instruction at 020A in the original version by a block of 12 bytes. It causes the first 4 bytes of the string to be copied into 0100-0103. These can be JSR XX XX RTS to create a subroutine, but if the next 2 bytes are 00 00, then STROUT (the routine is now so different that a new name seems justified!) can be used to put *any* 4-byte sequence into 0100-0103 using 9 program bytes.

```
020A A0 04        LDY #04          (set to store 4 bytes)
020C B1 EA SLOOP  LDA (LO), Y      (gets 1 of first 4 bytes)
020E 99 FF 00     STA STBASE, Y    (stores in low stack)
0211 88           DEY              (decrement Y index)
0212 DO F8        BNE SLOOP        (loop until zero)
0214 AO 04        LDY #04          (reset Y index)
0216       ALOOP INY               (same as double-null
                                    version)
```

The full program is now up to 58 bytes, but much more versatile.

Sincerely,
H. T. Gordon

P. S. Enclosed is a check for a 1-year subscription (in no way conditional on publishing my stuff – it has to stand on its own merits!). Your journal is really the best on programming

SECRET CODES, ANYONE?

NOTICE:

CRYPTOLOGIA, a journal devoted to all aspects of cryptology, will be published four times a year. There is a need for a forum for the exchange of ideas related to cryptology in the public sector. This journal will meet that need with research papers, survey articles, personal accounts, reviews, educational notes, and problems. Some of the areas which will be discussed are mathematical, computational, literary, historical, political, military, mechanical and archeological aspects of cryptology. Further information may be obtained from CRYPTOLOGIA , Albion College, Albion, Michigan 49224. The Editors: Cipher A. Deavours, Department of Mathematics, Kean College; David Kahn, Department of Journalism, New York University; and Brian J. Winkel, Department of Mathematics, Albion College.

SYMPLE CYPHERING

by David Gruhn

8500 32nd Ave. NW, No. 1
Seattle, WA 98117

October 28, 1976

Jon Stedman wrote an interesting article on the Playfaire cypher in the October 1976 issue of *DDJ*; thank you, I enjoyed it. The Playfaire cypher would seem to have several good points: it does not seem trivial to break, it is relatively easy to use, and the only equipment required (beyond pencil and paper) is a small (5 x 5) table that is easily reproducible from a given key (so a person need not actually carry the table between uses). As part of his introduction, Mr. Stedman suggested that someone program the cypher. If the Play–faire, specifically, is desired, then the Playfaire should be programmed. If, on the other hand, *a* cypher (not necessarily Playfaire) is desired, there are easier cyphers to use with a computer.

TWO SIMPLE TECHNIQUES FOR CODING AND DECODING ON A COMPUTER

To encode a text in a computer, you need to take the bit configurations that represent each character, and substitute in some other bit configurations, using some predetermined set of rules that will allow you to, later, reverse the process; to decode the encoded characters. Say, for example, you represent the letter 'A' in your machine with the bit configuration X'41'; you could substitute some other bit configuration, say X'6E for each occurance of X'41'. Some other similar substitution would have to be made for 'B', 'C', etc. One relatively simple method for making this kind of substitution for all possible bit configurations is to exclusive-or the bit configuration with some 8-bit *key* (assuming an 8-bit configuration); the resulting bit configuration is the encoded data. For simplicity, let us call these 8-bit configurations bytes like everyone else. Since the exclusive-or has the most delightful property that

if A + K = C *then* C + K = A

we can retrieve our original byte by exclusive-or'ing the encoded byte with the same 8-bit key. This gives us a simple rule for substituting an encoded byte for a non-encoded byte; a rule that, in addition, is its own inverse. Note that this assumes that a byte is the easiest unit with which to work, regardless of the number of bits in a character. Since the text in the machine can be thought of as one, long bit string, there is no reason to choose the working unit to be on character boundries as long as you can reassemble working unit lengths of the encoded bit string when you wish to decode.

The obvious objection to the above method of encoding is that, if you always substitute the same encoded byte for a particular non-encoded byte, you will have a simple substitution cypher that is relatively easy to break. As long as the same key is used, there will be a one-to-one relationship between the non-encoded byte and the encoded byte; i.e., a given non-encoded byte will always generate the same encoded byte. One possible solution, then is to have some predetermined (but not obvious) method for altering the key. We want the key to appear to change randomly while actually changing according to some predetermined rule that will allow us to reproduce the sequence of keys for decoding; we want a pseudo random number generator (PRNG).

Now to encrypt a text, start the PRNG with some seed; this seed becomes the key for this text. For each non-encoded byte of text, get a number from the PRNG and select 8 bits from that number according to some rule; these 8-bits are the key for this non-encoded byte. Exclusive-or this 8-bit key with the byte of text to generate the encoded byte. For the next non-encoded byte, get the next number from the PRNG and use the same rule as before to select an 8-bit key from that number, then exclusive-or to generate the next encoded byte. Continue this process to convert the entire text. To decrypt the encoded text, start the PRNG with the same seed that was used to encrypt the text (the key for the text), and go through the same process for decrypting as was used for encrypting. The difference this time will be that we will be reading the encoded text and be generating the decoded text. We can use the same routine for encrypting and decrypting because the exclusive-or is its own inverse.

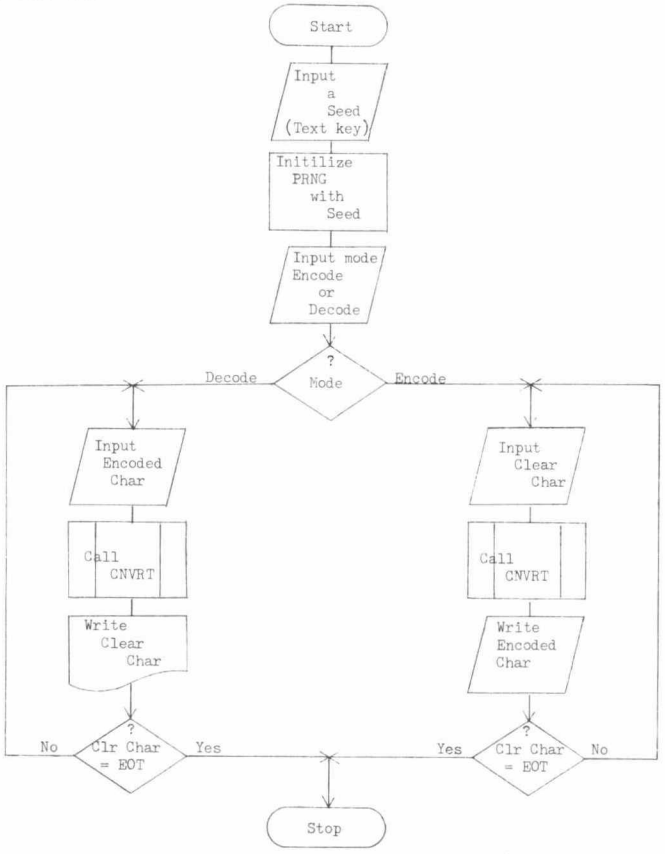
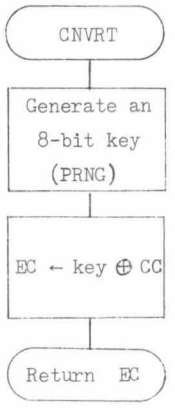


Figure 1a -- Flowchart to Encode/Decode

Note: EOT = End-of-Text Character



Enter: CC = Char to encode/decode

Exit: EC = Encoded/Decoded char

Figure 1b -- Encode/Decode with Exclusive-Or

This technique has the advantages of being relatively easy to program and, with some consideration, the series of keys can be quite long before repeating. The latter is important since finding repetitions of the key sequence is an important part of cryptographic analysis. One possible disadvantage, however, is that the encoded byte can take on any value from 0–255, most of which are unprintable. This is no problem if the encoded text is to be kept on a machine sensible medium, but if it is to be printed,

it will probably have to be printed in hex, which means two characters of the encoded text for every one character of the clear text. This doubling could make the printed, encoded text rather voluminous.

If the doubling of the printed, encoded text is too much of a problem, an alternative encoding scheme is to replace subroutine CNVRT with subroutines TCNVRT and TCNVRTD to encode and decode respectively.

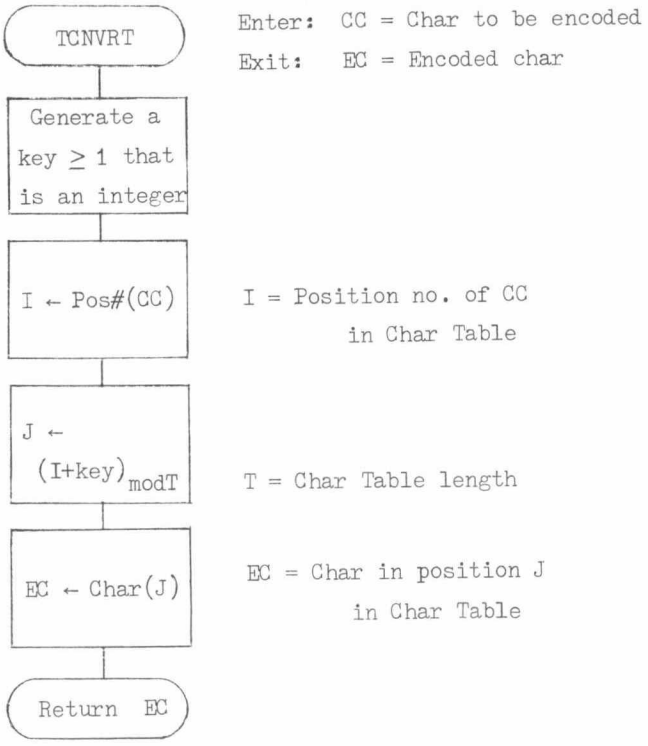


Figure 2a -- TCNVRT, Encode with Character Table

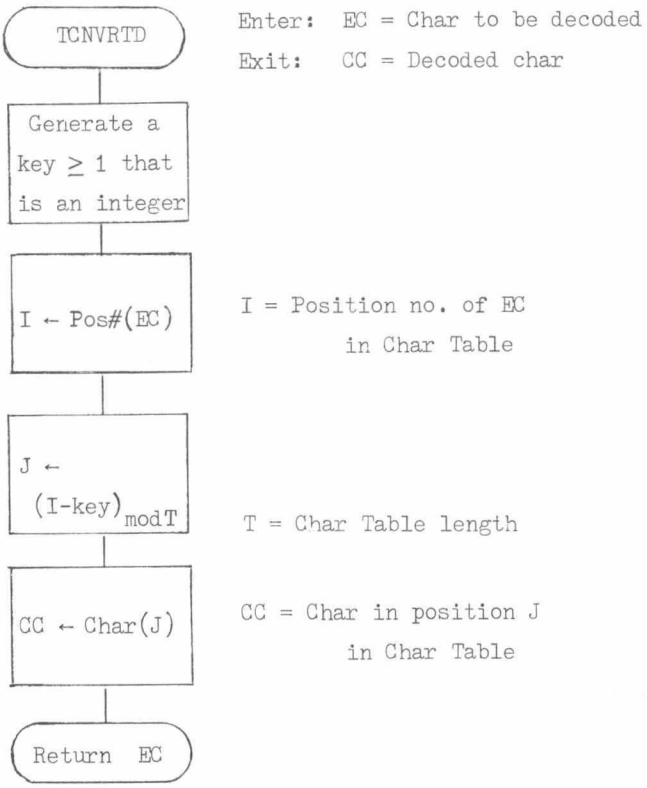


Figure 2b -- TCNVRTD, Decode with Character Table

TCNVRT uses a table of legal, clear text characters. To encode a character, its position in the table is found (for example 'A' might be in position 0, if you have no imagination). An integer, pseudo random number greater than zero is generated (a key) and added, modulo the table length, to the clear text character's position number. The character (printable) in the position generated is the encoded character.
Example:

Table: A B C D E
Pos.#: 0 1 2 3 4
Table length = 5
Pseudo random numbers (in order generated): 39, 2, 16
Clear Text: ACE
Coded Text: EEA
Pos.#(A) = 0
Key = 39
(0 + 39) mod 5 = 4
Encoded byte = E

Pos.#(C) = 2
Key = 2
(2 + 2) mod 5 = 4
Encoded byte = E

Pos.#(E): = 4
Key = 16
(4 + 16) mod 5 = 0
Encoded byte = A

The decoding is performed by subtracting, modulo the table length, the key from the encoded character's position to generate the clear character's position. This technique is not as fast as the other since it involves a table look-up, modulo addition, and a table access rather than one exclusive-or, but it does guarantee that every character of the encoded text will be printable. It does, however, restrict your clear text to characters in the table.

DR. DOBBS APOLOGIZES

Dear Dr. Dobbs:
Talking dirty in a cryptogram is super except when you and your ten year old daughter (for real) decode it. It lost a lot of its humor while I tried explaining it.
(Name Withheld) Sunnyvale, California

I wish to offer a sincere apology for this. When I scanned the article, it occurred to me to check the sample cypher for *accuracy* (but I didn't). However it never occurred to me to check it for immature vulgarity. Such childish trash has no place in *DDJ*. I sincerely regret the embarassment it must have caused you.
Jim Warren, Editor

PERSONAL COMPUTER STORE IN SAN FRANCISCO FINANCIAL DISTRICT

The BYTE SHOP is now open in the San Francisco financial district, at 321 Pacific Avenue between Battery and Sansome Streets. The BYTE SHOP features the sale of computer kits, books on theory, software, and the applications of computers, memories — both cassette, disc and semiconductor, as well as assembled ready-to-operate computer systems. The BYTE SHOP has a wide variety of operating computer systems on display programmed to demonstrate games such as Life, Star Trek, NIM, and Lunar Lander. Business applications such as general ledger, payroll, accounts receivable and accounts payable, text editing and special purpose application software are also up and running.

A computer doctor is always on hand to provide free consultations on your computer application. The BYTE SHOP is open Monday through Friday from 10:00 a.m. to 6:00 p.m. You are invited for free demonstrations of personal computers in operation and a free copy of our new Micro-computer Recipe Book.

For additional information contact: J. Benbow Bullock, BYTE SHOP, 321 Pacific Avenue, San Francisco, CA 94111, (415) 421-8686.

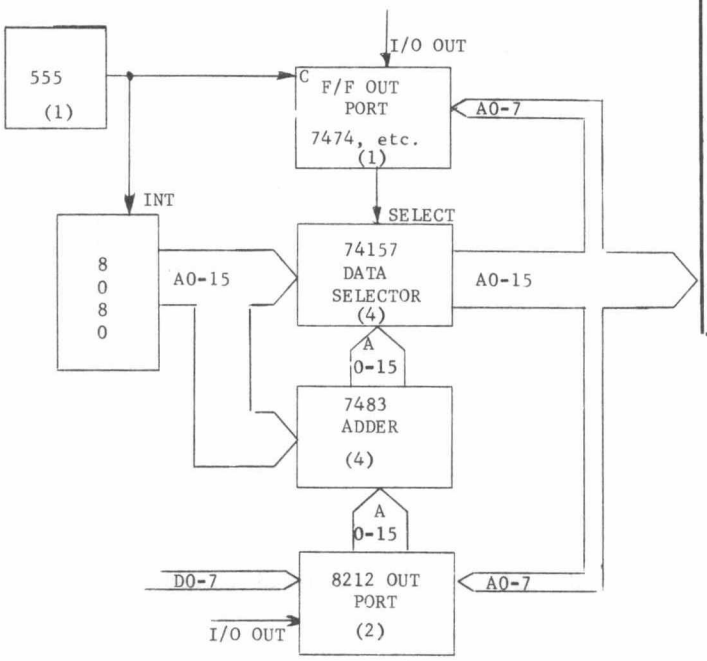
GAME SOLUTIONS & NOTES ON SEGMENTED MEMORY

Edward I. Comer

How would you like the answers to Suding's BRAIN TEASER game, and *Byte*'s SHOOTING STAR GAME. Using the chart below, you can win every time.

To use the chart, take the starting positions and convert them into a number. For example, if a star were in positions 1 and 7, then the number to use in the table would be 17. Take this number and enter the table. Enter the numbers beside the entry, and you will inevitably win the game.

NOTE: You can find the rules to play in *BYTE*, May, 1976.



How would you like to have SEGMENTED MEMORY for your Micro?

The sketch is an idea that you may like to develop into your system. This system divides memory into two sections. (1) KERNAL memory, which is where system programs reside, and (2) USER memory, where application programs reside. Divided in this way, if a user program crashes the system programs will not be harmed.

HOW IT WORKS: When the system starts up it is in the KERNAL mode (F/F Port RESET). Once every second the 555 generates an interrupt that does two things. First it resets the F/F out port, and it branches the 8080 to service a KERNAL interrupt routine. When the KERNAL programs are ready to release control to the USER programs, it sets the F/F out port. The F/F port drives the 74157 select line "high". This stops the address lines from being gated straight through, and instead gates the output of the ADDER onto the address bus. The adder adds together the PHYSICAL address from the 8080, and the segment offset address which was loaded into the 8212 upon startup. Thus if the physical address was 001020, the 8212 segment offset address was 007000, then the address that reached the address bus would actually be 010020. It should now be obvious why user programs cannot reach KERNAL area, because even if the user program tries to address 000000, it will really reach 000000+ the 8212 contents. Each second the interrupt circuit resets the F/F out port, and the 74157 gates out unaltered physical address to service KERNAL routines. The responsibility of saving the USER register contents rests in the KERNAL interrupt routines.

By offsetting the low order byte, the user and KERNAL modes can have segmented I/O.

If anyone implements this idea, please let me know.

Ed Comer
4818 Ferncrest Dr.
Greensboro, NC 27410
(919) 294-0147

1-1569521285
2-2153285
3-3587563465
4-4751465
5-521696359465
6-6359465
7-7523547465
8-8957285
9-9541589285
12-1417285
13-153285
14-47523547465
15-141589285
16-158957285
17-751465
18-156359465
19-141796359465
23-21569521285
24-41589285
25-213465
26-23547465
27-756359465
28-213468153285
29-954751645
34-358957285
35-323547465
36-3231465
37-787932153825
38-354751465
39-359465
45-471285
46-471826751465
47-7879465
48-87563465
49-952153285
56-639285
57-787563465
58-897465
59-969521285
67-752153285
68-69521285
69-63587563465
78-89541589285
79-957285
89-9693825

NEWS FROM EE TIMES ON TV GAMES

The Monday, February 7, 1977 issue of ELECTRONIC ENGINEERING TIMES has a number of interesting articles. One Page 1 is an article on low-cost video home computers such as the Commodore PET, with a projected price of under $500. National also plans to enter the games market with a $300 machine.

Page 2 describes I Corporation's resistive input pad capable of performing functions similar to a dual-potentiometer joystick. I Corporation can be contacted at 906 Treat Avenue, San Francisco, CA 94110 (415) 285-7922.

On Page 20 is an interesting discussion of National's new MM5789 monochrome TV game chip and the new MM57100 chip intended for color TV use. Both chips provide tennis, handball and hockey.

Another article on Page 40 of the same issue outlines current trends in the TV games industry.

Page 43 reports the entry of RCA into the TV games field with the COSMAC-based Studio II home TV programmer, which includes 5 games and is said to accomodate ROMs for other games or educational programs. The 5 standard games are bowling, auto racing, patterns, doodles, and a math game.

Jim Day February 8, 1977

High Speed Interaction Without Interrupt or DMA

Simple Floppy Disc Interface Detailed as an Example

(reprinted with permission from *Electronic Design*, Volume 23, Number 23)

by Eugene Fisher, design engineer, Lawrence Livermore Labs, L-403, Box 808, Livermore CA 94550

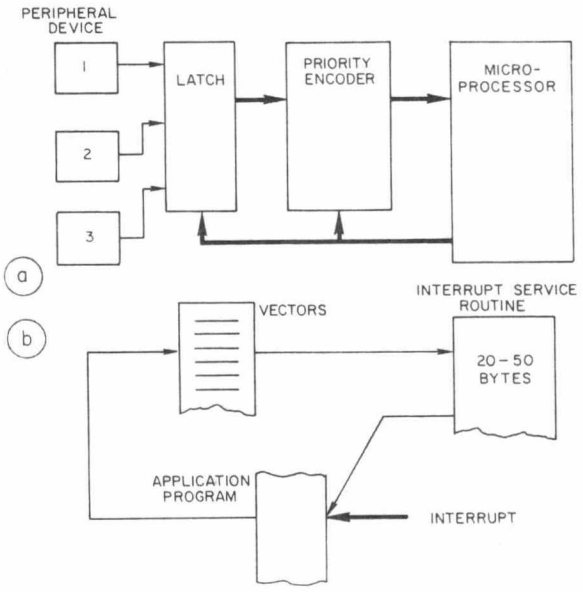
Microcomputer designs don't require techniques based on minicomputer hardware to achieve high speeds. Though designers often employ speed-enhancing interrupt or direct-memory-access (DMA) techniques borrowed from the mini world, the same improvements can be obtained far more simply. All that's needed is a microprocessor whose operations can be suspended readily during data transfers. Microprocessors like Intel's 8080 permit just this kind of solution.

When applied to an 8080-based floppy-disc controller, the simple approach requires only about 20 ICs. By comparison, an interrupt structure uses 40 to 80 ICs, and a DMA approach needs 80 to 100. Moreover the 8080 exercises full format and timing control over floppy-disc functions, with all of the algorithms contained in less than 512 words of ROM.

The alternatives

Interrupt structures often require hardware that is external to the microprocessor, as well as special programming (Fig. 1). Further, high-speed applications can easily require all of the microprocessor's real-time capability. For this reason, other interrupt sources usually must be disabled during the execution of the high-speed routine. Also, it may be necessary to save—in advance—sensitive registers to increase the speed of the interrupt service routine.

After an interrupt in the application program, a series of vectors (either hardware or software) transfer program control to the interrupt-service routine. This routine is the actual program that performs the synchronized I/O transfer. After the servicing of the interrupt, control returns to the application program to prepare again for the next high-speed interrupt service. As many as 100 bytes of code may be needed to handle the high-speed interrupt. This doesn't include the code for standard I/O that would be required without interrupt synchronization.



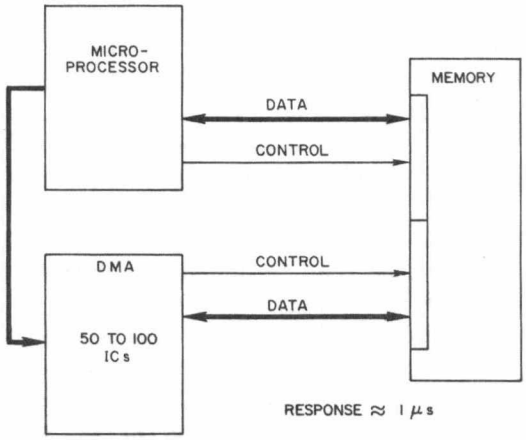
1. **Interrupt techniques may entail the addition** of such external circuits as storage latches and priority encoders (a). Typical response time is 32 μs. However, the major portion of an interrupt structure resides in software (b). Interrupt techniques can achieve typical data rates of about 31 k bytes per second.

Debugging of this interrupt service is extremely difficult, since it cannot be effectively simulated on a separate computer. In fact, debugging is a very time-consuming effort, requiring the use of all of the system's hardware or a trial-and-error method using oscilloscopes and other external hardware. This difficulty alone has discouraged designers. The typical speed for an interrupt interface is approximately 31,000 bytes per second.

Unlike the interface for a high-speed interrupt, the direct-memory-access interface isn't really part of a microprocessor or microcomputer system. It actually is a separate hardware controller that communicates directly with the computer memory (Fig. 2). The only function a microprocessor has in these cases is to initialize the data channel.

Once DMA has begun, the microprocessor must be able to disconnect itself from the associated memory, so as not to inhibit the transfer. Thus

debugging focuses on the hardware, because the only software functions employed are those needed for DMA-channel controls. The typical data-transfer rates are limited primarily by the memory. For example, memories for most single-chip MOS processors operate at 415 ns to 1 μs. So the data rate can be in excess of 100-k bytes per second.



2. **Direct-memory access, or DMA, requires a separate,** external controller that communicates directly with memory. When built with standard ICs, the controller can use up to 100 circuits.

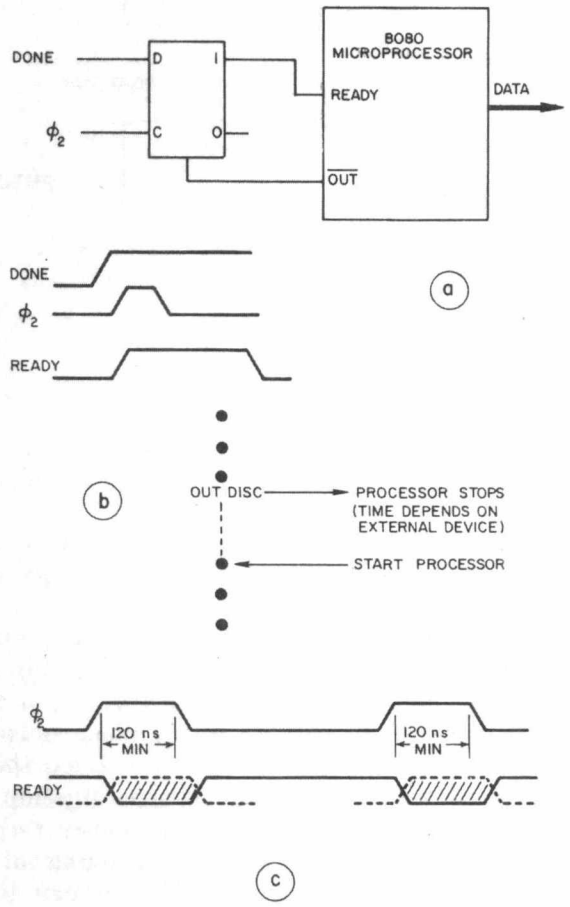
A simple solution

An alternative approach—one that requires the μP to halt for each transfer—takes full advantage of the processor. The technique can be applied to any microprocessor that has Ready-line synchronization or otherwise allows the stopping of the processor for data transfers. The hardware to implement this synchronization appears in Fig. 3. The block diagram shows that a simple, external flip-flop can synchronize the microprocessor.

The circuit operates as follows: An I/O instruction—in this case an output from the microprocessor—clears the flip-flop, halting the processor with the I/O data available on the microprocessor's data bus. The external device takes the data presented by the microprocessor and then returns a signal. Called DONE, the signal is used by the synchronizing flip-flop to raise the Ready line, letting the microprocessor continue. The microprocessor responds to an external input within one cycle—500 ns in our case.

Not only is this interface simple, but the microprocessor worst-case response is 1 μs. This response time is comparable to a DMA transfer, but with none of the associated hardware complexity. And the software is extremely simple; no difficult timing loops are required.

The output instruction actually stops the proc-

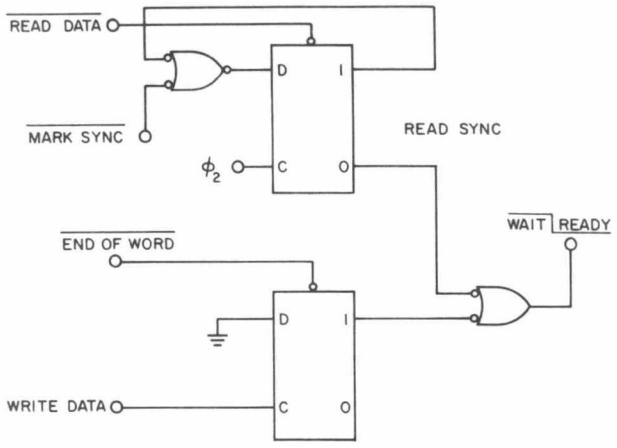


3. **The alternative to DMA and interrupt techniques** employs a microprocessor's Ready line (a) to suspend the processor's operation during data transfers (b). The Ready line is sensed by the microprocessor during phase two of state T_2 or T_W, the Wait state (c). This straightforward approach has a response time of 1 μs, and it yields data rates of 62-k bytes per second.

essor for a time that depends on the external device. If the microprocessor is already running at 100% utilization, there isn't time for other operations anyway. Thus the time lost by halting the processor is of no real consequence. And now software for the output or input is merely a standard I/O operation, with synchronization and data timing taken care of automatically. The data rate is 62-k bytes per second—a program-execution limitation.

Employing the Ready line

The Ready line on the 8080 processor was designed primarily to interface the processor to a slow memory or a slow I/O operation. But while the processor is stopped, the data bus is present on the processor's data lines. On an input instruction, the processor takes the data within 500 ns after starting or after the assertion of the Ready Signal. After the Ready line is raised,



4. **The synchronizing circuit for the Ready line** employs two flip-flops, one for the output and one for the input.

the microprocessor responds within the next clock cycle. Thus over-all response is about 1 μs when a 500-ns clock is used with the 8080.

A typical Ready-line synchronization circuit appears in Fig. 4. Two synchronizing flip-flops are used, one for input and one for output from the peripheral. In both cases all synchronization is handled by an external flip-flop. Even the requirement for an external D-type flip-flop has been eliminated by a new clock driver recently announced by Intel. The synchronizing of the Ready line can be taken care of by the new IC.

Programming constraints

The program must be able, of course, to accept synchronization by the external hardware. In the sample program (Fig. 5) the timing of each instruction in a critical loop has been calculated. This is necessary to ensure that the next I/O instruction is asserted in time. The routine allows the maximum data-transfer rate for the 8080 by employing the processor's stack pointer.

By adding the two halves of the loop, we see that the maximum time between instructions is 16 μs. Also the response to an external stimulus is 1 μs. And the data rate for this interface is twice that possible in any equivalent interrupt-driven interface.

Specifying the interface

Our floppy-disc application employs the IBM format (Fig. 6). Note the different types of sector information. Each has a unique indicator —a Mark—for Index, Data, Deleted Data and Address. A Mark doesn't contain a full set of clock pulses, and the missing clocks and Mark words, form synchronizing elements. Another critical item is the cycle redundancy check (CRC) character. This appears at the end of each data sector. It must be read to determine if there has been an error in reading or writing.

The data rate in this format is 4 μs per data bit, with the clock pulses coming between each data pulse (Fig. 7). The data rate of the standard floppy disc is 4 $\times$ 8 bits/byte, or 32 μs per 8-bit byte. However, to eliminate any possibility of timing problems, the interface is designed for twice that rate, or 16 μs per 8-bit byte—an effective transfer rate of 62-k bytes per second. And to eliminate the need for double buffering, the previously collected data word is read before the next clock pulse occurs. This results in the 2-μs interval at our double speed.

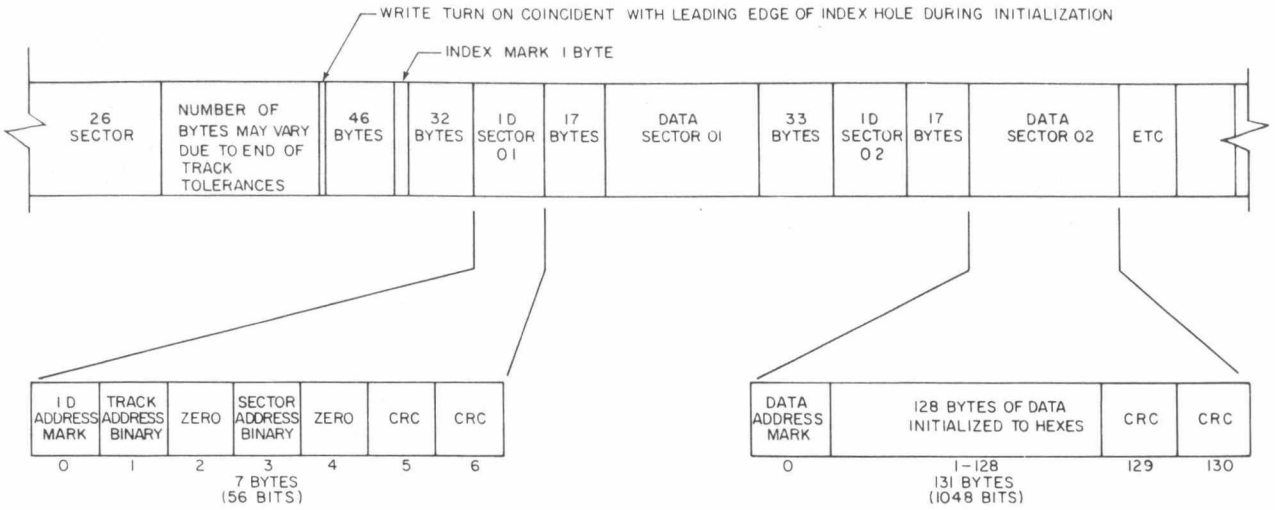
Solving interface problems

The "missing"—but implied—clock signal is detected by a retriggerable, monostable multivibrator. With the aid of a separate clock signal from the floppy disc (Fig. 7b), the detected signal is then used to synchronize an 8-bit register and to generate an end-of-word signal.

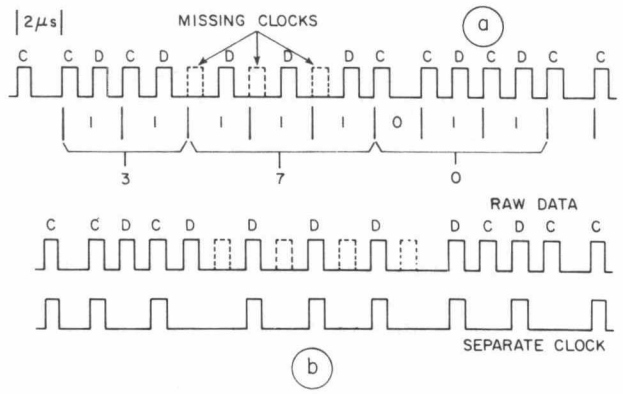
A 4-bit counter (Fig. 8) regenerates the missing pulses. The microprocessor's crystal clock

```
005161   333 011          ROVR:   IN DATAR          :READ ONE WORD—5.5
005163   157                      MOV L-A          :SAVE -2.5
005164   015                      DCR C            :BUMP CNTR -2.5
005165   345                      PUSH H           :STORE TWO WORDS -5.5
005166   333 011          DATCOL: IN DATAR          :NEXT—AND CRC WORD 1 -5.5
005170   147                      MOV H-A          :SET UP WORD—2.5
005171   302 161 012              JNZ ROVR         :AGAIN—5
005174   333 011                  IN DATAR         :CRC WORD 2
005176   333 011                  IN DATAR         :CRC DELAY WORD
005200   333 007                  IN STAT          :CHECK STATUS
005202   027                      RAL              :SET UP FLAG
005203   052 301 014              LHLD TEMP        :GET THE STACK POINTER
005206   371                      SPHL             :RESTORE SP
005207   332 217 012              JC CRER          :CRC ERROR
005212   311                      RET              :WE MADE IT
```

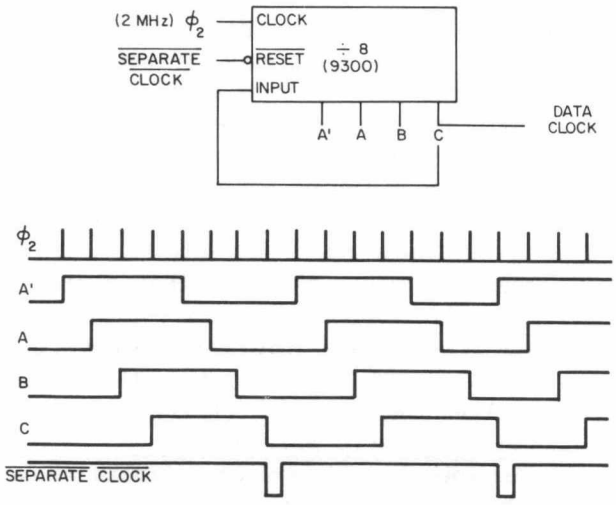
5. **The time of each instruction in a critical loop** must be calculated to ensure that I/O instructions are asserted soon enough. A portion of the read routine lists these times in microseconds.



6. **An IBM data format applies to our floppy-disc** example. Different sectors of information use an indicator known as a Mark. It relies on a series of implied—but not present—clock pulses for synchronization.



7. **The Data Mark has a transfer rate of 2 μs** per bit (a), or twice that of the usual format. This eliminates possible errors and the need for double buffering. The missing clock pulses are generated with the aid of a separate clock signal (b) from the floppy disc.



8. **A simple 4-bit counter,** synchronized to each separate-clock pulse from the floppy disc, regenerates the missing clocks.

provides the time base for the counter and thus, the floppy disc.

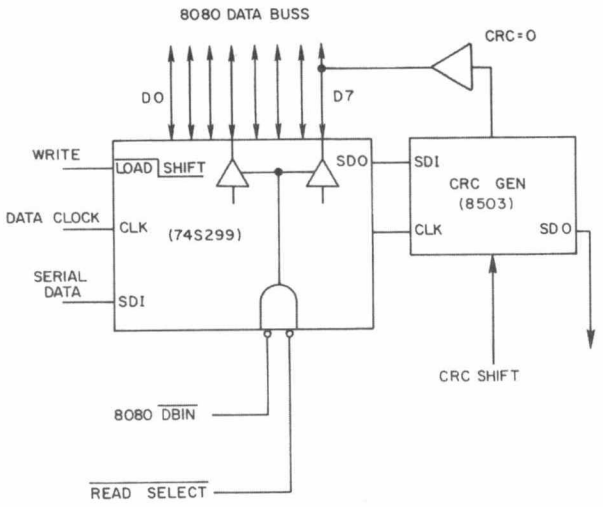
The counter is synchronized with the data received from the floppy disc, thereby providing a clock signal that doesn't have missing pulses. This counter and the crystal clock also generate the accurate timing pulses required for the Write operation to the floppy disc.

A single IC—the 8-bit register—handles both the serialization and deserialization of the signal from the floppy disc to the microcomputer (Fig. 9). This register has a common I/O; the same eight pins are used for parallel data input and output. Three-state data lines are compatible with the 8080 microprocessor data bus.

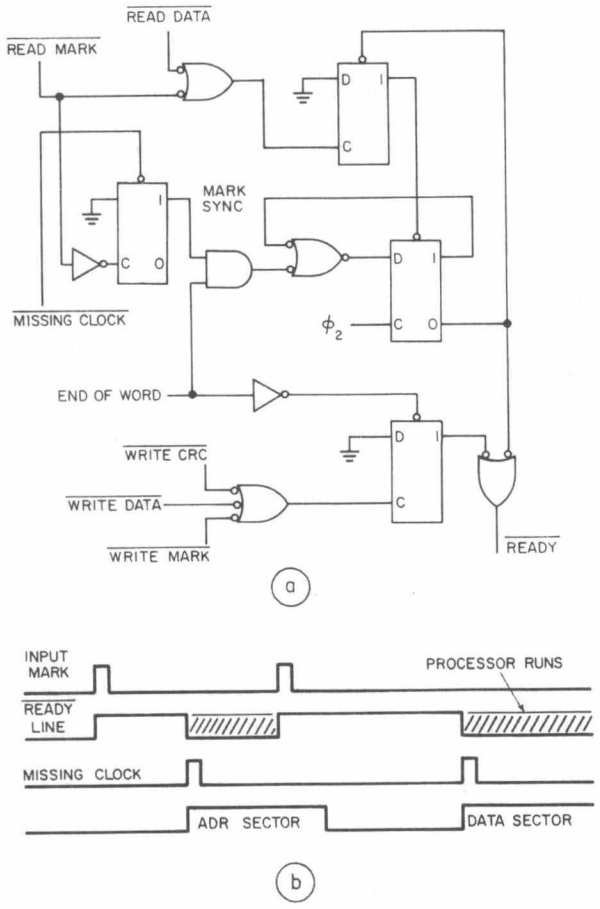
The Ready-line synchronizer for our application appears in Fig. 10. To detect a Mark word, the microprocessor sends the input signal called $\overline{\text{READ MARK}}$. This signal sets the flip-flop called Mark Sync, which suspends microprocessor operation until a "missing" clock signal occurs. When missing pulses are detected, the synchronizing flip-flop resets, allowing the processor to read the Mark word collected.

At this time the microprocessor verifies that the correct Mark word has been read. If it is not the correct word, the microprocessor loops back, sending out the $\overline{\text{READ MARK}}$ signal. Again, the processor stops until the next Mark word is received. The timing diagram shows an input Mark signal being sent two times. The first time it detects the address sector and the second time a data sector.

After the Mark word has been read, the succeeding data words must be read to verify the data format, the data transfer or the address sector. Therefore a second input signal called $\overline{\text{READ DATA}}$ is sent in a manner similar to that of the Mark operation just described. However, the input-data synchronizer resets at the end of



9: **An 8-bit shift register,** synchronized by the "missing" clocks, provides the parallel-to-serial conversion of the signal from the floppy disc. The second IC generates the cycle-redundancy-check character.



10. **The Ready-line synchronizer** for the floppy-disc application employs only four flip-flops and a few gates (a). The input Mark signal is sent twice (b) to detect address and data sectors.

each data word rather than at the end of a Mark word. The processor reads the data word at the end of each serial string in less than 1 μs, giving a 100% safety factor even for a floppy disc having twice the usual storage capacity.

The lower portion of the synchronizer controls Write operations for the Ready line. Three types of Write signals require synchronization: Write Mark, Write Data and Write CRC. Write Data simply writes the next 8-bit data word from the microprocessor into the shift register at the appropriate time. The Write Mark signal loads an 8-bit shift register (not shown), and this generates the missing clock lines for the writing of a Mark word.

Also on this shift register, Data Bit 6 is provided for one of the inputs, thereby allowing the software to create any of the three missing clock words required. The Write CRC command converts the CRC generator from a coded mode to a 16-bit serial shift register. The latter shifts out the collected CRC data word.

The control of the floppy disc functions—such as Head Load, Head Step In and Head Step Out —are handled in the usual I/O fashion, with flip-flops set to perform the required functions.

Key portions of the software for the floppy-disc application appear in Fig. 11. The first routine verifies that the floppy-disc head is actually on the correct track. The sequence is as follows: Wait for an address Mark; read the rest of the address sector; save the critical words of interest, the track and sector addresses; verify that there are no errors; then drop into a routine that steps the head to the desired track. Note that no critical timing loops are required. The only requirement is that the program return within 16 μs, so it can read the next data block.

Fig. 11b shows the code used to read a data block from the floppy disc and to store it in memory at the maximum transfer rate. This routine uses the processor's stack pointer to store 16 bits or two bytes per operation (thereby transferring data at 16 μs per byte). Note that this code lists timing intervals in the comment, so the programmer can actually count the instruction times that will result.

Fig. 11c shows the code that writes a data file on the floppy disc. This routine is similar to the previous one. However, a preamble of zero words must be written before the actual data file. Also, the last CRC character must be followed with a zero word. ■■

11. **Excerpts from the program,** compiled by an 8080 macroassembler, illustrate the three major routines. The first, a search routine, checks to see that the disc head is properly positioned (a). The next routine reads a block of data in the disc and stores it in memory (b). The third routine writes data into the floppy disc (c).

```
004403  365          STRT:   PUSH PSW        :SAVE A REG
004404  323 012              OUT HDDN        :LOWER HEAD
004406  315 222 011          CALL LDDLY      :WAIT FOR HEAD
004411  176                  MOV A - M       :TRK ADR
004412  376 112              CPI 112Q        :TEST FOR TOO LARGE ADR
004414  322 147 011          JNC NOADER       :ADR ERROR—TOO LARGE
004417  315 242 011          CALL SEEK       :STEP TO TRK
004422  315 222 011          CALL LDDLY      :WAIT TO SETTLE
004425  333 010              IN RDMRK        :SYNC AFTER STEP
004427  333 010      TSTRD:  IN RDMRK        :READ MARKS
004431  376 376              CPI 376Q        :WAIT FOR ADR
004433  302 027 011          JNZ TSTRD       :LOOP
004436  333 011              IN DATAR         :TRK ADR
004440  107                  MOV B - A       :SAVE TRK ADR
004441  333 011              IN DATAR         :ZEROS
004443  333 011              IN DATAR         :SECTOR ADR
004445  333 011              IN DATAR         :ZEROS
004447  333 011              IN DATAR         :CRC
004451  333 011              IN DATAR         :CRC
004453  333 011              IN DATAR         :CRC DELAY WORD
004455  333 007              IN STAT         :CHECK CRC
004457  027                  RAL            :SET UP FLAG
004460  332 165 011          JC ERROR         :CRC ERROR                    (a)
004463  072 300 014          LDA ADR         :GET EXISTING ADR
-------------------------------------------------------------------------------
                     :ROUTINE TO READ A DATA SECTOR
005064  333 010      READ:   IN RDMRK        :ENTER SECTOR ADR IN B
005066  376 376              CPI 376Q        :LOOK FOR ADR MARK
005070  302 064 012          JNZ READ        :LOOP BACK
005073  333 011              IN DATAR         :SKIP TRK ADR
005075  333 011              IN DATAR         :SKIP ZERO WORD
005077  333 011              IN DATAR         :READ SECTOR ADR
005101  270                  CMP B           :TEST SECTOR
005102  302 064 012          JNZ READ        :WRONG SECTOR—LOOP
005105  333 011              IN DATAR         :ZEROS
005107  333 011              IN DATAR         :CRC
005111  333 011              IN DATAR         :CRC
005113  016 013              MVI C . 13Q     :SET UP JUNK-IN-GAP COUNTER
005115  333 011              IN DATAR         :CRC DONE GAP WORD #1
005117  333 007              IN STAT         :CHECK CRC
005121  027                  RAL             :SET UP FLAG
005122  332 217 012          JC CRER         :CRC ERROR
005125  333 011      CRAPLP: IN DATAR         :SKIP THE JUNK-IN-THE-GAP -13 WORDS
005127  015                  DCR C           :DONE? -2.5
005130  302 125 012          JNZ CRAPLP      :NOPE LOOP BACK -5
005133  041 000 000          LXI H - 0       :CLR H AND L
005136  071                  DAD SP          :ADD STACK POINTER
005137  042 301 014          SHLD TEMP       :SAVE SP
005142  016 100              MVI C - 1000    :SET DATA CNTR
005144  061 000 014          LXI SP - BUFF   :POINT TO BUFFER
005147  333 010              IN RDMRK        :WAIT FOR DATA
005151  376 373              CPI 373Q        :TEST FOR DATA MARK -3.5
005153  302 213 012          JNZ CRERX       :NOT A CRC ERROR BUT GET OUT FOR NOW -5
005156  333 011              IN DATAR         :FIRST WORD -5.5
005160  147                  MOV H - A       :SET UP -2.5
005161  333 011      ROVR:   IN DATAR         :READ ONE WORD—5.5
005163  157                  MOV L - A       :SAVE -2.5
005164  015                  DCR C           :BUMP CNTR -2.5
005165  345                  PUSH H          :STORE TWO WORDS -5.5
005166  333 011      DATCOL: IN DATAR         :NEXT—AND CRC WORD 1 -5.5
005170  147                  MOV H - A       :SET UP WORD—2.5
005171  302 161 012          JNZ ROVR        :AGAIN—5
005174  333 011              IN DATAR         :CRC WORD 2
005176  333 011              IN DATAR         :CRC DELAY WORD
005200  333 007              IN STAT         :CHECK STATUS
005202  027                  RAL            :SET UP FLAG
005203  052 301 014          LHLD TEMP        :GET THE STACK POINTER
005206  371                  SPHL            :RESTORE SP
005207  332 217 012          JC CRER         :CRC ERROR
005212  311                  RET             :WE MADE IT                   (b)
005213  052 301 014  CRERX:  LHLD TEMP        :GET THE STACK POINTER
005216  371                  SPHL            :RESTORE SP
-------------------------------------------------------------------------------
005024  174          OVR:    MOV A - H       :SET UP NEXT WORD -2.5
005025  323 014              OUT DATAW        :WAIT FOR DISC—5.5
005027  341          DATWT:  POP H           :GET TWO WORDS -5
005030  175                  MOV A - L       :NEXT WORD -2.5
005031  323 014              OUT DATAW        :WAIT FOR DISC—5.5
005033  015                  DCR C           :BUMP CNTR—2.5
005034  302 024 012          JNZ OVR         :OVERS—5
005037  174                  MOV A - H       :SET UP LAST WORD
005040  323 014              OUT DATAW        :OUT LAST WORD
005042  323 016              OUT CRC         :SHIFT OUT CRC
005044  323 016              OUT CRC         :IBID
005046  257                  XRA A           :CLEAR THE A
005047  323 014              OUT DATAW        :WRITE OFF —LAST WORD IS ZEROS
005051  052 301 014          LHLD TEMP       :GET SP
005054  371                  SPHL            :RESTORE SP                   (c)
005055  311                  RET             :FAREWELL
```

MODIFICATIONS TO MICROSOFT 8K BASIC 3.1 FOR CROMEMCO Z-80.

ADDR	OLD	NEW	ADDR	NEW	ADDR	NEW
096B	3A	CD	7FE0	CD	7FF0	7D
096C	19	E8	7FE1	EC	7FF1	EA
096D	02	7F *	7FE2	7F *	7FF2	F9
096E	8F	00	7FE3	C0	7FF3	7F *
			7FE4	3C	7FF4	E6
12D1	C0	C3	7FE5	C3	7FF5	04
12D2	3C	E0	7FE6	EC	7FF6	C3
12D3	C9	7F *	7FE7	7F *	7FF7	FB
			7FE8	3A	7FF8	7F *
			7FE9	19	7FF9	F6
			7FEA	02	7FFA	04
			7FEB	8F	7FFB	6F
			7FEC	E5	7FFC	E5
			7FED	F5	7FFD	F1
			7FEE	E1	7FFE	E1
			7FEF	A7	7FFF	C9

* TO RELOCATE PATCH TO THE TOP OF YOUR MEMORY, CHANGE THESE BYTES AND LOAD PATCH THAT STARTS AT 7FE0 TO START AT THE TOP OF YOUR MEMORY. FOR EXAMPLE, IF YOU HAVE 12K OF MEMORY CHANGE THESE BYTES TO 2F AND LOAD PATCH AT 2FE0.

NOTE: WHEN RESPONDING TO THE QUESTION 'MEMORY SIZE?', BE SURE TO ANSWER 32 BYTES LESS THAN YOUR MEMORY SIZE, TO ALLOW ROOM FOR THE PATCHES. FOR EXAMPLE, IF YOU HAVE 12K OF MEMORY ANSWER WITH 12256 INSTEAD OF 12288.

$10 CASSETTE-ORIENTED 8080 ASSEMBLER

November 23, 1976

RO-CHE SYSTEMS announces a new 8080 CPU Assembler for use with their Multi-Cassette Controller. This new assembler (Assem 1.0) now makes it possible to assemble source code programs larger than memory.

The Multi-Cassette Controller allows an 8080 computer to read or write individual records to any of four software selected recorders, rather than blocks of data. Normally with a two-pass assembler the source code from cassette tape record by record and builds a symbol table during the first pass. During the second pass the source is read again and the object code is created and stored in memory. Upon completion of the second pass the assembled program may be executed or dumped to cassette.

When using the RO-CHE Assembler it is no longer necessary to limit the program comments (so necessary six months later) because comments never take up memory space as with a conventional assembler.

Assem 1.0 is available from RO-CHE Systems, 7101 Mammoth Avenue, Van Nuys, CA 91405, for only $10.00. Included is the Assem 1.0 and line editor on cassette tape with an instruction manual.

MICROPROCESSOR CROSS-ASSEMBLERS THAT RUN ON THE PDP8.

RENO, NEVADA October 1, 1976

SIERRA DIGITAL SYSTEMS has introduced a series of microprocessor cross-assemblers which run on Digital Equipment Corporation PDP8 minicomputers. According to a company spokesman, the group of cross-assemblers called the X8 series (pronounced Cross-8), fills a software gap experienced by many PDP8 users involved with microprocessor program development. The cross-assemblers run in 8K of memory under the OS/8 operating system.

X8 assemblers feature a Universal Assembler Format of common assembler directives and techniques. Pseudo-ops and run-time options provide for conditional assembly and extensive listing control. Arithmetic and logical operators are supported in complete operand expression evaluation. Generated object code may be output in the microprocessor's standard loader format or BNPF format for ROM generation.

Immediately available are X8 series assemblers for the 6502, 6800, and 8080 microprocessors. Each X8 assembler is priced at $400 and distributed in PDP8 binary format on papertape, Dectape, or DEC floppy diskette. Cross-assemblers for the 2650 and F8 microprocessors are also near completion. Contact: SIERRA DIGITAL SYSTEMS, 1440 Westfield Avenue, Reno, Nevada 89509 (702) 329-9548

OSI FLOPPY DISK PRICED AT $599

An under-$1,000 floppy disk for personal microcomputer systems is now available, making possible reliable random storage of from 252,314 to 504,628 bytes of data per disk at read/write rates of 100,000 baud and up.

For $599, Ohio Scientific Instruments is selling a fully-assembled diskette drive including read/write electronics, manuals, mating connectors, system interface board (bare) and 6502/6800 operating system, delivery guaranteed to be less than 120 days. Zilog, 8080 and probably PDP-8 systems can be expected later.

OSI claims it has made a price breakthrough by incorporating into the hobbyist's own microprocessor system the disk control functions and interface. Further savings were realized by making user options of the 24v-1A power supply, case, connector cables and components for the interface board.

User-supplied parts are conservatively estimated to cost an additional $145, bringing total system cost to as high as $744. Eight to ten evenings of assembly time and testing are suggested. These added costs are $50 each for interface parts and power supply, $25 for enclosure and $10 each for diskettes and cables/hardware.

It is widely held that mass storage of programs and data is currently the weakest area of hobby computing.

Everytime a computer "crashes", which may be frequent, both program and data may have to be reloaded. In the case of 12-K BASIC, this may take 20 minutes from paper tape, or almost 8 minutes at standard cassette tape speed – but just a couple of seconds with OSI floppy!

Putting such programs on non-volatile memory is one solution to the problem, yet EPROM from one of the leading systems-suppliers costs about $600 per 2-K card.

Not only is cassette tape slow, but it is "touchy". If more than one program is stored on a tape, one or more programs may be destroyed searching for one, since stopping the tape in the middle of a block of information can easily result in tape stretch and the loss of a vital segment of data.

Magnetic sorts, via tape, are thus at least theoretically risky – and further, require expensive, computer-controlled tape decks.

To maintain mailing addresses on tape, for example, the user would have to re-record a whole tape just to add, delete or correct one address. Three tape decks would be required (unless corrections were being made on-line, then two), one reading the old tape, the other reading new data, and the third recording the corrected version. The two tapes that were"merged" would be filed for two or three "update" generations, in case something went wrong in the correction process or the new tape is ruined.

Disk systems, however, can read from or write to any block of data located anywhere within their storage capacity. Only one machine is required.

Thus it is much easier to maintain a file on disk. Corrections can be handled instantly, on-line, and milliseconds later the updated file can be assessed by the system.

The OSI 470-Floppy-Disk/Modem board, which occupies one slot in the company's standard backplane, can control up to four disk drives. It also can be configured to provide a 600-baud modem with full dial-up and automatic answer capabilities.

With the OSI interface, all tasks that can be handled by software are, thus making it format-independent so that the system can read or write virtually any diskette format including the popular IBM 3740 format. Of course, appropriate software must be used for each format change.

Double-density recording can be used, but greater-than-1-MHz system-clocks are required unless a very simple disk format is utilized.

The Model 470 comes with a simple low-memory-overhead 6502 or 6800 operating system. Purchaser must specify which he wants.

The operating system allows the transfer of 4-K blocks of memory to and from the floppy's 77 tracks. To perform a write operation, the user specifies the starting address of the source block and track number. For a read operation, he simply specifies track number and starting address of the destination block.

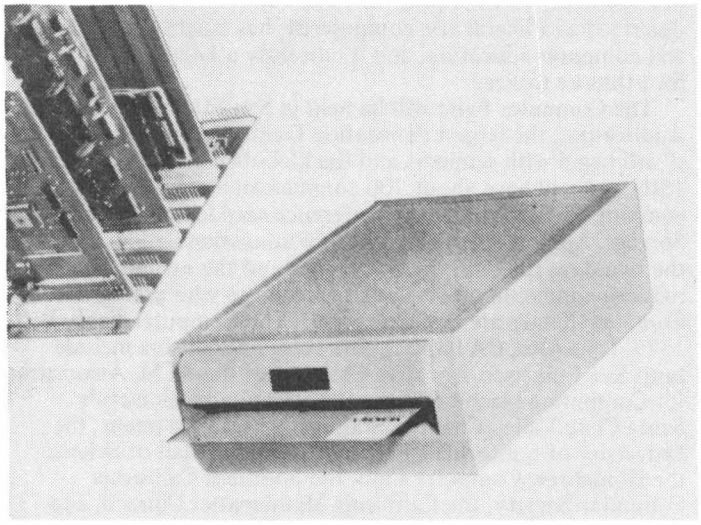
Memory is transferred in eight-bit segments, with one parity bit and one stop bit. Software automatically performs a write/read/verify test when writing and parity-tests when reading. The software will attempt to write three times before labeling a track unsafe and will attempt to read a track a user-specified number of times to recover data.

More sophisticated operating systems are being written for the 470, but this simplest one should allow the user to get his floppy up and running with minimum difficulty, using a small amount of memory and slow clocking.

OSI says its $535 price for diskette drive, manuals and connectors means slim profits and therefore orders must be in writing – prepaid via check, Master Charge or BankAmericard – or via 20 percent deposit for COD shipment. Purchase orders are not accepted. There is a 20 per cent restocking charge for cancellation or order unless delivery is delayed beyond 120 days after receipt of order.

Drives are manufacturer-guaranteed against defective workmanship for 90 days by General Systems International. Although the OSI interface is delivered with the GSI Model 105, it is compatible with the GSI 50 and 110 drives. OSI's $535 price is considerably less than GSI's 50-piece industrial price.

Ohio Scientific Instrument products are available in Southern California exclusively from Barefoot Computer Company, Box 158, San Luis Rey, CA 92068.



CLASSY CARDBOARD COMPUTER COVERS, JAMES ELECTRONICS PRAISE, & PAIN FROM ADVANCED MICRO–ELECTRONICS & SWTPC

Dr. Dobb's Journal etc., Dec. 30, 1976

Jef Raskin's "Cardboard Computer Company" – I've had the same problem (money), so for a couple of bucks, I got some walnut grained contact paper which I use to cover cardboard, plywood, or aluminum chassis enclosures. If you take a little time to smooth the wrinkles and air bubbles, you will have the "Classy Cardboard Computer Company". Oh yes, I really enjoy Jef's articles – wish he had written more about the S-100 bus earlier before I committed to a system that doesn't use said standard.

I also want to say a good word about James Electronics. I have sent various orders to James over the last year plus, and always have the parts in my hands within 7 calendar days of the time I drop the order in the mail box. (Considering the lousy postal service, that is *really* moving!) It doesn't make any difference if the order is simple (one chip) or complex (500 items – all different – totally $20), he delivers quickly, accurately, and as advertised. While you may save a few cents shopping elsewhere, we should support this supplier to let him know we appreciate service.

Noticed your mention of trouble with Advanced Micro-Electronics of IRVINE, CA. I ordered one chip almost a month ago (4 December) and haven't heard anything yet. (It took two months to get a spec sheet, for which I had sent $1, last summer.) I notice also, that AME is in violation of the California law requiring mail order firms to list a street address in their advertisements – he even uses a P.O. Box in his catalog. Does he have something to hide? **[AM-E has been offered the opportunity to tell its side of the story in a past complaint, and has not chosen to offer ANY reply.]**

Am in process of writing a mini-book about my troubles with a SWTPC 6800 – and SWTPC. Will send you a synopsis when I get the problem resolved – at present rates, I'd estimate that at July or August. In the meanwhile, I recommend the purchase of S-100 bus machines. **[Suggest readers take this with a wait-and-see attitude. Look for Mr. Goldsmith's complaint *and* SWTPC's reply, before passing judgment.]**

Yours,

W. B. Goldsmith, Jr. 4724 Maybank Avenue
Lakewood, CA 90712
(213) 423-3411

WEST COAST COMPUTER FAIRE SCHEDULES PRESENTWORLD AND FUTUREWORLD BANQUET SPEAKERS

PALO ALTO, CA. – The First West Coast Computer Faire, a major convention exclusively devoted to the subject of personal and home computing, will include two unusual banquets. Each banquet will have a "presentworld" speaker and a "futureworld" speaker.

Frederik Pohl, the widely known science fiction writer and editor will be the futureworld speaker at the Friday evening banquet, April 15th. His topic will be, "Robots You Can Build for Fun and Profit." Aside from being a major scifi author, Mr. Pohl is also well known as a lecturer. He has spoken before the New York Academy of Science, NASA groups , the American Astronautical Society, the World Future Society, and many other groups.

Friday's present world speaker will be John Whitney, an internationally recognized pioneer in experimentation and research concerning computer graphics and film-making. His talk, "Digital Pyrotechnics: The Computer in Visual Art," will present a number of his ideas connecting the well known theories of musical harmony – audio art – with his developing theories of visual harmony – visual art. Mr. Whitney is a fascinating speaker, and will include exhibition of several of his most recent films in his talk. He has been working in this area for many years, including work supported by the National Endowment for the Arts, the Guggenheim Foundation, the Ford Foundation, and IBM.

The presentworld speaker at the banquet on Saturday, April 16th, will actually be a "pastworld" speaker. Dr. Henry Tropp will discuss the early days of computing in his talk, "The 1940's: The FIRST Personal Computing Era." For the past several years, Professor Tropp has been the Principal Investigator for the History of Computers Project, jointly sponsored by the Smithsonian Institution and AFIPS, the American Federation of Information Processing Societies. Dr. Tropp spent much of that time traveling nationally and internationally, conducting in-depth, half-day to week-long interviews with most of the pioneers in digital computing. He has a fascinating collection of stories and anecdotes from the first years of computers.

Saturday evening's futureworld speaker will be Ted Nelson, the author of *Computer Lib/Dream Machines* and Director of the Xanadu Electronic Literary Project. His talk, "Those Unforgettable Next Two Years," will discuss personal and home computing as it appears likely to evolve in the next several years. The flamboyant and inventive Mr. Nelson will focus primarily on some of the more esoteric products, projects, and applications that are likely to exist prior to 1980. Mr. Nelson, who might be described as a liberal arts computerist, has taught art, sociology and computer education, and is currently a Lecturer at Swarthmore College.

The Computer Faire will be held in San Francisco's Civic Auditorium, the largest convention facility in northern California It will begin with seminars and the kickoff banquet on April 15th, and will have about 200 commercial and homebrew computer exhibits and 100 conference sessions on Saturday and Sunday, April 16-17th. At the door admission will be $9 for the two days for both the conference and the exhibition. A reduced admission rate is available to those who preregister. Complete details are available from: The Computer Faire, Box 1579, Palo Alto, CA 94302. The Faire co-sponsors include both San Francisco Bay Area Chapters of the ACM, Association for Computing Machinery, the IEEE Computer Society's Santa Clara Valley Chapter, Stanford's EE Department, the University of California's Berkeley Lawrence Hall of Science, the Homebrew Computer Club, the Southern California Computer Society, the California Mathematics Council, and other groups.

CALL FOR PAPERS: UPIEEE-77 WORKSHOP, JUNE 1977 BENCH PROGRAMMING OF MICROPROCESSORS

You *can* incorporate microprocessors into products without the use of expensive microprocessor program development equipment. This workshop is soliciting papers from people who have been doing just that. Three types of papers are expected – software, hardware and trouble-shooting (debugging). All hardware papers accompanied by a working board will be published in the Proceedings.

In the 1950's the physicists gave us the transistor. Before we could conveniently use it in circuit design, we had to strip it of the theory with which the physicists had saddled it, and provide it with a practical engineering approach. Now the computer people have given us the microprocessor. Before we can conveniently use it, we have to strip it of the top-heavy computer approach and provide it with a practical engineering approach for use in circuit design.

The purpose of the UPIEEE-77 workshop is to exchange experiences and learn pitfalls and shortcuts in writing firmware; and in trading-off firmware, software and hardware right at the design bench. Papers are due before February 1st.

The workshop will take place on Friday, Saturday and Sunday, June 10, 11 and 12 in Philadelphia. Dormitory space will be available to out-of-town participants. Proceedings will be published beforehand and mailed to participants. There will be no reading of papers – only discussion, questions and answers, and demonstration of boards. Boards chosen for-publication-only will be displayed in the lobby, and authors will be given time for poster talks. Time will be allocated for bull sessions.

For further information about UPIEEE-77 contact Miss Helen B. Yonan, Philadelphia IEEE, Moore School, Univ. of Penna., Philadelphia, PA 19174, or telephone K.V. Amatneek, Chairman, (215) 448-7951 or leave message at (215) 233-2962.

ZAPP . . . CONVERTED TO 12K BASIC

Dear Dr., January 6, 1977

I have very much enjoyed the "Sapp the Moon Man" program which appeared in your Nov/Dec issue. I have converted it from CASUAL to MITS 12K BASIC, and have enclosed a listing. The program is set up for the standard VDM memory page address of CCOO (in line 3), and standard control port C8 (in line 8). In order to use this program with MITS 8K BASIC, the VDM must be set up to be addressed in the lower 32K of memory since the POKE command in 8K BASIC is limited to addresses less than 32768.

Sincerely,

Dan Elliott

2250 Latham St., # 23
Mountain View, CA 94040

ZAPP THE MOON MAN

```
3 Z=-13312
5 INPUT "HOW MANY BOLTS DO YOU WANT";S
6 IF S<1 OR S>99 THEN 5
7 FOR I=1 TO 16:PRINT:NEXT
8 OUT 200,0
10 U=Z+754:L=Z+864:M=L:P=Z+32:N=Z-32:J=Z+722
15 POKE Z,32:H=0:I=0:POKE L+128,160:L=L+128
20 D=Z+592:POKE D,90:POKE D+1,65:POKE D+2,80:POKE D+3,83
30 POKE D+31,66:POKE D+32,79:POKE D+33,76:POKE D+34,84
40 POKE D+35,83:POKE D+96,76:POKE D+97,69:POKE D+98,70
45 POKE D+99,84:S=S+1:GOSUB 400
50 B=0:X=Z:Y=Z+64
100 GOSUB 3000
110 IF INP(255)=0 THEN 100
120 GOSUB 400
130 POKE L+64,32:POKE L,4:L=L-64:GOSUB 3000
150 IF L<>N THEN 130
200 POKE P,32
210 IF B=0 AND (X-1=P OR X-2=P) THEN 240
212 IF B=1 AND (X+1=P OR X+2=P) THEN 240
220 IF S<1 THEN 5
230 GOTO 110
240 H=H+1:I=INT(H/10):POKE J+1,H-I*10+48:POKE J,I+48:I=H
242 D=Z+32:POKE D,90:POKE D+1,65:POKE D+2,80:POKE D+3,80
244 POKE D+4,33:X=Z:W=0
245 POKE X,32:X=X+1:W=W+1:IF W<>50 THEN 245
250 GOTO 50
400 S=S-1:L=M:V=INT(S/10):POKE U+1,S-V*10+48:POKE U,V+48
410 V=S:RETURN
3000 IF B=1 THEN 3900
3100 POKE X-1,32:POKE X,7:X=X+1:IF X<>Y THEN RETURN
3200 X=X-1:B=1:RETURN
3900 POKE X+1,32:POKE X,7:X=X-1:IF X<>Z THEN RETURN
4000 X=X+1:B=0:RETURN
```

$10 FOR 64 CHARACTERS & SCROLLING ON A TVT II

Dear Mr. Warren: January 27, 1977

I am enclosing details for a conversion I made on my SWTPC TVT-II to allow it to operate in a scrolling mode with 64 character lines. I've been using this feature on my TVT for over two months now and find it a marked improvement over the original. The modified terminal displays 16 types lines each with up to 64 characters. All data is entered along the bottom line in typical scroll fashion. Similarly, a line feed will cause all typed lines to move up one position on the screen while the top line "wraps" around to the bottom. The additional circuitry fits nicely on a 3" x 3" universal P.C. board which I have mounted on the TVT memory board with plastic stand-offs. All parts are readily available and the whole project should not run over $10.00.

The change divides nicely into two main sections and two options. Each is treated separately below. Should anyone have need for less than the total circuitry either section and the options can be used separately or in any combination.

SCROLLING CONVERSION:

In the unmodified TVT circuit, cursor line position information comes from up-down counter, IC-34. This then feeds IC-41 where it is compared with data on memory address A-5 thru A-8. Disconnecting this feed from IC-34 and placing the four "A" input lines of IC-41 at logic 1 will locate the cursor along line 16 at bottom line of the screen. Scrolling is then accomplished by inserting, essentially in series, a 7483, 4-bit binary adder, in memory address lines A-5, A-6, A-7 and A-8 as they feed to the 2102 memory IC's.

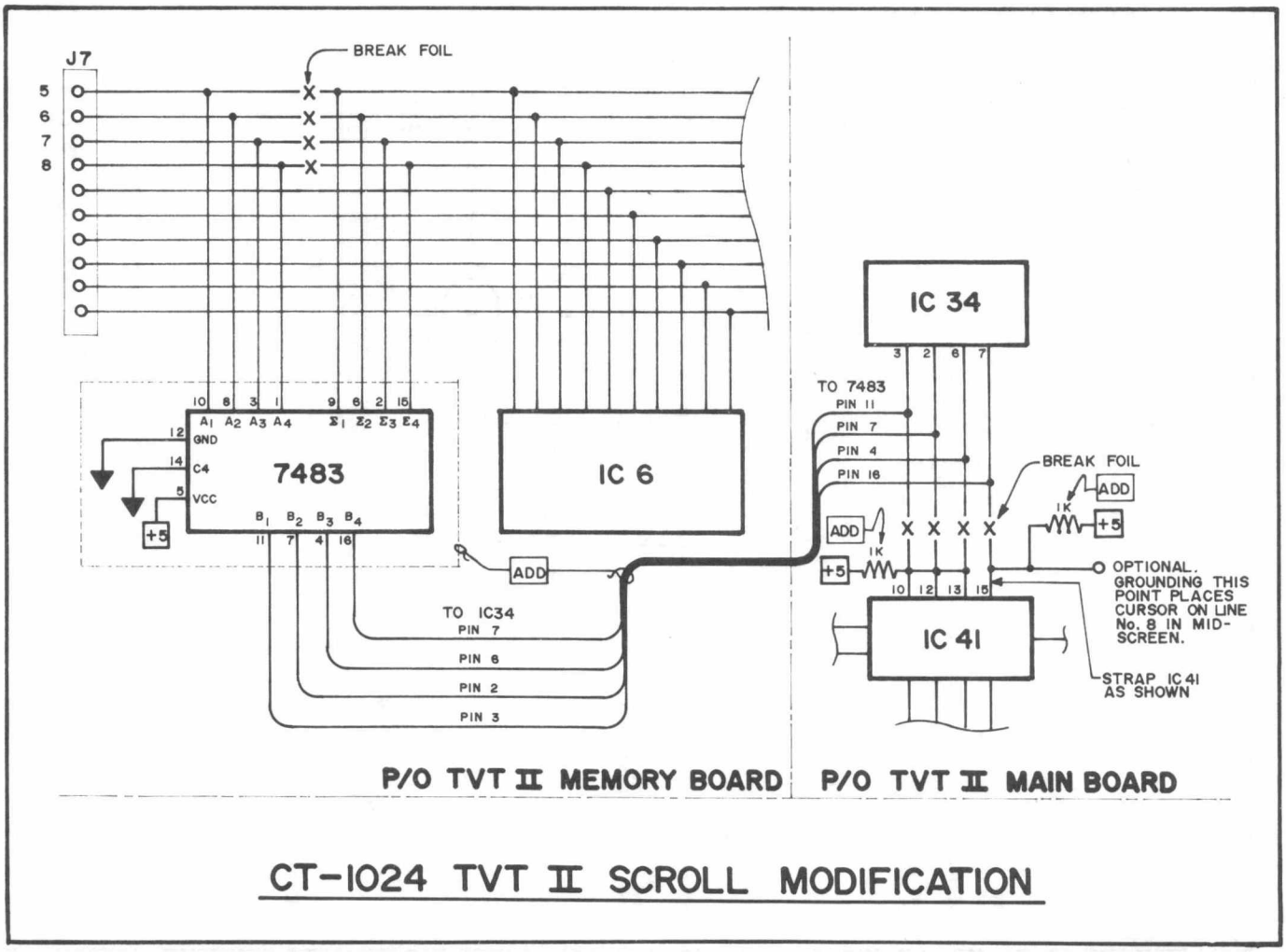
The four outputs of IC-34 are fed into the "B" inputs on the 7483 full adder and used to provide a binary "off set", shifting the TVT display up one line each time IC-34 is incremented or down if it is decremented. Since only 16 TV display lines are involved, one 4-bit adder is all that is necessary. The CARRY OUTPUT is ignored and input CO tied to ground.

64 CHARACTER CONVERSION:

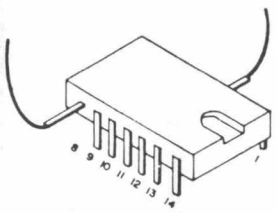
In order to generate 64 character lines, the dot generator, IC-18B, must operate at approximately twice its normal frequency. To do this capacitor C-4 was changed to 15 PF.

To address 64 characters 6 memory address bits are required. Five are provided for from the 32 character lines in the original TVT-II. Along with an extra memory address bit, one additional register to feed it is also needed. Fortunately, both of these are available in the TVT-II. One of the flip-flops in IC-14 is unused and with only slight wiring changes this fourth flip-flop can be used for the extra address bit. Although it's a bit unconventional, we can use the tenth memory address line, the page select (A-9), as the sixth bit memory address. This works well and with only a minimum of changes to the original circuitry.

Also required for the 64 character conversion is a means of keeping track of the cursor for characters 33 thru 64. One additional stage of up-down counting has been added to the cursor position registers just ahead of



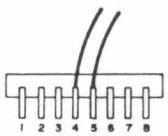
CT-1024 TVT II SCROLL MODIFICATION



NOTE:
DO THE SAME FOR PIN No. 3

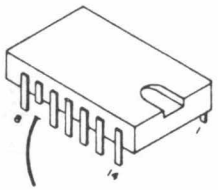
DETAIL (1)

CUT PINS FROM P.C. BOARD AND SOLDER NEW WIRE DIRECTLY TO THE CUT IC PINS.



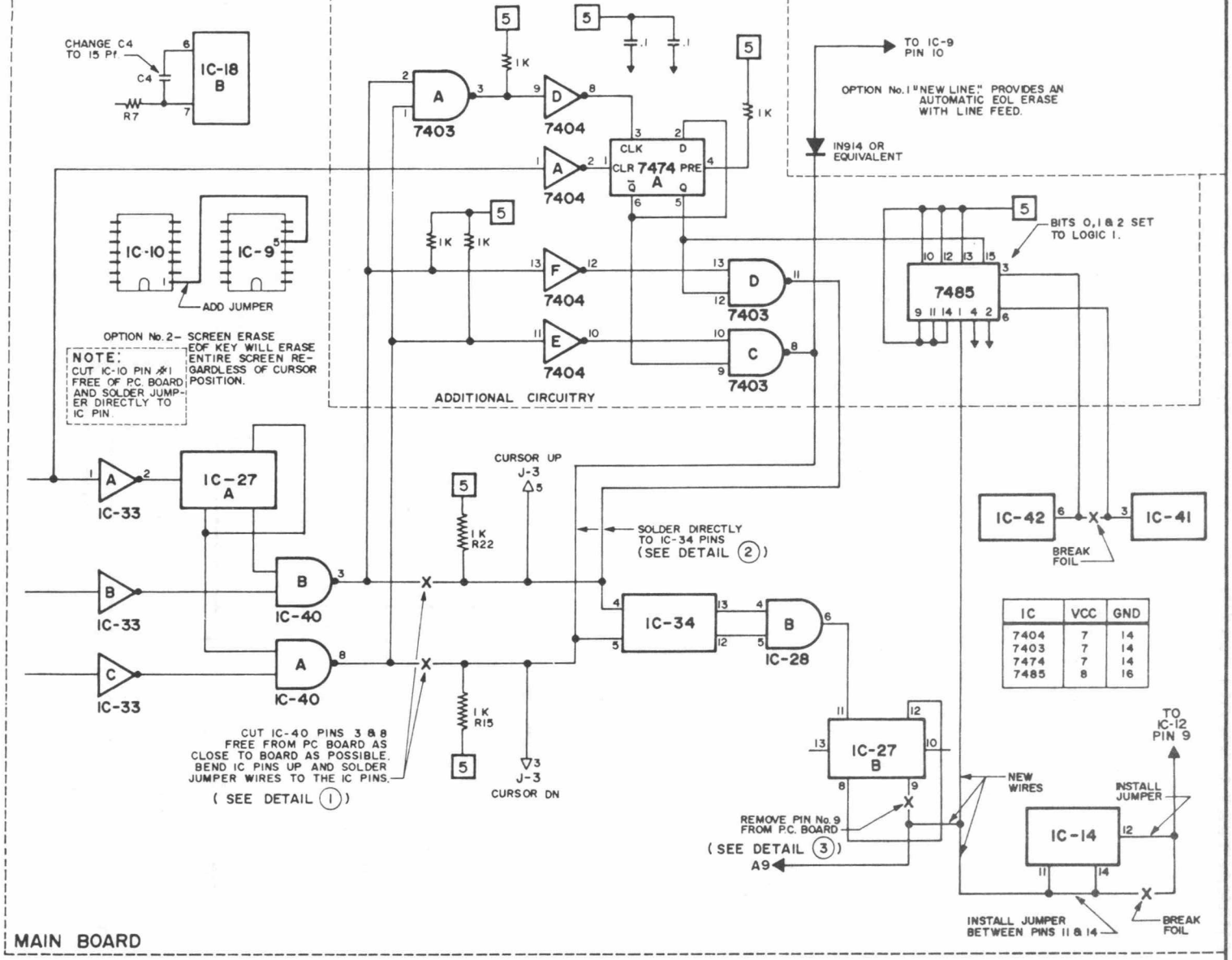
DETAIL (2)

SOLDER DIRECTLY TO IC-34, PINS 4 & 5 WITHOUT REMOVING PINS FROM P.C. BOARD.



DETAIL (3)

CUT PIN 9 AND SOLDER NEW WIRE TO PIN HOLE.



IC	VCC	GND
7404	7	14
7403	7	14
7474	7	14
7485	8	16

CT-1024 TVT-II 64 CHARACTER CONVERSION

IC-34 to take care of this. This extra counter uses essentially the same circuitry as the existing fifth bit counter. Along with the counter one additional comparator is needed to compare the counter output with the new memory address data from line A-9. A 7485 comparator has been added for this purpose. Only one of the four circuits on the 7485 is used. The others are all tied to the +5VDC. All comparators are connected in series in the TVT so this "extra" one may be tied in at any convenient point in the string.

OPTION NO. 1:

This option provides an automatic EOL erase on line 16 with a "line feed' so new lines will always come up "clean". No attempt was made to store data in the TVT-II beyond what is actually on the screen. Doing this will require additional memory.

OPTION NO. 2:

This option charges the "EOF" erase circuit to a "full screen" erase regardless of cursor position. Scrolling places the cursor on the bottom line which defeats the normal "Home-up", "EOF" screen erase procedure. The "Home-up" jumper on the comparator controlled cursor board should also be removed.

Many of the circuit elements needed for the project are available from unused IC sections on the TVT. However, using them would have increased the complexity of the wiring. Therefore, I chose to use separate IC's. Placing the extra circuitry on a small P.C. board makes the full conversion quite simple. A 3" x 3" "universal" PC board is adequate for the purpose. D.C. voltage was taken directly from the TVT's regulated +5VDC supply. The TVT lends itself well to modifications of this type and I expect there are several changes which have been made by others that we all could profit from. I would very much like to see what other readers are doing or have done with their TVT-II's.

Sincerely,
R. Lynn Smith
PSC No. 1, Box 7327
APO San Francisco, CA 96286

PARTS LIST

Item No.	Description	Qty.
1.	Scrolling Conversion 7483	1 Ea.
2.	64 Character Conversion	
2a.	15 PF Capacitor	1 Ea.
2b.	.1 Mdf Capacitors Cermic	2 Ea.
2c.	1000 Ohm 1/4 Watt Resistors	4 Ea.
2d.	7403	1 Ea.
2e.	7404	1 Ea.
2f.	7474	1 Ea.
2g.	7485	1 Ea.
2h.	IN914 or Equiv.	1 Ea.

These beautiful, immaculate, exquisite, masterfully executed perfectly produced, ever-so-wonderful schematics have brought joy to this old editor's heart. O...what great pleasure doth arise from well-done artwork. –EDITOR

ALTAIR DISTRIBUTION MOVES

January 21, 1977

MITS, INC.has decided to move its subsidiary, Altair Distribution Company, Inc., from 730 Boston Post Road, Wayside Plaza, Sudbury, Massachusetts 01776 to MITS headquarters at 2450 Alamo, S.E., Albuquerque, New Mexico 87106.

The transfer is in progress and is planned to be a smooth one. If you have any questions, please contact Mr. Robert Tinley, Vice President of Operations, or Mr. Robert Lindemuth, Vice President of Finance, at MITS in Albuquerque.

NEW SIGNETICS 8K PROM OFFERS HIGHER SPEED, 30% SIZE REDUCTION OF CHIP

SUNNYVALE, California – A field programmable 8192 bit bipolar Read Only Memory with a 2048 x 4 organization is now available in production quantities from Signetics.

One of the first 8K PROMs to be marketed in large quantities, the new Signetics device also offers improvements in speed, size and power requirements over similar devices announced by other manufacturers, according to Ralph Kaplan, Marketing Manager for Memories.

The TTL-compatible memory is available with either Open Collector outputs (82S184) or Tri-State outputs (82S185), allowing to optimal word expansion in bussed organizations.

Key performance specifications include an address access time of 100 nanoseconds maximum and typical power dissipation of 50 microwatts/bit.

The 82S184/185 devices also feature a significantly smaller chip size of 26,600 square mils, about 30% smaller than current industry standards for devices with this capacity, according to Kaplan.

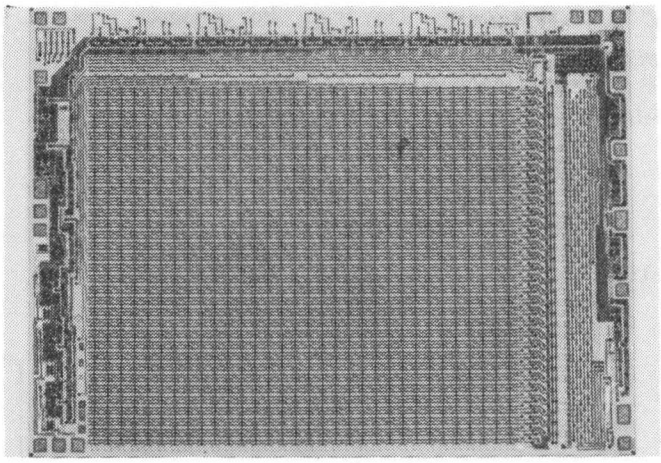
Both devices are available in a space-saving 18-pin ceramic DIP package and offer substantial savings in costs. Priced at $31.00 in quantities of 100 and up, they represent a cost/bit savings of approximately 35% compared with equivalent PROMs now on the market, Kaplan says.

The 82S184/185 devices are an extension of the Signetics generic family of PROMs, utilizing the same basic fusible-link programming techniques. They also represent a further extension of the company's line of PROMs with 4-bit word organization, which facilitates memory expansion.

Other features include on-chip address decoding and a chip enable input.

The 82S184 and 185 are currently available in the commercial temperature range (0 C to +75C), with military grade devices expected in the near future.

For further information contact Signetics, 811 Argues Avenue, Sunnyvale, California 94086. (408)739-7700.



SINGER/FRIDEN DOCUMENTATION

Dr. Dobbs & Readers:

I have schematics, wire lists, and operator instructions for Singer/Friden Key-to-Tape terminals that were widely sold on the S. Calif. surplus market. I charge $20.00, which is refundable if it isn't what you need. Send me your top assy number and plug-in card numbers.

John Clothier, EE
c/o Interstate Electronics
707 E. Vermont Ave.
Anaheim, CA 92803

64 CHAR/LINE TVT-II MODS, REQUESTS & SUGGESTIONS

Dear Dr. DJOCC&O, Dec. 29, 1976

First, praise for the excellent product review of the Poly 88 system. Could you arrange a comparison/evaluation of some floppy disc units (i.e. Digital Systems w/CPM), Processor Technology's Helios and the ICOM system. Many of us are waiting to buy a floppy storage system and at 2K dollars, purchasers are entitled to some of *DDJ's* expert evaluation.

Second, enclosed you will find an article that describes how the TVT-II can very simply be modified to display 64 characters/line. It requires only one IC as indicated and it works. It was originally developed in the Atlanta Area Micro Computer Club.

Third, I would like to compliment Marvin Wizenread on the interesting VDM software he has submitted to *DDJ*. I pose this challenge to his ingenuity – how about a PONG game?

PLEASE: ask him to document for a VDM at the Proctoc standard address of CCOOH-CFFFH, the UP and DOWN assembly language sequences of his programs.

I enjoyed Marvin's Life on a VDM but if "Life's Like That" having to enter the starting configuration from the front panel, then why bother to use a computer? Life can be done with graph paper, pencil and eraser. However, despair not, haters of front panel switches. I am writing an article that discusses humane ways of putting data in your machines.

I would like to suggest a new approach for *DDJ* to pursue in the area of software development. Till now, *DDJ* has set up specifications for languages that can be implemented by its readers. The response in the form of Tiny Basics, and other languages has been wonderful. I propose that *DDJ* publish specifications for applications programs that would serve as an impetus to program development as did the original specifications for Tiny Basic. For instance:

Applications Project I: Payroll

a) Language: ANSI Basic or ALTAIR 8K Basic V3,2

b) Minimal Hardware: ALTAIR/IMSAI computer
20K RAM memory
Tarbell cassette
ASCII keyboard and
VDM or other video display unit
ASR 33 or other hardcopy printing device

c) Objective: To provide payroll computations for a business with 15 to 20 employees. It should allow the preparation of payroll for hourly and salaried employees on a periodic basis. Data should be available as weekly summary sheets for all employees and also individual payroll files for each employee. The program should be able to produce quarterly and annual reports of information needed for filing with local, state and federal tax authorities.

That's all for the time being.

Sincerely,
Gary Alevy
1056 Fifth Avenue
New York, NY 10028

TVT-II 64 CHARACTER MODIFICATION

This modification converts the TVT-II display to a single page of 16 lines of 64 characters. Operation of the cursor board, screen read board, parallel or serial interface is unaffected. After conversion the TVT-II will operate as usual with the cursor homing after character 64 on line 16.

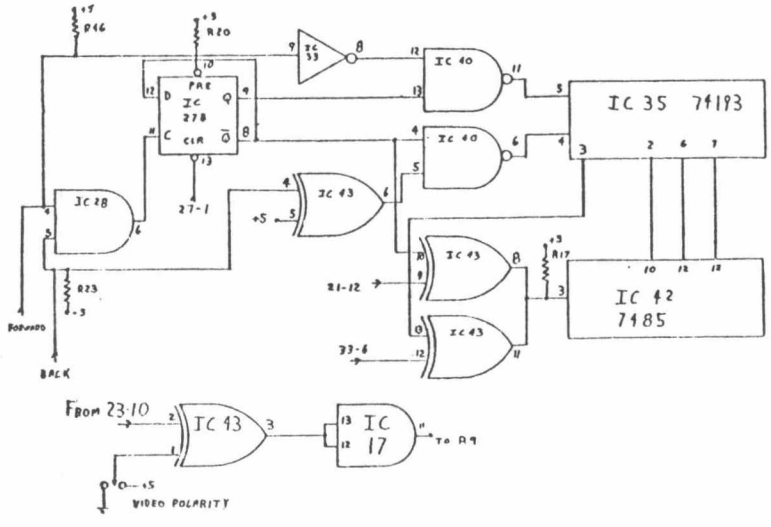
Parts Needed:

1) IC-43 (1) one 7486 quad exclusive or gate
2) IC-40 if needed (see Step 2)
3) Sharp razor to cut printed circuit foils
4) Jumper wires - insulated solid conductor smaller than 24 guage.

Steps:

You will cut PC foils (only a 1/16" gap is needed) and add jumpers. The notation 40-5 means IC-40 pin 5

1) Remove R7 (4.7K) and replace with a 1.8K resistor to increase the dot clock frequency.
2) Remove IC-40. (If you cannot remove IC's from a plated thru board without damaging the board then clip each lead of IC-40, remove its body, and desolder each pin individually. If you cut IC-40's pins then you will need a new one to complete the modification.) Cut the foil between 40-6 and 40-11. Replace IC-40.
3) On the bottom of the board cut the foil at 40-5 and 40-13. On the top cut the foil at 40-11 and 40-12. Cut the foil at 40-11 and 40-12. Cut the foil from 40-4 to 33-8 at the feed thru near 40-14, not under the IC.
4) Piggyback the 7486, now called IC-43, on top of IC-40 by bending all pins except 7 and 14 out horizontally. Then solder pins 7 and 14 of IC-43 to pins 7 and 14 of IC-40.
5) Bottom of the board cut the foil at 23-10. Connect 43-3 to 17-12. Connect 43-2 to 23-10. Connect a short wire to 43-1. If this wire is connected to +5V then video polarity is inverted; if grounded, then polarity is normal. You can add a switch here or solder permanently as desired.
6) On the bottom cut the foil at 27-13 and at 27-9. Top of the board cut the foil at 35-5 at the feed thru. Connect 40-11 to 35-5. Connect 40-12 to 33-8. Connect 40-13 to 27-9. Connect 27-1 to 27-13. Connect 27-8 to 40-4.



TVT-II 64 CHARACTER MODIFICATION

7) Bottom of the board cut the foil between 35-4 and R23. On the top cut the foils at 28-4, 28-5, 33-9. Connect 40-6 to 35-4. Connect 40-5 to 43-6. Connect 40-4 to 43-10. Connect 28-5 and 43-4 to R23. Connect 43-5 to 40-14. Connect 28-4 to R46 at 35-5. Connect 33-9 to R46.
8) On the bottom cut the foil from 12-4 to 21-1 at 21-1. Connect 12-4 to 21-14. Connect 21-1 to 21-12 to address line, A9, which is the foil that was cut from 27-9. Connect 43-9 to A9.
9) Connect 43-8 to 43-11. Connect 42-3 (the foil which was cut from 40-11) to 43-11. Connect 33-6 (the foil which was cut from 40-12) to 43-12. Connect 35-3 (which was cut from 33-9) to 43-13.
10) The modification is complete. Whew! Now double check everything....
11) Apply power. Adjust R4 and R6 to center the display. If the width is too large then decrease R7. After centering the display, if some characters lose dots you may have to change IC-22. Ugh!!! Some surplus 2513 characters are not fast enough.

Happy Computing,

Gary Alevy
1056 Fifth Avenue
New York, NY 10028

Copyright 1976

THE TRENTON COMPUTER FESTIVAL RIDES AGAIN IN NEW JERSEY!

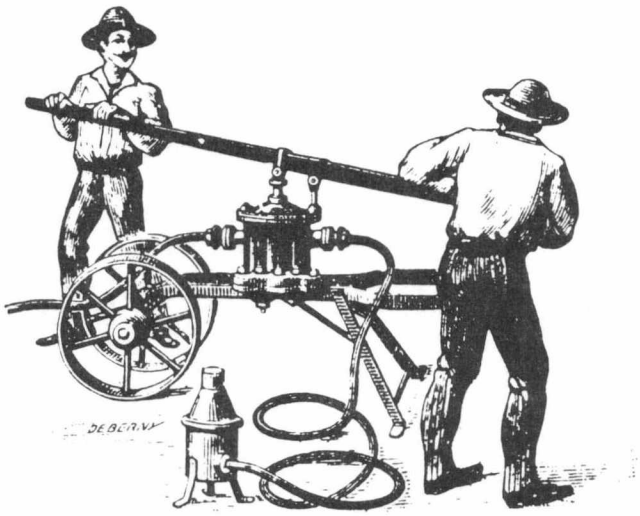
The *Trenton Computer Festival* (TCF), the originator of amateur computer fests, will happen again, on April 31 – May 1, 1977.

The second annual *TCF* will be bigger and better than the original. It is being expanded to two full days, with new and larger facilities to house up to 90 exhibitors. There will be 30 speakers and demonstrations galore. There will be hundreds of door prizes and a huge outdoor flea market.

Computer conference sections and forums are planned on the following topics: Microcomputers for home, radio amateurs, education and medicine; consumer applications of microprocessors; computer music; robots; graphics; speech synthesis; establishing amateur computer standards; computer club convention.

It is expected that attendance will exceed 3,000. There will be amateur computing contests and awards, Saturday night awards banquet, programming copying service, forum audio cassette copying service, free bus service between hotels/motels and fest, activities for wives and kids, and free parking for up to 4,000 cars.

TCF-77 is being sponsored by: Amateur Computer Group of New Jersey, Philadelphia Area Computer Society, Trenton State College Digital Computer Society, Institute of Electrical & Electronic Engineers-Princeton Section and the Department of Engineering Technology, Trenton State College.



An 8080 Disassembler
with exquisite documentation

PLUS

1. *Details about how to get it on your cassette tape for 59¢*
2. *Notes on an interesting though unstandard floppy disc with 2.5 megabit transfer rate*
3. *An example and details of a "word processing" home computer system*
4. *More good comments on Kildall's CP/M*

Ward Christensen

Editor's Note: Normally, we would typeset the text portion of an article such as this. However, we are publishing it "as received" (1) to illustrate what can reasonably be expected from a personal computer used for so-called "word processing" using a plain-jane I/O Selectric, and (2) to exhibit a beautifully-prepared manuscript, the likes of which we would be delighted to receive more often.

688 E. 154th St.
Dolton, Il. 60419
February 13, 1977

RE: My letter offering my Disassembler

DDJ:

Enclosed you will find an assembly listing of my disassembler, user documentation, and sample runs. I also have included my symbol table edit routine, which allows you to substitute meaningful labels for the simple xxxx (xxxx=hex addr) default ones. I hope you publish it, and your readers enjoy using it. I do not have facilities to distribute machine readable copies (either source or object) in any form other than Tarbell, 187 bytes/sec (Tarbell standard), in a series of source files compatible with the Processor Technology Package I assembler.

The Tarbell tape is available from the Chicago Area Computer Hobbyists Exchange (CACHE) Software Library. Although Lloyd Smith, 520 Pearce St., Dyer In. 46311 is the software librarian, I am his assistant, so deal with me directly, as I will be making the tapes. Address all correspondence to: CACHE Software Library, c/o Ward Christensen, 688 E. 154th St., Dolton, Il. 60419. Make checks payable to CACHE, NOT myself, as this is not for my profit, but rather a service from CACHE. Checks to me personally will be returned. Send $2 for a Tarbell tape of the disassembler source files, and the symbol table edit routine. Since DDJ is publishing the documentation, the only documentation shipped will be for the format and contents of the tape. Those of you on a budget can send your own cassette, a padded mailer with $.46 postage, and an extra $.13 stamp (to cover my copying costs), and I will write your tape.

To answer the questions you asked when you published my first letter:

1) What kind of disk am I running? My disk is a 'Dynastor' disk. I knew for years that when I would own my own computer, I would want a dual floppy disk for it. Therefore, when the ALTAIR came out, I ordered one, then started looking for a floppy. MITS advertised one, but the price seemed high, and I was doubtful of delivery schedules.

I looked thru Computerworld, and came across an ad for the Dynastor disks, at a reasonable price. I wrote for information (Dynastor Inc, 5867 N. Broadway, Denver Colo. 80216) and received a detailed interface manual. It appeared to need only a 4 port parallel interface, so I ordered a dual disk (gotta' have dual if you are really interested in backing up a disk).

The final price I paid, including power supply, head cleaning disk, etc. came out just about what MITS wanted for a dual, so I didn't really save there. Where I did save, was in getting my disk running nearly a year before most hobbyists who had ordered the MITS disk got theirs running.

The Dynastor technology is quite interesting - they do not use an 'industry standard' floppy, but instead their own, which is spun at 3600 RPM (most floppies are spun at 360 RPM). It is then read with a FLYING head, for a data transfer rate of 2.5 million bits/sec. The seeking is completely controlled on the disk controller itself - the disk is preformatted with track and sector info so all you do is say 'seek to track x sector y' then wait for a busy signal to end. I think it was this 'state of the art' technology that made me go with them.

I did have a few headaches at first, primarily because the disk is VERY humidity sensitive - and in the home, with cooking, etc. humidity is a problem. They solved the problem with a simple heater on the metal plate which the disk flies a fraction of an inch from.

The disk controller (F.A.D. for Formatting, Addressing, and Decoding) has an on-board shift register one sector long, so the computer can read and write data at any rate from DC to 2.5MHz, out of, or into, the shift register. I originally programmed this 1 bit at a time, with a transfer rate of about 2K bytes/sec. I recently added a hardward 8 bit shift register to the read circuit, upping the data

transfer considerably. I wrote a program to compare 2 disks after copying, and it takes about 1 minute to read both disks and compare. (Each disk is 270K bytes.) Disk copy is even faster, because the data doesn't have to be read into the computer, but just into the controller shift register, issue a new seek, then a write back to the new sector right from the shift register. Very nice.

My software consists of Processor Technology Package I assembler development system, with disk source file LOAD and SAVE added. I have kludged (sp?) up a system for saving assembled programs, and executing them by name, with simple parameter passing, but am now working on my own DOS. (DYNADOS?) If I had it to do over again, I would probably go with Digital System's excellent (Shugart) disk and Digital Researches super operating system, CP/M. (see the P.S. for the latest on this)

2) What sort of printer and printing software am I using? I wanted an IBM selectric typewriter, for its print quality, and variety of type balls, particularly APL, as I am truly an APL bigot. ..Hope to have my own on the ALTAIR some day.

My printer software consists of a software driver to convert from ASCII to BCD, taking care of upshifts, downshifts, etc, and waiting for a microswitch closure (my hardware addition) to detect end of carriage return. I sometimes drive it directly, but do have a program which allows me to buffer up to 8K characters in memory, so I can print one file, letter, listing, BASIC program, etc. while going on with other programming or editing. I have a variety of programs, such as one to format (into columns) an assembler program - that way I can save it on disk or Tarbell with no unnecessary blanks, but can still get a very readable source or assembly listing. A software translate table is used to allow using different type balls.

In your question, you were wondering about my formatted letter - you were probably interested in my 'script' processor, which has commands such as:

```
.B     Break (new paragraph, no blank line)
.Sn    Space n lines
.U     Unformatted mode
.Fnn   Format (fill but don't justify)
       nn columns wide
.Jnn   Justify (right and left columns)
       nn columns wide
.E     Page eject
.Mnn   Set margin nn col's in
.Pnn   Paragraph indent nn col's
.Lnn   Set page length
```

I call it a SCRIPT processor, after IBM's product which it is an improper subset of. I originally wrote it in MITS 8K BASIC, using PEEKS into a resident source file, printing the output. After debugging (interpretive languages are SUPER debugging tools) in BASIC, I hand-converted it to assembler, using such conversions as 'CALL S2000' where BASIC had 'GOSUB 2000'. I had to hand code some 'MID$' functions as subroutines, but otherwise it is pretty much one-for-one from BASIC. I did add disk input, such that I can type the command 'SCRIPT nn' where nn is a sector address of the file to be 'scripted'.

I did add some sophistication, in that when justifying, I alternate lines padding blanks from the left, then the right, then left, etc. to prevent 'rivers of white' which appear with a simple justification program which always adds blanks starting with the leftmost word space first. I did not go all out, i.e. trying to prevent 'widows' which is a single word left on a line by itself at the end of a paragraph.

My SCRIPT program is not for distribution because it is totally undocumented, integrated into my unique disk, and would possibly represent a conflict of interest with my regular job.

The printer is an IBM selectric, '2741 compatible' which means it has an EIA interface. I am not very hardware oriented, so I was not willing to attack direct interfacing with solenoids, etc. I bought it used directly from the local Trendata office for about 40% off of list. Trendata buys I/O selectrics from IBM, then wraps their own electronics around it, much like Anderson Jacobson does. Selectrics come in 2 types, BCD (binary coded decimal) and Selectric (also called correspondence). They differ in what position on the typeball is typed based on what character is received. Office selectrics use the latter, while only a few '2741' type terminals, and IBM '1050' terminals use the BCD. If you have a 'correspondence' Selectric, then the wealth of office typewriter balls is available for your use. If you have a BCD terminal, there are only 2 balls available, a regular one and an APL one. That is why you noticed the different (=plain?) type style of my first letter. This letter is being typed with an office selectric ball, because I use a software translate table to allow my ASCII to be translated to whatever ball I want.

Keep up the good work; Best to all of you in the (relatively) new year...

P.S. I'd like to add my name to the growing list of satisfied CP/M users. I purchased it from Digital Research, and installed it in about 5 days. The majority of the time was spent making a disk I/O package for it. It was relatively easy to install because the I/O part of CP/M is split out as a series of JMPs to the proper routines.

I heard an interesting comment on CP/M, when I told someone how good a floppy disk operating system it was, they said "Oh really? I wasn't interested, because I figured, what good could it be for only $70". The truth is it is very good. Assembler, editor, dynamic debugger, automatic disk allocation, etc.

yours,

Ward Christensen
688 E. 154th St.
Dolton, Ill. 60419

DISASSEMBLER OPERATING ENVIRONMENT:

This disassembler is for 8080 object programs. It is intended to run in the 8080, along with the machine (object) code of the program to be disassembled.

It is designed to operate on an 8080 having 8 sense switches accessible via input 255. These sense switches control the following functions:

```
 BIT
VALUE ----------ACTION TAKEN IF ON----------


80H Write listing into memory
20H Generate DB's, not instructions
10H  Generate ASCII characters in quotes based on
       sense switch 20H.

       20H on:  Generate DB 'char' if possible;
                otherwise, generate hex;
       20H off: If instruction is an accumulator
                immediate, (MVI, CPI, etc), then
                generate hex, then ASCII. Example:
                MVI A,30H '0'

08H Generate labels Lxxxx instead of generating
      xxxxH, for operands of 3 byte instructions
      (SHLD, LXI, JMP..)
04H  Build symbol table - place every 3 byte
       instruction operand in the symbol table,
       with a default value of Lxxxx where xxxx is
       the operand's hex value
```

01H Use symbol table - Before a line of disassembly is generated, look up the address of the instruction in the symbol table. If it is found, use the corresponding symbol name as the label of the instruction. Also, when generating 3 byte instruction operands, look up the hex value, and if found in the symbol table, print the corresponding label as the operand.

It would require a major rewrite of the program to make it able to run without sense switches. This is the operator is required to intervene to differentiate between assembler instructions and data, by changing the sense switch settings at the proper time.

GENERAL DISASSEMBLY PROCEDURE:

The easiest way is to do a 'quick and dirty' 1 pass disassembly. The instructions will show up correctly, with either hex operands (switch 08H off) or Lxxxx type operands (switch 08H on). When data is encountered, it will be interpreted as instructions, unless it is an invalid instruction, in which case a DB xxH will be generated, with xx the data byte value. If reassembled at it's original location, the program should assemble correctly. The instructions such as 'MOV B,A' which were erroniously generated from an ASCII data field, will correctly assemble as a 'G' just as it originally was.

A better way to disassemble is to look at the output of the first pass of the disassembler, and determine where data areas are - they can be recognized various ways: 1) ASCII data disassembles as a lot of MOV instructions in a row with quite a few being meaningless (MOV L,L) or repetitive (MOV D,E followed by MOV D,E). Also, DB 20H appears for every space in the ASCII data. 2) An area which is cleared to 0 (such as an input buffer or work area might be) will show up as a string of NOP instructions. 3) Other data will show up as unusual instruction sequences such as loading a register or register pair without being used between loads; also DB's of 10H, 20H, 30H, 38H, etc. will usually indicate data areas.

When you have determined where the data areas are in the program being disassembled, write them down, and use them as you disassemble the program again. This time, turn the DB sense switch up for data, and down for instructions, based on your notes.

If you have a slow output device for the disassembly listing, you will have time to change the sense switches on an instruction by instruction basis. However, if you have a fast device, such as a CRT, or as I do, the Processor Technology VDM, may I make the following suggestion: Modify the output routine ('CALL TYPE' in the assembly) such that each time a C/R is printed, you have the option of waiting. I use the ESC key to stop - when I press any key (including ESC again) the output routine returns to the caller. Thus, I can use ESC to 'single step' my listing, allowing precise sense switch changing. If I want the disassembly to proceed, I just press some key other than ESC.

For an even better disassembly, combine the above procedure of finding data areas, with use of symbolic labels. To do this, just turn on the sense switch (04H) to build the symbol table, and execute the disassembler. This will cause the operands of every LDA, STA, LHLD, SHLD, JMP, CALL, and conditional JMP or CALL to be placed in the symbol table. NOTE THAT the symbol table must be cleared to binary zeros before being used. This is so that you can re-execute the disassembler without automatically having the symbol table cleared.

Then, re-do the disassembly, with sense switch 01H on, to use the labels in the table. Also, use your notes on where the data areas are to set switch 20H appropriately.

The last and most complete technique of disassembly requires that you have some knowledge of the program being disassembled, specifically knowledge of the labels. This may be obtained either by having a listing of the original assembly (but only the object code in machine readable form), or by studying the disassembled code, and determining the function of some routines or data areas. For example, if you discover an ASCII to binary routie at a particular address, you could call it ASCBI (labels in the disassembler have max. 5 byte labels).

Then using this knowledge, edit the symbol table, substituting meaningful labels for the Lxxxx default labels. The final disassembly, using the more meaningful labels, and with data generated in the proper location, will generally be quite usable as a base for modification and reassembly.

The symbol table is edited using the program shown following the disassmebler itself. The program uses 1 parameter passed to it, which is the relative number of the first symbol to be edited (0=first). This allows you to edit part of the table, then go back and do a test disassembly, then continue editing the symbol table.

Here is how the symbol table editor works: When you execute the editor, it prints a label, it's hex value, then stops. You then key one of the following:

1. A new 1 to 5 character label value
   (use backspace (control-H) for corrections)
2. A carriage return, in which case the label
   is not changed
3. A control-X, in which case symbol table
   editing is terminated.

DISASSEMBLER EXTERNAL DATA REFERENCES

BBUF -

This area contains the address of the program to be disassembled. For example, if the program is at 4000 hex, then BBUF would contain 00 hex, and BBUF+1 would contain 40 hex.

FILE -

FILE is an 8 byte area. The first 2 bytes contain the address (in 8080 'low order byte first' format) of the area of memory into which the disassembled program listing will be stored, if sense switch A15 is on. If this sense switch is not on, then none of this area will never be looked at. FILE is an 8 byte area - you initialize the first 2 bytes to the starting address (in 8080 'low order byte first' format). The disassembler uses the next 2 bytes as a running end of file pointer. The last 4 bytes have the highest line number stored in ASCII when disassembly is shut down due to hitting the HIFIL address (see below), or due to the operator entering control-x.

When the disassembler receives control, the stack must be initialized. The start of the program to be disassembled must be at BBUF, and if used, the output file pointer must be at FILE.

Execute the disassembler. Change the sense switch settings as outlined in the 'DISASSEMBLER OPERATING ENVIRONMENT' documentation.

GENERAL NOTES:

If sense switch 80H is on, the file written into memory has line numbers which are the address that the instruction came from. I modified my Processor Tech. Package 1 routine to allow hex digits (A-F) in the last 3 digits of the line number. It won't accept a leading alpha char, because when keying such a line in the assembler thinks the data is a command, not a line number.

I usually use an address of 40H as the location to write the file into memory. However, if the program I am disassembling occupies that space, I use a higher address, up to 2800H. Remember that the HIFIL equate in the assembly defines1 where writing the file into memory is to shut down. Therefore, change it, if writing the file into other memory. For example, if you have memory at F000-FFFF, and are writing the file at F000, change HIFIL to 0FFH. NOTE that HIFIL is not the point BELOW which disassembly stop. Rather, it is the page address at which the disassembly stops when a line written into memory ends IN that page.

Processor Technology file format is as follows:

1 byte binary line length - this length includes the length byte, the carriage return at the end of the line, and all data bytes inbetween. A line length of 1 signifies end of file.

4 byte ASCII line number.

1 byte ASCII space.

n byte data line - if there is a label on the line, it starts in this location. If no label is present, an additional ASCII space is included, then the operation code.

1 byte ASCII carriage return (0DH, 13 decimal).

DISASSEMBLER PROGRAM LOGIC

Most of the program logic is gotten directly by reading the commented listing. However, the use of the op code table will be explained here, in depth, because the op code table is the heart of the disassembler. Each entry in the op code table is 7 bytes long. The bytes are as follows:

BYTE	------CONTENTS------
0	Mask to 'AND' the instruction with to delete register dependent bits.
1	Value to compare the 'registerless' instruction against for a match.
2	The instruction type (0-B hex).
3-6	The printable (ASCII) instruction value.

The table is used as follows: An instruction byte is fetched from memory, at address 3148, and is then processed in the 'ULP' loop. The first byte of an op code table entry is ANDed with the instruction to delete the register dependent bits.

For example, an op code in hex of 14, which is INR D, would, when the correct table entry is 'about to be found', be ANDed with a C7 (at address 3587). This would make the instruction an 04. The 04 would match the 04 as the registerless 'INR' instruction, at address 3588.

The table lookup routine having got a match, jumps to label FOUND (3164 hex) which then picks up the instruction type from the op code table (address 3589) and, using the 02 (instruction type 02) found there causes a JMP to instruction type 2, via the jump table, JMTBL. (The instruction types are shown as comments at address 3176 in the listing, the address of JMTBL.)

POSSIBLE MODIFICATIONS TO THE DISASSEMBLER

In using the disassembler, I have found a few things which I might like to change. First, the LXI instruction always generates a hex value for it's operand. This is the result of an evolutionary process: my first version of this disassembler always caused label operands to be generated for LXI instructions. I found the programs I disassembled then had too many 'LXI H,L0000' type instruction sequences, i.e. where data, not a label was being LXI'd. Therefore, for subsequent versions of the disassembler, I changed the LXI logic to generate hex data. Now, I have less changes to do to programs, but there are still some times in which the LXI is to a data area or instruction which has been disassembled, and therefore is in the symbol table. Perhaps making this another sense switch option would be the best way.

Anyone interested in increasing the speed of execution of the disassembler might persue ordering the lookup table into 'most frequently used' sequence. At least, MOV should be made second in the table, with HLT first. HLT is not used more, but MUST preceed the MOV instruction, otherwise HLT instructions will be decoded as MOV M,M.

```
3100          0010 *@200-10
3100          0020 *
3100          0030 * * * * * * * * * * * * * * *
3100          0040 *           8 0 8 0           *
3100          0050 * D I S A S S E M B L E R *
3100          0060 * * * * * * * * * * * * * * *
3100          0070 *
3100          0080 * PRODUCES PRINTED LISTING
3100          0090 * OF AN OBJECT PROGRAM IN
3100          0100 * STORAGE.
3100          0110 *
3100          0120 * OPTIONALLY WRITES A FILE
3100          0130 * INTO MEMORY, COMPATIBLE WITH
3100          0140 * THE PROCESSOR TECHNOLOGY PACKAGE I
3100          0150 * AND ALS-8 ASSEMBLER FILE FORMAT.
3100          0160 *
3100          0170 * GENERATING INSTRUCTIONS VERSUS DB'S
3100          0180 * IS SELECTED BY SENSE SWITCHES DURING
3100          0190 * THE EXECUTION OF THE DISASSEMBLER.
3100          0200 *
3100          0210 PSW    EQU  6
3100          0220 SP     EQU  6
3100          0230 *
3100          0240 * DEFINE KEYBOARD INPUT PORT ADDR
3100          0250 * (USED ONLY FOR CTL-X CHECK TO STOP
3100          0260 * WRITING FILE INTO MEMORY)
3100          0270 KEYB   EQU  1        KEYBOARD DATA INPUT PORT
3100          0280 *
3100          0290 * DEFINE ROUTINE TO TYPE ASCII CHAR IN A.
3100          0300 TYPE   EQU  409CH    ASCII OUTPUT ROUTINE
3100          0310 *
3100          0320 * DEFINE ADDRESS TO EXIT FROM DISASSEMBLER
```

```
3100                0330 SPVR  EQU  401FH   RETURN ADDR TO SUPERVISOR
3100                0340 *
3100                0350 * DEFINE A LOCATION WHICH CONTAINS THE ADDRESS
3100                0360 * OF WHERE THE FILE CAN BE BUILT IN MEMORY.
3100                0370 * (NOTE USERS OF PROC. TECH. PACKAGE 1 -
3100                0380 * THIS AREA IS ACTUALLY FILE+5, THE LOCATION
3100                0390 * AT WHICH THE START OF CURRENT FILE IS STORED)
3100                0400 FILE  EQU  3805H   ADDRESS OF ADDRESS OF FILE
3100                0410 *
3100                0420 * DEFINE A LOCATION WHICH CONTAINS THE ADDRESS
3100                0430 * OF WHERE THE DISASSEMBLY IS TO START
3100                0440 * (NOTE USERS OF PROC. TECH. PACKAGE 1 -
3100                0450 * THIS AREA IS ACTUALLY BBUF+2, THE LOCATION
3100                0460 * AT WHICH THE PARAMETER xxxx IS STORED WHEN
3100                0470 * YOU USE THE COMMAND 'EXEC wwww xxxx')
3100                0480 BBUF  EQU  3868H   ADDR OF BINARY PARM BUFFER
3100                0490 *
3100                0500 * DEFINE LOCATION FOR SYMBOL TABLE (IF USED)
3100                0510 SYMTB EQU  0C002H  SYMBOL TABLE
3100                0520 *
3100                0530 * DEFINE THE PAGE ADDRESS (I.E. HIGH 8 BITS OF
3100                0540 * THE ADDRESS) AT WHICH WRITING A FILE INTO
3100                0550 * MEMORY IS TO SHUT DOWN, TO PREVENT OVERLAYING
3100                0560 * OTHER PROGRAMS OR DATA.
3100                0570 HIFIL EQU  30H
3100                0580 *
3100                0590 * SEE ACCOMPANYING DOCUMENTATION FOR
3100                0600 * OPERATIONS AND PROGRAM LOGIC
3100                0610 *
3100                0620 *================================*
3100                0630 *===START OF PROGRAM EXECUTION===*
3100                0640 *================================*
3100 DB FF          0650 START IN   -1      READ SSW
3102 E6 04          0660       ANI  4       BUILD SYM TABLE?
3104 CA 0B 31       0670       JZ   NOSYM
3107 AF             0680       XRA  A       GET 0
3108 32 02 C0       0690       STA  SYMTB   INIT SYM TABLE
310B                0700 *---> START EXECUTION HERE IF YOU WANT
310B                0710 * TO KEEP PREV SYM TBL, SUCH AS TO ADD TO IT
310B 2A 05 38       0720 NOSYM LHLD FILE    LOAD FILE START ADDR
310E 23             0730       INX  H
310F DB FF          0740       IN   -1      READ SSW
3111 E6 80          0750       ANI  80H     FILE BEING BUILT?
3113 CA 19 31       0760       JZ   NOSTR   NO, SO DON'T SET END ADDR
3116 22 07 38       0770       SHLD FILE+2  SET END OF FILE ADDR
3119 2A 68 38       0780 NOSTR LHLD BBUF    HL=START ADDR TO DISASM
311C CD 22 31       0790 DISAM CALL LINE    DISASSEMBLE 1 LINE
311F C3 1C 31       0800       JMP  DISAM   LOOP UNTIL STOPPED
3122                0810 *
3122                0820 * ROUTINE TO PRINT A LINE OF DISASSEMBLY
3122                0830 * N-O-T-E THAT THIS ROUTINE IS A
3122                0840 * SUBROUTINE, I.E. MAY BE CALLED
3122                0850 * FROM ANY PROGRAM, WITH HL POINTING
3122                0860 * TO AN INSTRUCTION TO BE DISASSEMBLED.
3122                0870 * THIS SUBROUTINE PRINTS THE LINE, C/R,
3122                0880 * THEN RETURNS, W/HL AT NEXT INSTR.
3122                0890 *
3122 E5             0900 LINE  PUSH H       SAVE INPUT POINTER
3123 2A 07 38       0910       LHLD FILE+2  LOAD START OF LINE POINTER
3126 22 F1 35       0920       SHLD LINAD   SAVE
3129 E1             0930       POP  H
312A 7C             0940       MOV  A,H     PRINT
312B CD F1 32       0950       CALL XO      ..THE
312E 7D             0960       MOV  A,L     ..ADDR
312F CD FF 32       0970       CALL XOB     ..IN HEX
3132                0980 *
3132                0990 * PRINT LABEL FROM TABLE IF SS08 ON
3132                1000 *
3132 44             1010       MOV  B,H     SET UP
3133 4D             1020       MOV  C,L     ..FOR 'PRTLB' ROUTINE
3134 DB FF          1030       IN   -1
3136 E6 01          1040       ANI  1
3138 C4 FC 33       1050       CNZ  PRTLB   PRINT LABEL
313B 3E 20          1060       MVI  A,' '   GET A SPACE
313D CD 73 33       1070       CALL ZTYPE   PRINT AFTER LABEL
3140 DB FF          1080       IN   -1      READ SSW
3142 E6 20          1090       ANI  20H     SS A13 ON?
3144 C2 87 32       1100       JNZ  PRDB    ..SAYS PRT DB'S
3147 E5             1110       PUSH H
3148 66             1120       MOV  H,M     OP CODE TO H
3149                1130 *
3149                1140 * DO TABLE LOOKUP
3149                1150 *
3149 01 61 34       1160       LXI  B,OPTAB
314C 0A             1170 ULP   LDAX B
314D B7             1180       ORA  A       END OF TABLE?
314E CA 86 32       1190       JZ   PRDBP   YES, POP H, PRINT DB
3151 A4             1200       ANA  H       AND OFF REGISTER BITS
3152 6F             1210       MOV  L,A     SAVE REGISTERLESS INSTR.
3153 03             1220       INX  B
3154 0A             1230       LDAX B       GET REGISTERLESS MATCH
3155 BD             1240       CMP  L       SAME?
3156 CA 64 31       1250       JZ   FOUND   YEA!
3159 79             1260       MOV  A,C     ..BOO! NO MATCH
315A C6 06          1270       ADI  6       POINT TO NEXT OP ENTRY
315C 4F             1280       MOV  C,A
315D D2 4C 31       1290       JNC  ULP
3160 04             1300       INR  B       ..ADD CARRY O'FLO C TO B
3161 C3 4C 31       1310       JMP  ULP
3164                0010 *@220-5
3164                0020 *
3164                0030 * OP CODE FOUND IN TABLE
3164                0040 *
3164 03             0050 FOUND INX  B       POINT TO TYPE
3165 0A             0060       LDAX B       GET OP CODE TYPE (0-A)
3166 32 F5 35       0070       STA  TYFLG   SAVE INSTR TYPE
3169 D5             0080       PUSH D
316A                0090 *
316A                0100 * COMPUTE JMP TBL ADDR
316A                0110 *
316A 5F             0120       MOV  E,A     MOVE TYPE TO E,
316B 16 00          0130       MVI  D,0     0 TO E
316D 21 76 31       0140       LXI  H,JMTBL
3170 19             0150       DAD  D       ADD IN
3171 19             0160       DAD  D       ..3 * INSTR
3172 19             0170       DAD  D       ..TYPE
3173 D1             0180       POP  D
3174                0190 * PUT JMP TABLE ENTRY AT TOP OF
3174                0200 * STACK, RESTORE HL (PUSHED AT 'NOMOV')
3174 E3             0210       XTHL
3175 C9             0220       RET  ,       ..TO JMP TABLE ENTRY
3176                0230 JMTBL EQU  $
3176                0240 *                   LEN OPERANDS EXAMPLE
3176 C3 9A 31       0250       JMP  TYPE0    1  NONE     DAA, EI
3179 C3 A1 31       0260       JMP  TYPE1    1  REG      ADD, ANA
317C C3 AA 31       0270       JMP  TYPE2    1  REG      INR, DCR
317F C3 B3 31       0280       JMP  TYPE3    1  XR       STAX, LDAX
3182 C3 BC 31       0290       JMP  TYPE4    2  REG,DATA MVI (ONLY)
3185 C3 C8 31       0300       JMP  TYPE5    2  DATA     OUT, CPI
3188 C3 09 32       0310       JMP  TYPE6    3  XR,DATA  LXI (ONLY)
318B C3 2A 32       0320       JMP  TYPE7    3  ADDR     LHLD, JMP
318E C3 4E 32       0330       JMP  TYPE8    3  ADDR     JZ, CNC
3191 C3 59 32       0340       JMP  TYPE9    1  NONE     RZ, RC
3194 C3 64 32       0350       JMP  TYPEA    1  RST #    RST (ONLY)
3197 C3 74 32       0360       JMP  TYPEB    1  REG,REG  MOV (ONLY)
```

```
319A                0370 *----------------------------*
319A                0380 *
319A                0390 * TYPE 0 - SINGLE BYTE, NO OPERANDS (DAA, RET)
319A                0400 *
319A CD 1E 33       0410 TYPE0 CALL PROP     PRINT OP, RELOAD OP CODE
319D 23             0420 IXHRT INX  H        POINT TO NEXT CHAR
319E C3 71 33       0430       JMP  RET      END OF LINE
31A1                0440 *
31A1                0450 * TYPE 1 - 1 BYTE, REG IN .....XXX (ADD, ANA)
31A1                0460 *
31A1 CD 1E 33       0470 TYPE1 CALL PROP     PRINT OP, RELOAD OP CODE
31A4 CD 47 33       0480 T1B   CALL REG      PRINT REG
31A7 C3 9D 31       0490       JMP  IXHRT    INX H, PRINT C/R
31AA                0500 *
31AA                0510 * TYPE 2 - 1 BYTE, 1 REG IS ..XXX... (INR)
31AA                0520 *
31AA CD 1E 33       0530 TYPE2 CALL PROP     PRINT OP, RELOAD OP CODE
31AD CD 44 33       0540       CALL REGM     PRINT REG IN ..XXX...
31B0 C3 9D 31       0550       JMP  IXHRT    INX H, C/R, RET
31B3                0560 *
31B3                0570 * TYPE 3 - 1 BYTE, XR IN ..XX.... (STAX)
31B3                0580 *
31B3 CD 1E 33       0590 TYPE3 CALL PROP     PRINT OP, RELOAD OP CODE
31B6 CD 35 33       0600       CALL XR       ..THEN XR
31B9 C3 9D 31       0610       JMP  IXHRT    INX H, C/R
31BC                0620 *
31BC                0630 * TYPE 4 - MVI (ONLY)
31BC                0640 *
31BC CD 1E 33       0650 TYPE4 CALL PROP
31BF CD 44 33       0660       CALL REGM     PRINT REG IN ..XXX...
31C2 CD 81 32       0670       CALL COMMA    ..A COMMA,
31C5 C3 CB 31       0680       JMP  T52      ..THEN HEX BYTE
31C8                0690 *
31C8                0700 * TYPE 5 - 1 BYTE IMMEDIATE, (OUT, CPI)
31C8                0710 *
31C8 CD 1E 33       0720 TYPE5 CALL PROP
31CB 23             0730 T52   INX  H        POINT TO OPERAND
31CC 7E             0740       MOV  A,M      GET IMMEDIATE OPERAND
31CD CD 62 33       0750       CALL XOO      ..OUTPUT
31D0 3E 48          0760 T54   MVI  A,'H'    H FOR HEX
31D2 CD 73 33       0770       CALL ZTYPE
31D5 7E             0780       MOV  A,M      RELOAD CHAR
31D6 23             0790       INX  H
31D7 C5             0800       PUSH B
31D8 47             0810       MOV  B,A
31D9 DB FF          0820       IN   -1
31DB E6 10          0830       ANI  10H      SS A12 ON SAYS
31DD 78             0840       MOV  A,B
31DE C1             0850       POP  B
31DF CA 71 33       0860       JZ   RET      ..TO PRINT ASCII
31E2 FE 20          0870       CPI  ' '
31E4 DA 71 33       0880       JC   RET      UNPRINTABLE
31E7 FE 5B          0890       CPI  'Z'+1
31E9 D2 71 33       0900       JNC  RET      UNPRINTABLE
31EC                0910 *
31EC                0920 * OUTPUT CHAR IN A, IN QUOTES
31EC                0930 * IF INSTR IS AN IMMEDIATE INSTR
31EC                0940 *
31EC 3A F5 35       0950       LDA  TYFLG    GET TYPE FLAG
31EF FE 04          0960       CPI  4
31F1 DA 71 33       0970       JC   RET      IF TYPE 0-3
31F4 FE 06          0980       CPI  6
31F6 D2 71 33       0990       JNC  RET      IF TYPE 6-A
31F9 2B             1000       DCX  H        BACK UP
31FA 7E             1010       MOV  A,M      GET CHAR
31FB 23             1020       INX  H        RESTORE H
31FC F5             1030 QCHAR PUSH PSW
31FD CD 07 33       1040       CALL OQ       OUTPUT QUOTE, CHAR
3200 F1             1050       POP  PSW
3201 3E 27          1060       MVI  A,''''
3203 CD 73 33       1070       CALL ZTYPE    TYPE CLOSING QUOTE
3206 C3 71 33       1080       JMP  RET      END OF LINE
3209                1090 *
3209                1100 * TYPE 6 - LXI (ONLY)
3209                1110 *
3209 CD 1E 33       1120 TYPE6 CALL PROP
320C 7E             1130       MOV  A,M
320D CD 35 33       1140       CALL XR
3210 CD 81 32       1150       CALL COMMA
3213 C3 1C 32       1160       JMP  TYP6H    ALWAYS PRINT HEX FOR LXI
3216                1170 *
3216                1180 * PRINT 2 BYTE HEX OPERAND
3216                1190 *
3216 CD DF 33       1200 PRT2X CALL SYMBO    PRINT FR TBL IF SS10 ON
3219 DA 71 33       1210       JC   RET
321C 23             1220 TYP6H INX  H
321D 46             1230       MOV  B,M      SAVE FIRST CHAR
321E 23             1240       INX  H
321F 7E             1250       MOV  A,M      GET 2ND CHAR
3220 CD 62 33       1260       CALL XOO      OUTPUT IT FIRST
3223 78             1270       MOV  A,B      GET FIRST CHAR
3224 CD F1 32       1280       CALL XO       THEN OUTPUT IT
3227 C3 D0 31       1290       JMP  T54
322A                0010 *@240-4
322A                0020 *
322A                0030 * TYPE 7 - 2 BYTE ADDR OPERAND (LHLD, JMP)
322A                0040 *
322A CD 1E 33       0050 TYPE7 CALL PROP     PRT OP CODE, RELOAD OP
322D                0060 *** GO DO TABLE LOOKUP OF OPERAND
322D                0070 *** (RET'S CARRY OFF IF NOT USING TABLE)
322D CD DF 33       0080       CALL SYMBO    PRINT IF IN TABLE
3230 DA 71 33       0090       JC   RET      LABEL CAME FROM TABLE
3233                0100 *** IF SS11 ON, PRINT Lxxxx IF OFF, 0xxxxH
3233 DB FF          0110       IN   -1       READ SSW
3235 E6 08          0120       ANI  8
3237 CA 16 32       0130       JZ   PRT2X    PRINT AS HEX
323A 3E 4C          0140       MVI  A,'L'    LABEL PREFIX
323C CD 73 33       0150       CALL ZTYPE
323F 23             0160       INX  H
3240 46             0170       MOV  B,M      FIRST BYTE
3241 23             0180       INX  H
3242 7E             0190       MOV  A,M      SECOND BYTE
3243 CD F1 32       0200       CALL XO       OUTPUT 2ND FIRST
3246 78             0210       MOV  A,B      GET 1ST
3247 CD F1 32       0220       CALL XO       ..THEN OUTPUT IT
324A 23             0230       INX  H
324B C3 71 33       0240       JMP  RET      C/R
324E                0250 *
324E                0260 * TYPE 8 - CONDITIONAL JUMP OR CALL
324E                0270 *
324E 03             0280 TYPE8 INX  B        POINT TO OP CODE
324F 0A             0290       LDAX B        GET 'J' OR 'C'
3250 32 F6 35       0300       STA  CC1      STORE IT
3253 CD B2 32       0310       CALL GETCC    GET COND CODE
3256 C3 2A 32       0320       JMP  TYPE7    THEN PRINT OPERAND
3259                0330 *
3259                0340 * TYPE 9 - CONDITIONAL RETURN
3259                0350 *
3259 3E 52          0360 TYPE9 MVI  A,'R'
325B 32 F6 35       0370       STA  CC1      INIT OP CODE
325E CD B2 32       0380       CALL GETCC    GET CONDITION
3261 C3 9A 31       0390       JMP  TYPE0    PRINT OP CODE, C/R
3264                0400 *
3264                0410 * TYPE A - RST (ONLY)
3264                0420 *
3264 CD 1E 33       0430 TYPEA CALL PROP     PRT OP CODE, RELOAD OP
```

```
3267 1F        0440        RAR
3268 1F        0450        RAR
3269 1F        0460        RAR
326A E6 07     0470        ANI  7        ISOLATE RST NUMBER
326C F6 30     0480        ORI  '0'      MAKE PRINTABLE
326E CD 73 33  0490        CALL ZTYPE    PRINT IT
3271 C3 9D 31  0500        JMP  IXHRT    INX H, PRINT C/R
3274           0510 *
3274           0520 * TYPE B - MOV (ONLY)
3274           0530 *
3274 CD 1E 33  0550 TYPEB  CALL PROP     PRT OP CODE, RELOAD OP
3277 CD 44 33  0560        CALL REGM     PRINT ..XXX... REG
327A CD 81 32  0570        CALL COMMA    PRINT COMMA
327D 7E        0580        MOV  A,M      RELOAD OP
327E C3 A4 31  0590        JMP  T1B      PRINT REG, C/R
3281           0620 *
3281           0630 *    -------------------------
3281           0640 *   | DISASSEMBLER SUBROUTINES |
3281           0650 *    -------------------------
3281           0660 *
3281           0670 * PRINT COMMA
3281           0680 *
3281 3E 2C     0690 COMMA MVI  A,','
3283 C3 73 33  0700        JMP  ZTYPE
3286           0710 *
3286           0720 * PRINT DB
3286           0730 *
3286 E1        0740 PRDBP POP  H
3287 3E 44     0750 PRDB  MVI  A,'D'
3289 CD 73 33  0760        CALL ZTYPE
328C 3E 42     0770        MVI  A,'B'
328E CD 73 33  0780        CALL ZTYPE
3291 3E 20     0790        MVI  A,' '
3293 CD 73 33  0800        CALL ZTYPE
3296 2B        0810        DCX  H
3297 DB FF     0820        IN   -1
3299 E6 30     0830        ANI  30H
329B FE 30     0840        CPI  30H      SSW A13, A12 ON?
329D C2 CB 31  0850        JNZ  T52
32A0 23        0860        INX  H        YES - PRINT ASCII
32A1 7E        0870        MOV  A,M
32A2 2B        0880        DCX  H
32A3 FE 20     0890        CPI  ' '
32A5 DA CB 31  0900        JC   T52      UNPRINTABLE
32A8 FE 5B     0910        CPI  'Z'+1
32AA D2 CB 31  0920        JNC  T52      UNPRINTABLE
32AD 23        0930        INX  H
32AE 23        0940        INX  H
32AF C3 FC 31  0950        JMP  QCHAR    PRINT QUOTED CHAR
32B2           0960 *
32B2           0970 * PRINT CONDITION CODE
32B2           0980 *
32B2 7E        0990 GETCC MOV  A,M      GET CONDITION CODE
32B3 1F        1000        RAR
32B4 1F        1010        RAR
32B5 E6 0E     1020        ANI  0EH      ISOLATE COND. CODE
32B7 E5        1030        PUSH H
32B8 21 D1 32  1040        LXI  H,CCTAB  POINT TO CC TABLE
32BB CD 55 33  1050        CALL ADD      ADD A TO HL
32BE 7E        1060        MOV  A,M
32BF 32 F7 35  1070        STA  CC1+1
32C2 23        1080        INX  H
32C3 7E        1090        MOV  A,M
32C4 32 F8 35  1100        STA  CC1+2
32C7 E1        1110        POP  H
32C8 01 F5 35  1120        LXI  B,CC1-1
32CB 3E 20     1130        MVI  A,' '    GET SPACE
32CD 32 F9 35  1140        STA  CCB      SAVE AS ' ' AFTER OP CODE
32D0 C9        1150        RET
32D1 4E 5A     1160 CCTAB  DW   'ZN'     CONDITION CODE TABLE
32D3 5A 20     1170        DW   ' Z'
32D5 4E 43     1180        DW   'CN'
32D7 43 20     1190        DW   ' C'
32D9 50 4F     1200        DW   'OP'
32DB 50 45     1210        DW   'EP'
32DD 50 20     1220        DW   ' P'
32DF 4D 20     1230        DW   ' M'
32E1           1240 *
32E1           1250 * HEX CONVERSION ROUTINES
32E1           1260 *
32E1 1F        1270 HEXL  RAR  GET      LEFT DIGIT
32E2 1F        1280        RAR
32E3 1F        1290        RAR
32E4 1F        1300        RAR
32E5 E6 0F     1310 HEXR  ANI  0FH      GET RIGHT DIGIT
32E7 FE 0A     1320        CPI  10
32E9 DA EE 32  1330        JC   HEXRN
32EC C6 07     1340        ADI  7        FUDGE ALPHA HEX
32EE C6 30     1350 HEXRN ADI  '0'      MAKE PRINTABLE
32F0 C9        1360        RET
32F1           1370 *
32F1           1380 * HEX OUTPUT ROUTINE
32F1           1390 *
32F1 F5        1400 XO    PUSH 6        SAVE
32F2 CD E1 32  1410        CALL HEXL     GET LEFT NIBBLE
32F5 CD 73 33  1420        CALL ZTYPE    PRINT IT
32F8 F1        1430        POP  6        RESTORE
32F9 CD E5 32  1440        CALL HEXR     GET RIGHT NIBBLE
32FC C3 73 33  1450        JMP  ZTYPE    PRINT IT, RETURN
32FF           1460 *
32FF           1470 * HEX OUTPUT FOLLOWED BY A BLANK
32FF           1480 *
32FF CD F1 32  1490 XOB   CALL XO
3302 3E 20     1500        MVI  A,' '
3304 C3 73 33  1510        JMP  ZTYPE
3307           0010 *@260-5
3307           0020 *
3307           0030 * OUTPUT SPACE, QUOTE, CHAR IN A, RETURN
3307           0040 *
3307 F5        0050 OQ    PUSH PSW
3308 3E 20     0060        MVI  A,' '
330A CD 73 33  0070        CALL ZTYPE
330D 3E 27     0080        MVI  A,''''
330F CD 73 33  0090        CALL ZTYPE
3312 F1        0100        POP  PSW
3313 C3 73 33  0110        JMP  ZTYPE
3316           0120 *
3316           0130 * OUTPUT CHAR IN A, THEN CLOSING QUOTE
3316           0140 *
3316 CD 73 33  0150 CQ    CALL ZTYPE
3319 3E 27     0160        MVI  A,''''
331B C3 73 33  0170        JMP  ZTYPE
331E           0180 *
331E           0190 * PRINT 4 BYTE OP CODE FROM TABLE
331E           0200 *
331E D5        0210 PROP  PUSH D
331F 16 04     0220        MVI  D,4      OP CODE LENGTH
3321 03        0230 PROPL INX  B        OP CODE PTR
3322 0A        0240        LDAX B        GET OP CHAR
3323 CD 73 33  0250        CALL ZTYPE    PRINT IT
3326 15        0260        DCR  D        DONE?
3327 C2 21 33  0270        JNZ  PROPL    ..NO
332A 0A        0280        LDAX B        SPACE IF
332B FE 20     0290        CPI  ' '      ..LAST CHAR
332D 3E 20     0300        MVI  A,' '    ..OP CODE WAS
332F C4 73 33  0310        CNZ  ZTYPE    ..NOT A SPACE
```

```
3332 D1        0320        POP  D
3333 7E        0330        MOV  A,M     RETURN WITH OP
3334 C9        0340        RET  ,       CODE IN A
3335           0350 *
3335           0360 * PRINT XR IN BITS ..XXX...
3335           0370 *
3335 1F        0380 XR     RAR  ,       MOVE ..XXX...
3336 1F        0390        RAR  ,       ..REGISTER
3337 1F        0400        RAR  ,       ..TO .....XXX
3338 E6 06     0410        ANI  6       ISOLATE XR VALUE
333A FE 06     0420        CPI  6       IS IS 6?
333C C2 47 33  0430        JNZ  REG     ..NO
333F           0440 * PRINT 6 FOR PSW OR SP
333F 3E 36     0450        MVI  A,'6'   GET A 6,
3341 C3 73 33  0460        JMP  ZTYPE   PRINT IT
3344           0470 *
3344           0480 * PRINT REG IN BITS ..XXX...
3344           0490 *
3344 1F        0500 REGM   RAR  ,       RIGHT
3345 1F        0510        RAR  ,       ..ADJUST
3346 1F        0520        RAR
3347           0530 *
3347           0540 * PRINT REG IN BITS .....XXX
3347           0550 *
3347 E5        0560 REG    PUSH H
3348 21 5A 33  0570        LXI  H,RTAB  POINT HL TO REG TABLE
334B E6 07     0580        ANI  7       ISOLATE REG
334D CD 55 33  0590        CALL ADD     ADD REG # TO H,L
3350 7E        0600        MOV  A,M     GET ASCII REG
3351 E1        0610        POP  H
3352 C3 73 33  0620        JMP  ZTYPE
3355           0630 *
3355           0640 * ROUTINE TO ADD (A) TO (HL)
3355           0650 *
3355 85        0660 ADD    ADD  L       ADD IN A,
3356 6F        0670        MOV  L,A     MOVE BACK
3357 D0        0680        RNC  ,       RET IF NO CARRY TO H
3358 24        0690        INR  H       ADD CARRY TO H
3359 C9        0700        RET
335A           0710 *
335A           0720 * REGISTER TABLE
335A           0730 *
335A 42 43     0740 RTAB   DW   'CB'
335C 44 45     0750        DW   'ED'
335E 48 4C     0760        DW   'LH'
3360 4D 41     0770        DW   'AM'
3362           0780 *
3362           0790 * HEX OUTPUT W/ LEADING 0 IF REQ'D
3362           0800 *
3362 FE A0     0810 XOO    CPI  0A0H
3364 DA F1 32  0820        JC   XO
3367 F5        0830        PUSH PSW
3368 3E 30     0840        MVI  A,'0'
336A CD 73 33  0850        CALL ZTYPE
336D F1        0860        POP  PSW
336E C3 F1 32  0870        JMP  XO
3371           0880 *
3371           0890 * CARRIAGE RETURN ROUTINE
3371           0900 *
3371 3E 0D     0910 RET    MVI  A,13    GET C/R, FALL INTO ZTYPE
3373           0920 *
3373           0930 * OUTPUT ROUTINE - LOOKS AT
3373           0940 * SENSE SWITCH A15 TO DETERMINE
3373           0950 * IF A FILE IS BEING BUILT, OR IF
3373           0960 * ONLY A PRINTED LISTING IS DONE.
3373           0970 *
3373 C5        0980 ZTYPE PUSH B
3374 47        0990        MOV  B,A
```

```
3375 DB FF     1000        IN   -1
3377 17        1010        RAL
3378 78        1020        MOV  A,B
3379 C1        1030        POP  B
337A D2 9C 40  1040        JNC  TYPE    IF SS15 IS OFF
337D           1050 *
337D           1060 * SS 15 ON - STORE IN FILE
337D           1070 *
337D E5        1080        PUSH H
337E 2A 07 38  1090        LHLD FILE+2  CURR FILE PTR
3381 77        1100        MOV  M,A     SAVE CHAR
3382 23        1110        INX  H       POINT TO NEXT
3383 22 07 38  1120        SHLD FILE+2  SAVE PTR BACK
3386 F5        1130        PUSH 6
3387 CD 9C 40  1140        CALL TYPE    TYPE CHAR
338A F1        1150        POP  6
338B E1        1160        POP  H
338C           1170 *
338C           1180 * IF END OF LINE, UPDATE LINE LENGTH
338C           1190 *
338C FE 0D     1200        CPI  13
338E C0        1210        RNZ
338F E5        1220        PUSH H
3390           1230 *
3390           1240 * COMPUTE LINE LENGTH BY SUBTRACTING
3390           1250 * END OF LINE (STORED IN FILE+2) FROM START
3390           1260 * OF LINE (STORED IN 'LINAD')
3390           1270 *
3390 2A F1 35  1280        LHLD LINAD   GET COUNT
3393 22 F3 35  1290        SHLD LINUM   SAVE ADDR OF LINE #
3396 2B        1300        DCX  H
3397 7C        1310        MOV  A,H     COMPLEMENT IT,
3398 2F        1320        CMA  A       MOVING IT TO D,E
3399 57        1330        MOV  D,A
339A 7D        1340        MOV  A,L
339B 2F        1350        CMA  A
339C 5F        1360        MOV  E,A
339D 13        1370        INX  D       TURN 1'S COMPLEMENT INTO 2'S
339E 2A 07 38  1380        LHLD FILE+2
33A1 19        1390        DAD  D
33A2 7D        1400        MOV  A,L     GET COMPUTED LINE LENGTH
33A3 2A F1 35  1410        LHLD LINAD
33A6 2B        1420        DCX  H
33A7 77        1430        MOV  M,A
33A8 2A 07 38  1440        LHLD FILE+2
33AB 22 F1 35  1450        SHLD LINAD
33AE 23        1460        INX  H
33AF 22 07 38  1470        SHLD FILE+2
33B2 7C        1480        MOV  A,H
33B3           1490 *
33B3           1500 * HAVE WE RUN OUT OF MEMORY?
33B3           1510 *
33B3 FE 30     1520        CPI  HIFIL
33B5 E1        1530        POP  H
33B6 C2 D5 33  1540        JNZ  CKX     NO - CHECK CTL-X TO END
33B9           1550 *
33B9           1560 * END OF FILE
33B9           1570 *
33B9 2A 07 38  1580 END    LHLD FILE+2  GET END OF PROG ADDR
33BC 2B        1590        DCX  H       BACK UP TO START OF LINE
33BD 36 01     1600        MVI  M,1     STICK EOF CHAR IN MEMORY
33BF 22 07 38  1610        SHLD FILE+2  SAVE UPDATED PTR
33C2           1620 *
33C2           1630 * MOVE HIGH LINE NUMBER TO FILE TABLE
33C2           1640 *
33C2 2A F3 35  1650        LHLD LINUM   GET LINE # ADDRESS
33C5 06 04     1660        MVI  B,4     4 BYTE LINE #
33C7 11 09 38  1670        LXI  D,FILE+4 HI LINE NUMBER PTR
```

```
33CA 7E        1680 MOVLN MOV  A,M     GET DIGIT OF LINE NO.
33CB 12        1690       STAX D       SAVE IN TABLE
33CC 23        1700       INX  H
33CD 13        1710       INX  D
33CE 05        1720       DCR  B       DONE?
33CF C2 CA 33  1730       JNZ  MOVLN   NO
33D2 C3 1F 40  1740       JMP  SPVR    <---------- ONLY EXIT FROM PROG
33D5           1750 *
33D5           1760 * CHECK CONTROL-X TO ABORT
33D5           1770 * STORING OF FILE IN MEMORY
33D5           1780 *
33D5 DB 01     1790 CKX   IN   KEYB    <--------- NOTE INPUT
33D7 E6 7F     1800       ANI  7FH     DELETE PARITY
33D9 FE 18     1810       CPI  'X'-40H (CTL X)
33DB C0        1820       RNZ
33DC C3 B9 33  1830       JMP  END     END IF CTL-X TYPED
33DF           0010 *@280-2
33DF           0020 *
33DF           0030 * SYMBO - ROUTINE TO BUILD AND ACCESS THE SYMBOL TABLE
33DF           0040 *         SS10 ON - BUILD SYMBOL TABLE
33DF           0050 *         SS08 ON - PRINT LABELS FROM SYMBOL TABLE
33DF           0060 *
33DF DB FF     0070 SYMBO IN   -1      READ SSW
33E1 E6 05     0080       ANI  5       EITHER BUILD OR USE REQ'D?
33E3 C8        0090       RZ   ,       NO - HAVE PROGRAM USE DEFAULT LABEL
33E4 E6 01     0100       ANI  1       REQ TO USE TABLE?
33E6 C2 EE 33  0110       JNZ  FRMTB   YES. USE FROM TABLE
33E9           0120 *** BUILD LABEL PHASE ***
33E9 CD 11 34  0130       CALL SYBLD   PUT ENTRY IN TABLE
33EC B7        0140       ORA  A       ENSURE NO CARRY
33ED C9        0150       RET  ,       ..THEN RETURN (PGM USE DFLT LBL)
33EE           0160 *
33EE           0170 * PROGRAM REQUEST TO OUTPUT LABEL
33EE           0180 * BASED ON OPERAND POINTED TO BY H,L
33EE           0190 *
33EE 23        0200 FRMTB INX  H       POINT TO OPERAND
33EF 4E        0210       MOV  C,M     GET HI BYTE
33F0 23        0220       INX  H       POINT TO NEXT BYTE
33F1 46        0230       MOV  B,M     GET LO BYTE
33F2 23        0240       INX  H       POINT TO NEXT INSTR
33F3 CD FC 33  0250       CALL PRTLB   PRINT LBL IF IN TBL
33F6 D8        0260       RC   ,       RETURN IF PRINTED
33F7 2B        0270       DCX  H       BACK UP FOR PROG
33F8 2B        0280       DCX  H       ..TO PRINT LABEL
33F9 2B        0290       DCX  H       ..BECAUSE WASN'T IN TABLE
33FA B7        0300       ORA  A       TURN OFF CARRY
33FB C9        0310       RET
33FC CD 3E 34  0320 PRTLB CALL LOOKU   FIND IT
33FF 3F        0330       CMC  ,       CARRY OFF IF NOT FOUND
3400 D0        0340       RNC  ,       RETURN IF NO LABEL FOUND
3401           0350 *
3401           0360 * LABEL FOUND IN SYMBOL TABLE - PRINT IT
3401           0370 *
3401 06 05     0380       MVI  B,5     MAX LBL LENGTH
3403 1A        0390 SYPLP LDAX D
3404 13        0400       INX  D
3405 B7        0410       ORA  A
3406 37        0420       STC  ,       SO PGM WON'T PRT DFLT LBL
3407 C8        0430       RZ   ,       RET IF END OF SHORT LABEL
3408 CD 73 33  0440       CALL ZTYPE   TYPE CHAR
340B 05        0450       DCR  B
340C 37        0460       STC  ,       SO PGM WON'T PRT LBL
340D C8        0470       RZ   ,       RET IF END OF 5 CHAR LABEL
340E C3 03 34  0480       JMP  SYPLP   LOOP PRINTING
3411           0490 *
3411           0500 * SYMBOL TABLE BUILD
3411           0510 *
3411 23        0520 SYBLD INX  H       BUMP TO OPERAND
3412 4E        0530       MOV  C,M     LO BYTE
3413 23        0540       INX  H
3414 46        0550       MOV  B,M     HI BYTE
3415 2B        0560       DCX  H       RESTORE
3416 2B        0570       DCX  H       ..HL TO ORIG VALUE
3417 CD 3E 34  0580       CALL LOOKU   ALREADY IN TABLE?
341A D0        0590       RNC  ,       ..IF ALREADY IN TABLE
341B           0600 *
341B           0610 * ADD NEW ENTRY TO SYMBOL TABLE
341B           0620 * B,C CONTAINS ENTRY
341B           0630 * D,E POINTS TO OPEN SYMBOL TABLE ENTRY
341B           0640 *
341B 3E 4C     0650       MVI  A,'L'   DEFAULT TO LABEL 'LXXXX'
341D 12        0660       STAX D
341E 13        0670       INX  D
341F 78        0680       MOV  A,B     GET HI ORDER
3420 CD 31 34  0690       CALL SYSTX   STORE PRINTABLE HEX
3423 79        0700       MOV  A,C     GET LO
3424 CD 31 34  0710       CALL SYSTX   STORE IT
3427 78        0720       MOV  A,B     GET HI
3428 12        0730       STAX D       STORE IN TABLE
3429 13        0740       INX  D
342A 79        0750       MOV  A,C     GET LOW
342B 12        0760       STAX D       STORE IN TABLE
342C AF        0770       XRA  A       GET A XERO, STORE IT
342D 13        0780       INX  D       AT START OF NEXT LABEL
342E 12        0790       STAX D       TO SHOW END OF TABLE
342F 37        0800       STC  ,       SET CARRY SO PROGRAM
3430 C9        0810       RET  ,       ..WILL PRINT DEFAULT LABEL
3431           0820 *
3431           0830 * ROUTINE TO STORE LABEL IN SYMBOL TABLE
3431           0840 *
3431 F5        0850 SYSTX PUSH PSW
3432 CD E1 32  0860       CALL HEXL    CONV LEFT NIBBLE
3435 12        0870       STAX D       ..TO PRINTABLE, AND STORE
3436 13        0880       INX  D
3437 F1        0890       POP  PSW     RESTORE VALUE
3438 CD E5 32  0900       CALL HEXR    CONV RIGHT NIBBLE TO PRINTABLE
343B 12        0910       STAX D       SAVE IT
343C 13        0920       INX  D       POINT TO NEXT
343D C9        0930       RET
343E           0940 *
343E           0950 * SYMBOL TABLE LOOKUP ROUTINE
343E           0960 * B,C = HEX SYMBOL TO FIND
343E           0970 * D,E RETURN POINTING TO SYMBOL
343E           0980 * H,L RESTORED
343E           0990 * IF NOT FOUND, CARRY SET, AND
343E           1000 * D,E POINTS TO OPEN LOCATION
343E           1010 *
343E 11 02 C0  1020 LOOKU LXI  D,SYMTB POINT TO SYMBOL TABLE
3441 1A        1030 LULP  LDAX D       GET PRINTABLE LABEL
3442 B7        1040       ORA  A       IS IT ZERO?
3443 37        1050       STC  ,       SET CARRY AND
3444 C8        1060       RZ   ,       ..RETURN IF NOT FOUND
3445           1070 *** NON BLANK ENTRY FOUND. DOES IT MATCH? ***
3445 13        1080       INX  D       SKIP
3446 13        1090       INX  D       ..ASCII
3447 13        1100       INX  D       ..LABEL
3448 13        1110       INX  D       ..VALUE
3449 13        1120       INX  D
344A 1A        1130       LDAX D
344B B8        1140       CMP  B       MATCH?
344C 13        1150       INX  D
344D C2 5D 34  1160       JNZ  LUNO    NO MATCH
3450           1170 *** HI MATCHED, CHECK LO ***
3450 1A        1180       LDAX D
3451 B9        1190       CMP  C
3452 C2 5D 34  1200       JNZ  LUNO    NO MATCH
```

```
3455            1210 *** ENTRY FOUND ***
3455 1B         1220        DCX  D       BACK
3456 1B         1230        DCX  D       UP
3457 1B         1240        DCX  D       TO
3458 1B         1250        DCX  D       ASCII
3459 1B         1260        DCX  D       PART
345A 1B         1270        DCX  D       OF LABEL
345B B7         1280        ORA  A       ENSURE NO CARRY
345C C9         1290        RET  ,       NO CARRY IMPLIES FOUND
345D            1300 *** ENTRY DIDN'T MATCH - POINT TO NEXT ***
345D 13         1310 LUNO   INX  D       POINT TO START OF NEXT
345E C3 41 34   1320        JMP  LULP    LOOP BACK
3461            0010 *@2A0-3
3461            0020 *
3461 FF         0030 OPTAB  DB   -1
3462 00 00      0040        DW   0
3464 4E 4F      0050        DW   'ON'
3466 50 20      0060        DW   ' P'
3468 FF         0070        DB   -1
3469 07 00      0080        DW   7
346B 52 4C      0090        DW   'LR'
346D 43 20      0100        DW   ' C'
346F FF         0110        DB   -1
3470 0F 00      0120        DW   0FH
3472 52 52      0130        DW   'RR'
3474 43 20      0140        DW   ' C'
3476 FF         0150        DB   -1
3477 17 00      0160        DW   17H
3479 52 41      0170        DW   'AR'
347B 4C 20      0180        DW   ' L'
347D FF         0190        DB   -1
347E EB 00      0200        DW   0EBH
3480 58 43      0210        DW   'CX'
3482 48 47      0220        DW   'GH'
3484 FF         0230        DB   -1
3485 1F 00      0240        DW   1FH
3487 52 41      0250        DW   'AR'
3489 52 20      0260        DW   ' R'
348B FF         0270        DB   -1
348C 22 07      0280        DW   722H
348E 53 48      0290        DW   'HS'
3490 4C 44      0300        DW   'DL'
3492 FF         0310        DB   -1
3493 27 00      0320        DW   27H
3495 44 41      0330        DW   'AD'
3497 41 20      0340        DW   ' A'
3499 FF         0350        DB   -1
349A 2A 07      0360        DW   72AH
349C 4C 48      0370        DW   'HL'
349E 4C 44      0380        DW   'DL'
34A0 FF         0390        DB   -1
34A1 2F 00      0400        DW   2FH
34A3 43 4D      0410        DW   'MC'
34A5 41 20      0420        DW   ' A'
34A7 FF         0430        DB   -1
34A8 32 07      0440        DW   732H
34AA 53 54      0450        DW   'TS'
34AC 41 20      0460        DW   ' A'
34AE FF         0470        DB   -1
34AF 37 00      0480        DW   37H
34B1 53 54      0490        DW   'TS'
34B3 43 20      0500        DW   ' C'
34B5 FF         0510        DB   -1
34B6 3A 07      0520        DW   73AH
34B8 4C 44      0530        DW   'DL'
34BA 41 20      0540        DW   ' A'
34BC FF         0550        DB   -1
34BD 3F 00      0560        DW   3FH
34BF 43 4D      0570        DW   'MC'
34C1 43 20      0580        DW   ' C'
34C3 FF         0590        DB   -1
34C4 76 00      0600        DW   76H
34C6 48 4C      0610        DW   'LH'     HLT PRECEEDS MOV TO
34C8 54 20      0620        DW   ' T'     PREVENT 'MOV M,M'
34CA C0         0630        DB   0C0H
34CB 40 0B      0640        DW   0B40H
34CD 4D 4F      0650        DW   'OM'
34CF 56 20      0660        DW   ' V'
34D1 FF         0670        DB   -1
34D2 C3 07      0680        DW   7C3H
34D4 4A 4D      0690        DW   'MJ'
34D6 50 20      0700        DW   ' P'
34D8 FF         0710        DB   -1
34D9 C9 00      0720        DW   0C9H
34DB 52 45      0730        DW   'ER'
34DD 54 20      0740        DW   ' T'
34DF FF         0750        DB   -1
34E0 CD 07      0760        DW   7CDH
34E2 43 41      0770        DW   'AC'
34E4 4C 4C      0780        DW   'LL'
34E6 FF         0790        DB   -1
34E7 D3 05      0800        DW   5D3H
34E9 4F 55      0810        DW   'UO'
34EB 54 20      0820        DW   ' T'
34ED FF         0830        DB   -1
34EE DB 05      0840        DW   5DBH
34F0 49 4E      0850        DW   'NI'
34F2 20 20      0860        DW   '  '
34F4 FF         0870        DB   -1
34F5 E3 00      0880        DW   0E3H
34F7 58 54      0890        DW   'TX'
34F9 48 4C      0900        DW   'LH'
34FB FF         0910        DB   -1
34FC E9 00      0920        DW   0E9H
34FE 50 43      0930        DW   'CP'
3500 48 4C      0940        DW   'LH'
3502 FF         0950        DB   -1
3503 F3 00      0960        DW   0F3H
3505 44 49      0970        DW   'ID'
3507 20 20      0980        DW   '  '
3509 FF         0990        DB   -1
350A FB 00      1000        DW   0FBH
350C 45 49      1010        DW   'IE'
350E 20 20      1020        DW   '  '
3510 FF         1030        DB   -1
3511 F9 00      1040        DW   0F9H
3513 53 50      1050        DW   'PS'
3515 48 4C      1060        DW   'LH'
3517 FF         1070        DB   -1
3518 C6 05      1080        DW   5C6H
351A 41 44      1090        DW   'DA'
351C 49 20      1100        DW   ' I'
351E FF         1110        DB   -1
351F CE 05      1120        DW   5CEH
3521 41 43      1130        DW   'CA'
3523 49 20      1140        DW   ' I'
3525 FF         1150        DB   -1
3526 D6 05      1160        DW   5D6H
3528 53 55      1170        DW   'US'
352A 49 20      1180        DW   ' I'
352C FF         1190        DB   -1
352D DE 05      1200        DW   5DEH
352F 53 42      1210        DW   'BS'
3531 49 20      1220        DW   ' I'
3533 FF         1230        DB   -1
3534 E6 05      1240        DW   5E6H
```

```
3536 41 4E    1250        DW    'NA'
3538 49 20    1260        DW    ' I'
353A FF       1270        DB    -1
353B EE 05    1280        DW    5EEH
353D 58 52    1290        DW    'RX'
353F 49 20    1300        DW    ' I'
3541 FF       1310        DB    -1
3542 F6 05    1320        DW    5F6H
3544 4F 52    1330        DW    'RO'
3546 49 20    1340        DW    ' I'
3548 FF       1350        DB    -1
3549 FE 05    1360        DW    5FEH
354B 43 50    1370        DW    'PC'
354D 49 20    1380        DW    ' I'
354F F8       1390        DB    -8
3550 80 01    1400        DW    180H
3552 41 44    1410        DW    'DA'
3554 44 20    1420        DW    ' D'
3556 F8       1430        DB    -8
3557 88 01    1440        DW    188H
3559 41 44    1450        DW    'DA'
355B 43 20    1460        DW    ' C'
355D F8       1470        DB    -8
355E 90 01    1480        DW    190H
3560 53 55    1490        DW    'US'
3562 42 20    1500        DW    ' B'
3564 F8       1510        DB    -8
3565 98 01    1520        DW    198H
3567 53 42    1530        DW    'BS'
3569 42 20    1540        DW    ' B'
356B F8       1550        DB    -8
356C A0 01    1560        DW    1A0H
356E 41 4E    1570        DW    'NA'
3570 41 20    1580        DW    ' A'
3572 F8       1590        DB    -8
3573 A8 01    1600        DW    1A8H
3575 58 52    1610        DW    'RX'
3577 41 20    1620        DW    ' A'
3579 F8       1630        DB    -8
357A B0 01    1640        DW    1B0H
357C 4F 52    1650        DW    'RO'
357E 41 20    1660        DW    ' A'
3580 F8       1670        DB    -8
3581 B8 01    1680        DW    1B8H
3583 43 4D    1690        DW    'MC'
3585 50 20    1700        DW    ' P'
3587 C7       1710        DB    0C7H
3588 04 02    1720        DW    204H
358A 49 4E    1730        DW    'NI'
358C 52 20    1740        DW    ' R'
358E C7       1750        DB    0C7H
358F 05 02    1760        DW    205H
3591 44 43    1770        DW    'CD'
3593 52 20    1780        DW    ' R'
3595 EF       1790        DB    0EFH
3596 02 03    1800        DW    302H
3598 53 54    1810        DW    'TS'
359A 41 58    1820        DW    'XA'
359C EF       1830        DB    0EFH
359D 0A 03    1840        DW    30AH
359F 4C 44    1850        DW    'DL'
35A1 41 58    1860        DW    'XA'
35A3 CF       1870        DB    0CFH
35A4 C1 03    1880        DW    3C1H
35A6 50 4F    1890        DW    'OP'
35A8 50 20    1900        DW    ' P'
35AA CF       1910        DB    0CFH
35AB C5 03    1920        DW    3C5H
35AD 50 55    1930        DW    'UP'
35AF 53 48    1940        DW    'HS'
35B1 C7       1950        DB    0C7H
35B2 06 04    1960        DW    406H
35B4 4D 56    1970        DW    'VM'
35B6 49 20    1980        DW    ' I'
35B8 CF       1990        DB    0CFH
35B9 03 03    2000        DW    303H
35BB 49 4E    2010        DW    'NI'
35BD 58 20    2020        DW    ' X'
35BF CF       2030        DB    0CFH
35C0 09 03    2040        DW    309H
35C2 44 41    2050        DW    'AD'
35C4 44 20    2060        DW    ' D'
35C6 CF       2070        DB    0CFH
35C7 0B 03    2080        DW    30BH
35C9 44 43    2090        DW    'CD'
35CB 58 20    2100        DW    ' X'
35CD C7       2110        DB    0C7H
35CE C2 08    2120        DW    8C2H
35D0 4A 2E    2130        DW    '.J'
35D2 2E 2E    2140        DW    '..'
35D4 C7       2150        DB    0C7H
35D5 C4 08    2160        DW    8C4H
35D7 43 2E    2170        DW    '.C'
35D9 2E 2E    2180        DW    '..'
35DB CF       2190        DB    0CFH
35DC 01 06    2200        DW    601H
35DE 4C 58    2210        DW    'XL'
35E0 49 20    2220        DW    ' I'
35E2 C7       2230        DB    0C7H
35E3 C0 09    2240        DW    9C0H
35E5 52 2E    2250        DW    '.R'
35E7 2E 2E    2260        DW    '..'
35E9 C7       2270        DB    0C7H
35EA C7 0A    2280        DW    0AC7H
35EC 52 53    2290        DW    'SR'
35EE 54 20    2300        DW    ' T'
35F0 00       2310        DB    0       (END OF TABLE)
35F1          2320 *----------WORK AREA----------*
35F1          2330 * NOTE: DISASSEMBLER IS ROM-ABLE
35F1          2340 * (= PROTECTABLE) IF THIS AREA
35F1          2350 * IS MOVED TO READ-WRITE MEMORY.


35F1          2360 LINAD DS   2       ADDR OF START OF CUR LINE
35F3          2370 LINUM DS   2       ADDR OF CUR LINE'S LINE #
35F5          2380 TYFLG DS   1       OP CODE TYPE
35F6          2390 CC1   DS   3       CONDITION STORED HERE
35F9          2400 CCB   DS   1       BLANK FOLLOWING COND CODE



* DISASSEMBLER SAMPLE RUN - USING THE DISASSEMBLER OBJECT
* PROGRAM AS THE PROGRAM TO BE DISASSEMBLED.

* THIS WILL ALLOW YOU TO COMPARE THE DISASSEMBLED PROGRAM
* AGAINST THE ORIGINAL SOURCE PROGRAM.
*                FOR THIS EXAMPLE:

* THE DISASSEMBLER IS ASSEMBLED TO 3100,
* THE SYMBOL TABLE EDIT ROUTINE TO 3600
```

```
****** DISASSEMBLE THE FIRST PART OF THE DISASSEMBLER
***** ALL SS (SENSE SWITCHES) DOWN

EXEC 3100 3100          <- Addr of disassembler program
                        <- Addr to be disassembled
3100  LHLD 3805H
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  310EH
310B  SHLD 3807H
310E  LHLD 3868H
3111  CALL 3117H
3114  JMP 3111H
3117  PUSH H
3118

***** NOW THE SAME, WITH SWITCH 08H UP:

EXEC 3100 3100
3100  LHLD L3805
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  L310E
310B  SHLD L3807
310E  LHLD L3868
3111  CALL L3117
3114  JMP L3111
3117  PUSH H
3

***** NOTE THE SUBSTITUTION OF LABELS (Lxxxx) FOR THE HEX.


***** NOW, SS 04 UP TO BUILD SYMBOL TABLE:
EXEC 3100 3100
3100  LHLD 3805H
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  310EH
310B  SHLD 3807H
310E  LHLD 3868H
3111  CALL 3117H
3114  JMP 3111H
3117  PUSH H
3118




* NOW, USE THE SYMBOL TABLE, BY TURNING ON SS 01

EXEC 3100 3100
3100  LHLD L3805
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  L310E
310B  SHLD L3807
310E L310E LHLD L3868
3111 L3111 CALL L3117
3114  JMP L3111
3117 L3117 PUSH H
3118
```

```
* NOW, EDIT THE SYMBOL TABLE TO PUT 'MORE MEANINGFUL' LABELS IN

EXEC 3600 0          E   <- SYMBOL TABLE edit routine addr.
L3805 3805 3805H         <- Relative symbol # to
L310E 310E NOSTR             start edit with
L3807 3807 3807H
L3868 3868 BBUF
L3117 3117 LINE
L3111 3111 DISAM

* NOW, AGAIN W/SS01 UP, - -

EXEC 3100 3100
3100  LHLD 3805H
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  NOSTR
310B  SHLD 3807H
310E NOSTR LHLD BBUF
3111 DISAM CALL LINE
3114  JMP DISAM
3117 LINE PUSH H
3

* NOW, TURN ON SS 80H TO WRITE FILE INTO MEMORY

EXEC 3100 3100
3100  LHLD 3805H
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  NOSTR
310B  SHLD 3807H
310E NOSTR LHLD BBUF
3111 DISAM CALL LINE
3114  JMP DISAM
3117 LINE PUSH H


* NOTE THE FILE THUS CREATED:

LIST
3100  LHLD 3805H
3103  INX H
3104  IN  0FFH
3106  ANI 80H
3108  JZ  NOSTR
310B  SHLD 3807H
310E NOSTR LHLD BBUF
3111 DISAM CALL LINE
3114  JMP DISAM
3117 LINE PUSH H

* NOW, RE-ASSEMBLE THE FILE

ASSM 3100

3100 2A 05 38 3100        LHLD 3805H
3103 23       3103        INX  H
3104 DB FF    3104        IN   0FFH
3106 E6 80    3106        ANI  80H
3108 CA 0E 31 3108        JZ   NOSTR
310B 22 07 38 310B        SHLD 3807H
310E 2A 00 00 310ENOSTR   LHLD BBUF
3111 CD 17 31 3111 DISAM  CALL LINE
3114 C3 11 31 3114        JMP  DISAM
3117 E5       3117 LINE   PUSH H
```

```
* GOT AN ERROR - BBUF UNDEFINED -
* SO ADD A LINE TO THE ASSEMBLY TO DEFINE IT

0010 BBUF EQU 3868H
* RE-ASSEMBLE

ASSM 3100 F000

3100 2A 05 38 3100        LHLD 3805H
3103 23         3103      INX  H
3104 DB FF      3104      IN   0FFH
3106 E6 80      3106      ANI  80H
3108 CA 0E 31 3108        JZ   NOSTR
310B 22 07 38 310B        SHLD 3807H
310E 2A 68 38 310E NOSTR  LHLD BBUF
3111 CD 17 31 3111 DISAM  CALL LINE
3114 C3 11 31 3114        JMP  DISAM
3117 E5         3117 LINE PUSH H
3118            0010 BBUF EQU  3868H



3600            0010 *U2C0 1
3600            0020 * * * * * * * * * * * * * * * * *
3600            0030 *  SYMBOL TABLE EDIT ROUTINE  *
3600            0040 *         FOR USE WITH       *
3600            0050 *     8 0 8 0  DISASSEMBLER  *
3600            0060 * * * * * * * * * * * * * * * * *
3600            0070 PSW    EQU  6
3600            0080 TYPE   EQU  409CH   TYPE ASCII FROM A
3600            0090 KEYIN  EQU  408EH   KEYIN ASCII TO A
3600            0100 *
3600            0110 * PLACE THE ENTRY NUMBER FROM THE
3600            0120 * SYMBOL TABLE AT WHICH YOU WANT TO
3600            0130 * START EDITING INTO THE FOLLOWING
3600            0140 * WORD (0 FOR START OF TABLE)
3600            0150 BBUF   EQU  3868H   PTCO PARM BUFFER+2
3600            0160 *
3600            0170 SYMTB  EQU  0C002H  SYMBOL TABLE START
3600 2A 68 38   0180        LHLD BBUF    GET TABLE ENTRY #
3603 54         0190        MOV  D,H     MOVE ENTRY #
3604 5D         0200        MOV  E,L     ..TO D,E
3605            0210 * MULTIPLY ENTRY BY 7 BYTES/ENTRY
3605 29         0220        DAD  H       TIMES 2
3606 19         0230        DAD  D       TIMES 3
3607 29         0240        DAD  H       TIMES 6
3608 19         0250        DAD  D       TIMES 7
3609 11 02 C0   0260        LXI  D,SYMTB POINT DE TO TABLE
360C 19         0270        DAD  D       POINT HL TO PROPER ENTRY
360D 7E         0280 LABEL  MOV  A,M     GET FIRST CHAR
360E B7         0290        ORA  A       END OF TABLE?
360F C8         0300        RZ   ,       RETURN TO CALLER IF SO
3610            0310 *** PRINT EXISTING LABEL ***
3610 06 05      0320        MVI  B,5     MAX LABEL LENGTH
3612 7E         0330 PRINT  MOV  A,M     GET CHAR OF LABEL
3613 B7         0340        ORA  A       END OF SHORT LABEL?
3614 CA 1F 36   0350        JZ   ENDP    ..YES - END OF PRINT
3617 CD 9C 40   0360        CALL TYPE    TYPE THE CHAR
361A 23         0370        INX  H       POINT TO NEXT
361B 05         0380        DCR  B       DONE?
361C C2 12 36   0390        JNZ  PRINT   NO
361F            0400 *** END OF LABEL
361F 78         0410 ENDP   MOV  A,B     GET LABEL RESIDUAL COUNT
3620 85         0420        ADD  L       ADD TO H,L
3621 6F         0430        MOV  L,A     MOVE IT BACK
3622 D2 26 36   0440        JNC  NC      SKIP ADD IF NO CARRY INTO H
3625 24         0450        INR  H       ADD IN CARRY
3626 CD 87 36   0460 NC     CALL SPACE   PRINT A SPACE
3629            0470 *** PRINT HEX LABEL VALUE
3629 7E         0480        MOV  A,M     FIRST CHAR
362A CD 92 36   0490        CALL XO      PRINT IN HEX
362D 23         0500        INX  H       POINT TO SECOND CHAR
362E 7E         0510        MOV  A,M     SECOND CHAR
362F CD 8C 36   0520        CALL XOB     SECOND CHAR
3632 11 FA FF   0530        LXI  D,-6    BACK HL UP TO
3635 19         0540        DAD  D       ..START OF LABEL
3636 E5         0550        PUSH H       SAVE START OF LBL ADDR
3637            0560 *
3637            0570 * READ NEW LABEL INTO TABLE
3637            0580 *


3637 16 05      0590        MVI  D,5     MAX LABEL LENGTH
3639 CD 8E 40   0600 INPLP  CALL KEYIN   GET CHAR FOR LABEL
363C FE 18      0610        CPI  'X'-40H CTL-X?
363E CA A9 36   0620        JZ   ABORT   RETURN IF CTL-X
3641 F5         0630        PUSH PSW     SAVE THE CHAR
3642 CD 9C 40   0640        CALL TYPE    ECHO IT
3645 F1         0650        POP  PSW     RESTORE THE CHAR
3646 FE 7F      0660        CPI  7FH     DEL CHAR? (BACKSPACE?)
3648 C2 5E 36   0670        JNZ  NOTBS   ..NO
364B            0680 * PROCESS BACKSPACE
364B 2B         0690        DCX  H       BACK UP 1 CHAR
364C 14         0700        INR  D       ADJUST COUNT
364D            0710 * PRINT DELETED CHAR IN SLASHES
364D 3E 2F      0720        MVI  A,'/'
364F CD 9C 40   0730        CALL TYPE
3652 7E         0740        MOV  A,M     GET DELETED CHAR
3653 CD 9C 40   0750        CALL TYPE
3656 3E 2F      0760        MVI  A,'/'
3658 CD 9C 40   0770        CALL TYPE
365B C3 39 36   0780        JMP  INPLP
365E FE 0D      0790 NOTBS  CPI  13      C/R?
3660 CA 6E 36   0800        JZ   ENDLB   YES - END OF LABEL
3663 77         0810        MOV  M,A     STORE CHAR INTO TABLE
3664 23         0820        INX  H       POINT TO NEXT POSITION
3665 15         0830        DCR  D       5 CHARS?
3666 C2 39 36   0840        JNZ  INPLP   NO - GO GET MORE
3669 3E 0D      0850        MVI  A,13    GET ASCII C/R
366B CD 9C 40   0860        CALL TYPE    C/R
366E            0870 *** END OF LABEL, PAD W/0'S IF < 5 CHARS
366E 7A         0880 ENDLB  MOV  A,D     GET RESIDUAL LABEL COUNT
366F B7         0890        ORA  A       ZERO?
3670 CA 7F 36   0900        JZ   NOLAB   YES - DON'T STORE THE ZERO
3673 FE 05      0910        CPI  5       NO LABEL CHANGE?
3675 CA 7F 36   0920        JZ   NOLAB   YES - LEAVE ALONE
3678 AF         0930        XRA  A       GET A ZERO
3679 77         0940        MOV  M,A     STORE INTO TABLE
367A 23         0950 PADLB  INX  H       POINT TO NEXT POSITION
367B 15         0960        DCR  D       STILL MORE RESIDUAL COUNT?
367C C2 7A 36   0970        JNZ  PADLB   STORE MORE UNTIL DONE
367F E1         0980 NOLAB  POP  H       RESTORE START OF LBL ADDR
3680 11 07 00   0990        LXI  D,7     5 FOR LABEL, 2 FOR VALUE
3683 19         1000        DAD  D       POINT TO NEXT LABEL
3684 C3 0D 36   1010        JMP  LABEL   LOOP FOR NEXT LABEL
3687            1020 *
3687            1030 * ROUTINE TO PRINT A SPACE
3687            1040 *
3687 3E 20      1050 SPACE  MVI  A,' '
3689 C3 9C 40   1060        JMP  TYPE    TYPE THE SPACE
368C            1070 *
368C            1080 * HEX OUTPUT ROUTINES
368C            1090 *
368C            1100 *---> HEX OUTPUT W/BLANK
368C CD 92 36   1110 XOB    CALL XO      PRINT HEX
```

```
368F C3 87 36 1120        JMP  SPACE    ..THEN SPACE
3692           1130 *---> HEX OUTPUT
3692 F5        1140 XO    PUSH 6        SAVE INCOMING A REG
3693 1F        1150       RAR  ,        SHIFT
3694 1F        1160       RAR  ,        ..LEFT NIBBLE
3695 1F        1170       RAR  ,        ..INTO THE
3696 1F        1180       RAR  ,        ..THE RIGHT NIBBLE

3697 CD 9B 36 1190        CALL NIBBL    PRINT LEFT NIBBLE
369A F1        1200       POP  6        RESTORE CHAR
369B           1210 *---> PRINT 1 HEX NIBBLE
369B           1220 *     (NIBBLE IS HALF A BYTE)
369B E6 0F     1230 NIBBL ANI  15       ISOLATE NIBBLE
369D FE 0A     1240       CPI  10       ALPHA HEX?
369F DA A4 36  1250       JC   HEXNU    NO, NUMBER
36A2 C6 07     1260       ADI  7        FUDGE FOR ALPHA HEX
36A4 C6 30     1270 HEXNU ADI  '0'      MAKE PRINTABLE
36A6 C3 9C 40  1280       JMP  TYPE     PRINT NIBBLE, RETURN
36A9 E1        1290 ABORT POP  H        DELETE PUSHED H
36AA C9        1300       RET  ,        RETURN TO CALLER
```

DISASSEMBLER PAGE 1 OF 2

```
DUMP 3100 349F
3100 DB FF E6 04 CA 0B 31 AF 32 02 C0 2A 05 38 23 DB
3110 FF E6 80 CA 19 31 22 07 38 2A 68 38 CD 22 31 C3
3120 1C 31 E5 2A 07 38 22 F1 35 E1 7C CD F1 32 7D CD
3130 FF 32 44 4D DB FF E6 01 C4 FC 33 3E 20 CD 73 33
3140 DB FF E6 20 C2 87 32 E5 66 01 61 34 0A B7 CA 86
3150 32 A4 6F 03 0A BD CA 64 31 79 C6 06 4F D2 4C 31
3160 04 C3 4C 31 03 0A 32 F5 35 D5 5F 16 00 21 76 31
3170 19 19 19 D1 E3 C9 C3 9A 31 C3 A1 31 C3 AA 31 C3
3180 B3 31 C3 BC 31 C3 C8 31 C3 09 32 C3 2A 32 C3 4E
3190 32 C3 59 32 C3 64 32 C3 74 32 CD 1E 33 23 C3 71
31A0 33 CD 1E 33 CD 47 33 C3 9D 31 CD 1E 33 CD 44 33
31B0 C3 9D 31 CD 1E 33 CD 35 33 C3 9D 31 CD 1E 33 CD
31C0 44 33 CD 81 32 C3 CB 31 CD 1E 33 23 7E CD 62 33
31D0 3E 48 CD 73 33 7E 23 C5 47 DB FF E6 10 78 C1 CA
31E0 71 33 FE 20 DA 71 33 FE 5B D2 71 33 3A F5 35 FE
31F0 04 DA 71 33 FE 06 D2 71 33 2B 7E 23 F5 CD 07 33
3200 F1 3E 27 CD 73 33 C3 71 33 CD 1E 33 7E CD 35 33
3210 CD 81 32 C3 1C 32 CD DF 33 DA 71 33 23 46 23 7E
3220 CD 62 33 78 CD F1 32 C3 D0 31 CD 1E 33 CD DF 33
3230 DA 71 33 DB FF E6 08 CA 16 32 3E 4C CD 73 33 23
3240 46 23 7E CD F1 32 78 CD F1 32 23 C3 71 33 03 0A
3250 32 F6 35 CD B2 32 C3 2A 32 3E 52 32 F6 35 CD B2
3260 32 C3 9A 31 CD 1E 33 1F 1F 1F E6 07 F6 30 CD 73
3270 33 C3 9D 31 CD 1E 33 CD 44 33 CD 81 32 7E C3 A4
3280 31 3E 2C C3 73 33 E1 3E 44 CD 73 33 3E 42 CD 73
3290 33 3E 20 CD 73 33 2B DB FF E6 30 FE 30 C2 CB 31
32A0 23 7E 2B FE 20 DA CB 31 FE 5B D2 CB 31 23 23 C3
32B0 FC 31 7E 1F 1F E6 0E E5 21 D1 32 CD 55 33 7E 32
32C0 F7 35 23 7E 32 F8 35 E1 01 F5 35 3E 20 32 F9 35
32D0 C9 4E 5A 5A 20 4E 43 43 20 50 4F 50 45 50 20 4D
32E0 20 1F 1F 1F 1F E6 0F FE 0A DA EE 32 C6 07 C6 30
32F0 C9 F5 CD E1 32 CD 73 33 F1 CD E5 32 C3 73 33 CD
3300 F1 32 3E 20 C3 73 33 F5 3E 20 CD 73 33 3E 27 CD
3310 73 33 F1 C3 73 33 CD 73 33 3E 27 C3 73 33 D5 16
3320 04 03 0A CD 73 33 15 C2 21 33 0A FE 20 3E 20 C4
3330 73 33 D1 7E C9 1F 1F 1F E6 06 FE 06 C2 47 33 3E
3340 36 C3 73 33 1F 1F 1F E5 21 5A 33 E6 07 CD 55 33
3350 7E E1 C3 73 33 85 6F D0 24 C9 42 43 44 45 48 4C
3360 4D 41 FE A0 DA F1 32 F5 3E 30 CD 73 33 F1 C3 F1
3370 32 3E 0D C5 47 DB FF 17 78 C1 D2 9C 40 E5 2A 07
3380 38 77 23 22 07 38 F5 CD 9C 40 F1 E1 FE 0D C0 E5
3390 2A F1 35 22 F3 35 2B 7C 2F 57 7D 2F 5F 13 2A 07
33A0 38 19 7D 2A F1 35 2B 77 2A 07 38 22 F1 35 23 22
33B0 07 38 7C FE 30 E1 C2 D5 33 2A 07 38 2B 36 01 22
33C0 07 38 2A F3 35 06 04 11 09 38 7E 12 23 13 05 C2
33D0 CA 33 C3 1F 40 DB 01 E6 7F FE 18 C0 C3 B9 33 DB
33E0 FF E6 05 C8 E6 01 C2 EE 33 CD 11 34 B7 C9 23 4E
33F0 23 46 23 CD FC 33 D8 2B 2B 2B B7 C9 CD 3E 34 3F
3400 D0 06 05 1A 13 B7 37 C8 CD 73 33 05 37 C8 C3 03
3410 34 23 4E 23 46 2B 2B CD 3E 34 D0 3E 4C 12 13 78
3420 CD 31 34 79 CD 31 34 78 12 13 79 12 AF 13 12 37
3430 C9 F5 CD E1 32 12 13 F1 CD E5 32 12 13 C9 11 02
3440 C0 1A B7 37 C8 13 13 13 13 13 1A B8 13 C2 5D 34
3450 1A B9 C2 5D 34 1B 1B 1B 1B 1B B7 C9 13 C3 41
3460 34 FF 00 00 4E 4F 50 20 FF 07 00 52 4C 43 20 FF
3470 0F 00 52 52 43 20 FF 17 00 52 41 4C 20 FF EB 00
3480 58 43 48 47 FF 1F 00 52 41 52 20 FF 22 07 53 48
3490 4C 44 FF 27 00 44 41 41 20 FF 2A 07 4C 48 4C 44
```

DISASSEMBLER PAGE 2 OF 2

```
DUMP 34A0 35FF
34A0 FF 2F 00 43 4D 41 20 FF 32 07 53 54 41 20 FF 37
34B0 00 53 54 43 20 FF 3A 07 4C 44 41 20 FF 3F 00 43
34C0 4D 43 20 FF 76 00 48 4C 54 20 C0 40 0B 4D 4F 56
34D0 20 FF C3 07 4A 4D 50 20 FF C9 00 52 45 54 20 FF
34E0 CD 07 43 41 4C 4C FF D3 05 4F 55 54 20 FF DB 05
34F0 49 4E 20 20 FF E3 00 58 54 48 4C FF E9 00 50 43
3500 48 4C FF F3 00 44 49 20 20 FF FB 00 45 49 20 20
3510 FF F9 00 53 50 48 4C FF C6 05 41 44 49 20 FF CE
3520 05 41 43 49 20 FF D6 05 53 55 49 20 FF DE 05 53
3530 42 49 20 FF E6 05 41 4E 49 20 FF EE 05 58 52 49
3540 20 FF F6 05 4F 52 49 20 FF FE 05 43 50 49 20 F8
3550 80 01 41 44 44 20 F8 88 01 41 44 43 20 F8 90 01
3560 53 55 42 20 F8 98 01 53 42 42 20 F8 A0 01 41 4E
3570 41 20 F8 A8 01 58 52 41 20 F8 B0 01 4F 52 41 20
3580 F8 B8 01 43 4D 50 20 C7 04 02 49 4E 52 20 C7 05
3590 02 44 43 52 20 EF 02 03 53 54 41 58 EF 0A 03 4C
35A0 44 41 58 CF C1 03 50 4F 50 20 CF C5 03 50 55 53
35B0 48 C7 06 04 4D 56 49 20 CF 03 03 49 4E 58 20 CF
35C0 09 03 44 41 44 20 CF 0B 03 44 43 58 20 C7 C2 08
35D0 4A 2E 2E 2E C7 C4 08 43 2E 2E 2E CF 01 06 4C 58
35E0 49 20 C7 C0 09 52 2E 2E 2E C7 C7 0A 52 53 54 20
35F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

SYMBOL TABLE EDIT ROUTINE

```
DUMP 3600 36AA
3600 2A 68 38 54 5D 29 19 29 19 11 02 C0 19 7E B7 C8
3610 06 05 7E B7 CA 1F 36 CD 9C 40 23 05 C2 12 36 78
3620 85 6F D2 26 36 24 CD 87 36 7E CD 92 36 23 7E CD
3630 8C 36 11 FA FF 19 E5 16 05 CD 8E 40 FE 18 CA A9
3640 36 F5 CD 9C 40 F1 FE 7F C2 5E 36 2B 14 3E 2F CD
3650 9C 40 7E CD 9C 40 3E 2F CD 9C 40 C3 39 36 FE 0D
3660 CA 6E 36 77 23 15 C2 39 36 3E 0D CD 9C 40 7A B7
3670 CA 7F 36 FE 05 CA 7F 36 AF 77 23 15 C2 7A 36 E1
3680 11 07 00 19 C3 0D 36 3E 20 C3 9C 40 CD 92 36 C3
3690 87 36 F5 1F 1F 1F 1F CD 9B 36 F1 E6 0F FE 0A DA
36A0 A4 36 C6 07 C6 30 C3 9C 40 E1 C9
```

APPENDICES TO CASUAL

Dear Jim, January 15, 1977

Thank you for printing CASUAL in your Nov/Dec issue.

From one of our early phone conversations, I got the impression that there was a deadline to meet in order to get CASUAL printed. I sent off the finalized source code at that time. I reluctantly sent along the partially completed documentation. Within a week, I sent the finalized documentation which either missed the deadline or it was not clear that it superseded the old documentation. In any case, it wasn't printed.

Fortunately, most of the errors in the early version were spelling and grammar. The biggest change was the addition of seven (7) more appendices. A copy of these appendices is enclosed in the hopes that you can find room for them in a future issue. If they are printed the same size as the first ones, they'll take about 1.2 pages.

The printed version was unclear in Appendix G about the address of USRL. Double indirect addressing is used. Memory locations 3 and 4 hold the address of the bytes where you put the starting address of your routine. This is so the address of USRL may change from version to version, but its address will always be held in locations 3 and 4.

Another error occurred in Appendix H. An ASCII tab character is 9 (decimal), not 11 as printed.

I would be interested in hearing from any readers any other errors or bugs.

I hope that aspiring authors can learn from my mistakes:

1) Don't send in anything until it is finalized.

2) Put a version number and/or date on everything sent in. In this way, corrections may be sent in labeled "this supersedes version . . . ".

Thanks,

Bob Van Valzah

1140 Hickory Trl.
Downers Grove, IL 60515
H-(312) 852-0472
W-(312) 971-2010 x231

```
        APPENDIX I
        ----------
        COMMANDS

        NEW COMMAND
THE NEW COMMAND CLEARS THE CASUAL PROGRAM BUFFER OF THE
CURRENT PROGRAM.  IT DOES NOT CLEAR VARIABLE VALUES.  IT
MAY APPEAR IN A PROGRAM, THOUGH IT IS SUICIDAL.
FORMAT:         N (CR)

        SAVE AND TAPE COMMANDS
THESE COMMANDS ALLOW THE CONTENTS OF THE CASUAL PROGRAM BUFFER
TO BE SAVED ON A MASS STORAGE DEVICE AND RECALLED FOR LATER
USE.  ROGRAMS MAY BE GIVEN A SINGLE CHARACTER NAME AT THE TIME
THEY ARE "SAVED".  THIS NAME IS THEN USED TO RECALL THE
PROGRAM.  BY USING DIFFERENT NAMES, MANY PROGRAMS MAY BE SAVED
ON THE SAME TAPE (CASSETTE), AND CASUAL WILL SEARCH FOR THE
REQUESTED PROGRAM AT LOAD TIME.  IF NO NAME IS GIVEN IN THE
TAPE COMMAND, THE FIRST PROGRAM FOUND ON THE TAPE WILL BE
LOADED.  THIS FEATURE IS USED TO LOAD PROGRAMS WITH UNKNOWN
NAMES.  IF NO NAME IS GIVEN WHEN THE PROGRAM IS SAVED, IT
MUST BE LOADED WITH NO NAME.  ANY PRINTING CHARACTER IS LEGAL
AS A PROGRAM NAME.  WHEN LOADING, CASUAL WILL PRINT THE
NAME OF THE PROGRAM WHICH IS BEING LOADED AS SOON AS LOADING
STARTS.  BOTH COMMANDS MAY APPEAR IN A PROGRAM, BUT WILL
RETURN TO COMMAND MODE WHEN DONE.  AN OUT OF MEMORY ERROR IS
POSSIBLE DURING LOADING.
FORMAT:         T<NAME> (CR)
                S<NAME> (CR)

        OS COMMAND
LEAVES CASUAL AND GOES TO AN OPERATING SYSTEM ROM.  WHEN
LOADED, CASUAL WILL JUMP TO 374 000 (SPLIT OCTAL) WHEN THIS
COMMAND IS FOUND.
FORMAT:         O (CR)

        RUN COMMAND
THIS COMMAND CAUSES CASUAL TO START INTERPRETING THE PROGRAM
CURRENTLY IN THE CASUAL PROGRAM BUFFER.  IT MAY OPTIONALLY BE
FOLLOWED BY A LINE NUMBER WHERE EXECUTION IS TO START.  IF
NO LINE NUMBER IS GIVEN, EXECUTION WILL START AT THE LOWEST
NUMBERED LINE.  THE LINE NUMBER MAY BE AN EXPRESSION.  THIS
COMMAND MAY APPEAR IN A PROGRAM.
FORMAT:         R (EXPR) (CR)

        LIST COMMAND
THIS COMMAND IS USED TO LIST THE CONTENTS OF THE CASUAL
PROGRAM BUFFER.  IT IS OPTIONALLY FOLLOWED BY A LINE NUMBER.
IF A NUMBER IS PRESENT, LISTING WILL START AT THE FIRST LINE
WITH A NUMBER GREATER THAN OR EQUAL TO THE NUMBER GIVEN.  IF
NO LINE NUMBER IS GIVEN, LISTING WILL START AT THE LOWEST
NUMBERED LINE IN THE BUFFER.  LISTING MAY BE STOPPED AT
ANY TIME BY TYPING A CONTROL C (^C).  THIS COMMAND MAY
APPEAR IN A PROGRAM, BUT WILL RETURN TO COMMAND MODE
WHEN DONE.
FORMAT:         L (NUMBER) (CR)


        APPENDIX J
        ----------
        STRINGS

        STRING INPUT
STRINGS MAY BE INPUT FROM THE TERMINAL BY A STATEMENT OF THIS
FORM:
        )[(EXPR)]

WHEN CASUAL FINDS THIS TYPE OF STATEMENT, EXECUTION WILL STOP
AND THE STRING INPUT PROMPT IS PRINTED ON THE TERMINAL.
THE PROMPT IS A MINUS SIGN AND SPACE ("- ").  THE USER NOW
ENTERS THE STRING TERMINATING HIS INPUT WITH A CARRIAGE
RETURN.  YOU MAY CORRECT TYPING ERRORS WITH THE RUB-OUT KEY
AS LONG AS YOU DO NOT RUBOUT MORE CHARACTERS THAN
YOU HAVE TYPED.  CONTROL U (^U) SHOULD NOT BE USED TO
DELETE THE LINE BEING TYPED.  ANY PRINTING CHARACTERS ARE
LEGAL IN STRING INPUT.  WHEN THE INPUT IS COMPLETE, CASUAL WILL
PLACE THE STRING IN MEMORY AT AN ADDRESS COMPUTED BY
ADDING THE VALUE OF (EXPR) TO THE SINGLE BYTE ARRAY BASE.
THE STRING WILL BE FOLLOWED BY THREE NULLS (000) IN MEMORY.
SINCE YOU MAY NOT INPUT MORE THAN 72 CHARACTERS, THE LONGEST
STRING POSSIBLE IS 75 CHARACTERS.

        STRING OUTPUT
STRINGS MAY BE OUTPUT USING THE PRINT STATEMENT.  A
STATEMENT LIKE THIS WILL PRINT A STRING WHICH HAS BEEN ENTERED
EARLIER.

        ?)[(EXPR)]

CHARACTERS WILL BE PRINTED STARTING AT A MEMORY ADDRESS
COMPUTED AS IN STRING INPUT, AND CONTINUING UNTIL A NULL IS
FOUND.

INDIVIDUAL CHARACTERS OF A STRING MAY ACCESSED BY USING THE
SINGLE-BYTE ARRAY.  SEE SAMPLE PROGRAM BELOW.  SEE ALSO
APPENDIX O.

        10 '=2048:)[0]
        20 ?)[0];?)[2]
        30 ?'[0];'[2]
        .R
        - TEST
        TEST
        ST
         84  83

        APPENDIX K
        ----------
        SAMPLE PROGRAMS

        THIS IS THE SHORTEST NON-TRIVIAL PROGRAM.  IT
IS A DESK CALCULATOR.  THE USER TYPES IN MATHEMATICAL
EXPRESSIONS, THE RESULT OF WHICH IS PRINTED.  THE "=" OPERATOR
IS VERY HANDY IN THIS MODE OF OPERATION.

        10 ??:R


        FOR GAMING, HERE IS A ONE LINE RANDOM NUMBER GENERATOR:

        65000 R=R*32771:R=R*((R)0)-(R(0))+1:.=$

CALL THIS AS A SUBROUTINE WITH THE SEED IN R THE FIRST
TIME.  THE RANDOM NUMBER IS RETURNED IN R.  RANGE
IS 1 - 32767.


        THIS PROGRAM PRINTS A "BARBER POLE" PATTERN.

        50 Y=32:X=64:Z=X
        60 N=0
        70 ?)N+X+Y-(N+X)/Z*Z;:N=N+1:.=.*(N#64)
        80 ?:X=X+1:.=60


        THIS IS A SILLY PROGRAM TO SHOW HOW STRINGS ARE
USED.

        10 ?/TYPE IN A SENTENCE/;:)[0]
        20 A=0
        30 A=A+1:.=.*('[A]#0)
        40 ?/HERE IT IS BACKWARDS/;
        50 A=A-1:?)'A;:.=.*(A#0)
```

```
        APPENDIX L
        ----------
        BINARY TAPE FORMAT

DATA BLOCK:

        377     START CHARACTER
                LOAD ADDRESS LOW
                LOAD ADDRESS HIGH
                NUMBER OF DATA BYTES IN BLOCK
                DATA BYTE(S)
                CHECKSUM, SUM OF ALL DATA BYTES (IGNORE CARRY)


END OF TAPE:

        277     EOT MARK

        OPTIONALLY FOLLOWED BY A TWO-BYTE START ADDRESS.  IF
THIS ADDRESS IS PRESENT, IT MUST BE PRECEDED BY A 100 PUNCH.


        THIS IS A DESCRIPTION OF THE DATA ON THE CASUAL
TAPE.

1.      BINARY LOADER IN BOOTSTRAP FORMAT.  LOADS AT
        ADDRESSES 007 000 - 007 301.
2.      BLANK TAPE.
3.      CASUAL I/O ROUTINES IN BINARY FORMAT, ADDRESSES
        000 040 - 000 047 AND 005 106 - 006 153.
4.      BLANK TAPE.
5.      THE I/O PATCH OVERLAY IN BINARY FORMAT, ADDRESSES
        000 100 - 001 124.
6.      BLANK TAPE.
7.      END OF TAPE MARK AND TRANSFER ADDRESS MARK.  ADDRESS
        OF I/O PATCH ROUTINE ENTRY 000 100.  THIS EXECUTES
        THE I/O PATCH OVERLAY.  WHEN DONE POKING, IT RETURNS
        TO THE BINARY LOADER TO CONTINUE READING CASUAL.
8.      BLANK TAPE.
9.      CASUAL ADDRESSES 000 100 - 005 105.
10.     BLANK TAPE.
11.     CASUAL ADDRESSES 006 154 - 007 346.
12.     BLANK TAPE.
13.     CASUAL ADDRESSES 000 050 - 000 077.
14.     BLANK TAPE.
15.     CASUAL ADDRESSES 000 000 - 000 037.
16.     BLANK TAPE.
17.     END OF TAPE MARK AND TRANSFER MARK.  TRANSFER ADDRESS
        IS 000 000, THIS EXECUTES CASUAL.

        APPENDIX M
        ----------

        CASUAL "BNF" SYNTAX DEFINITION

 I    MEANS "OR"

<CASUAL>        ::=     <CASUAL LINE> I <CASUAL LINE><CASUAL>
<CASUAL LINE>   ::=     <LINE NUMBER><CLINE>
<CLINE>         ::=     <STMT> I <STMT>:<CLINE>
<LINE NUMBER>   ::=     <NUMBER>
<NUMBER>        ::=     <DIGIT> I <DIGIT><NUMBER>       NOTE 1
<DIGIT>         ::=     1 I 2 I 3 I 4 I 5 I 6 I 7 I 8 I 9 I 0
<STMT>          ::=     <?STMT> I <=STMT> I <>STMT> I <COMMAND>
<?STMT>         ::=     ?<PLIST> I ?
<PLIST>         ::=     <PEXP> I <PEXP>, I <PEXP>;
<PEXP>          ::=     <EXPR> I <LITERAL> I <CHR$> I )<SUBS>
<LITERAL>       ::=     /<STRING>/
<STRING>        ::=     <CHR> I <CHR><STRING>
<CHR>           ::=     NOTE 3
<CHR$>          ::=     )<EXPR>
<EXPR>          ::=     <SUM> I <SUM>=<SUM> I <SUM><<SUM>
                        <SUM>><SUM> I <SUM>#<SUM>
<SUM>           ::=     <TERM> I <TERM>+<SUM> I <TERM>-<SUM>
<TERM>          ::=     <FACT> I <FACT>*<TERM> I <FACT>/<TERM>
<FACT>          ::=     -<FACT> I <NUMBER> I <VAR> I . I ? I $
                        % I ! I & I ↑ I ← I @ I \ I +<FACT>
                        (<EXPR>)
<VAR>           ::=     <SVAR> I <AVAR>
<SVAR>          ::=     <LETTER>
<LETTER>        ::=     A I B I C I D I E I F I G I H I I I J
                        K I L I M I N I O I P I Q I R I S I T
                        U I V I W I X I W I Y I Z
<AVAR>          ::=     "<SUBS> I '<SUBS>
<SUBS>          ::=     <SEXP> I [<SEXP> I <SEXP>] I [<SEXP>]
<SEXP>          ::=     NOTE 4
<>STMT>         ::=     )<SUBS>
<=STMT>         ::=     <LEFT PART>=<EXPR>
<LEFT PART>     ::=     <VAR> I . I $ I ! I & I ← I @ I ↑
<COMMAND>       ::=     R I R<EXPR> I L I L<NUMBER> I O I N
                        T I T<LETTER> I S I S<LETTER>


NOTE 1:         UP TO FIVE DIGITS IN THE RANGE
                0 - 65535.
NOTE 3:         ANY PRINTING ASCII CHARACTER.  040 - 176
                (OCTAL) INCLUSIVE.
NOTE 4:         <SEXP> IS IDENTICAL TO <EXPR> EXCEPT
                THAT <AVAR> IS NOT A LEGAL ELEMENT.
```

PRAISE FOR ADVANCED MICROCOMPUTER PRODUCTS & JAMES ELECTRONICS

Dear Dr., Dec. 28, 1976

Enclosed is a copy of the letter Advanced Microcomputer Products sent me. I returned the AY5-8500 as requested and received the New chip Dec. 10. Installed in its socket it worked right the first time power was applied.

I would recommend James to your readers; I sent them an order on a Monday for $40.00 and received it Saturday the same week.

Thank you for the help; your magazine is the only one I read from cover to cover.

Stuart R. Fallgatter
7910 Rio Vista Dr.
Goleta, CA 93017
Nov. 22, 1976

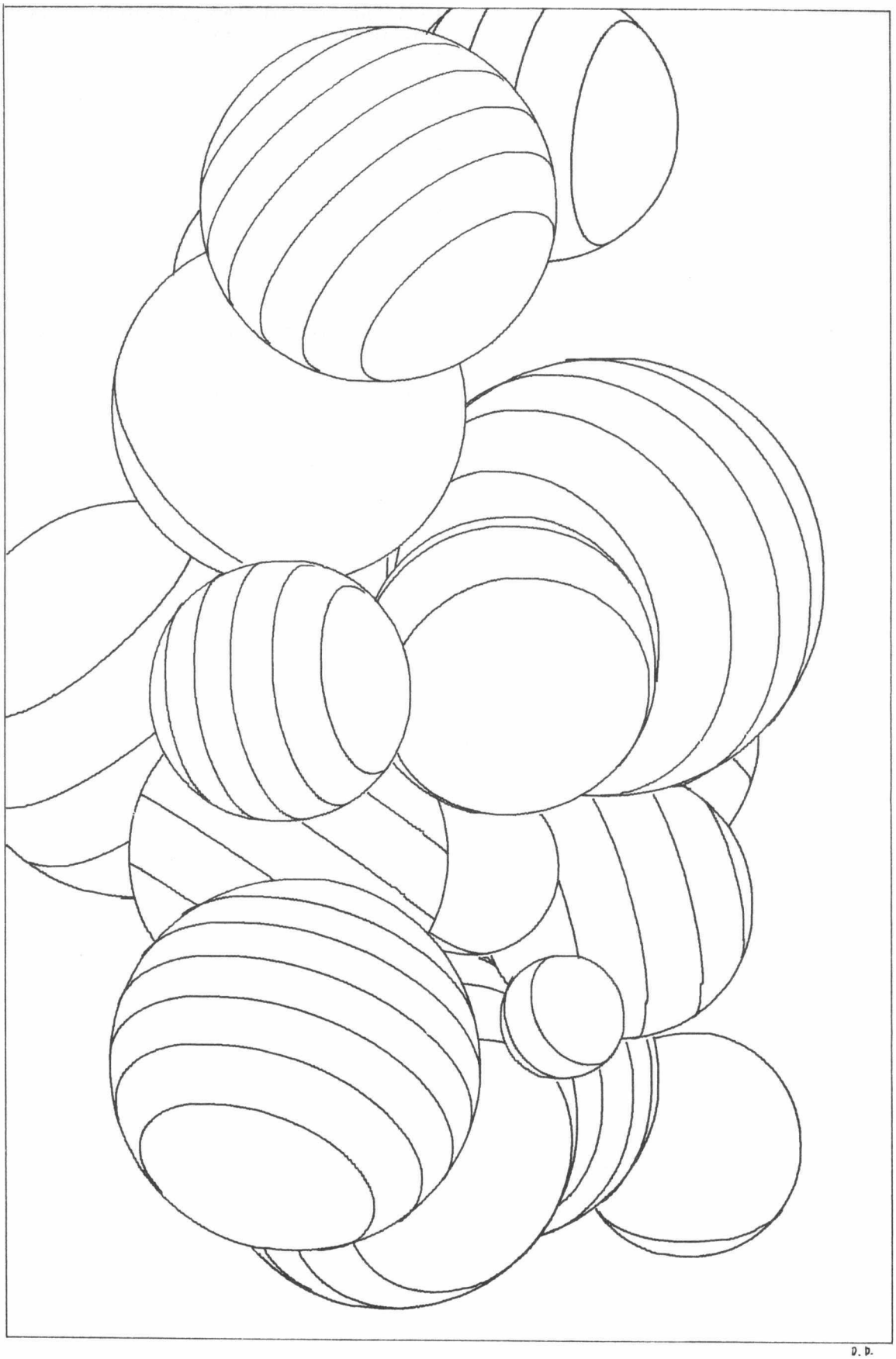
[From Advanced Microcomputer Products to Stuart Fallgatter]

In reference to your letter to Mr. Dobb's, we wish to advise that if you have a malfunctioned AY-8500 Game Chip that we shipped you, please return it to us and we will be more than happy to send you a replacement. If we may be of further assistance to you in the future, please do not hestiate to call upon us.

Sincerely,

Kimm Cornett
Advanced Microcomputer Products
P.O. Box 17329
Irvine, CA 92713
(714) 968-3655

```
        APPENDIX N
        ----------
        DECIMAL "POKE" ADDRESSES

        THE ADDRESSES BELOW MAY BE USED TO MODIFY CASUAL
AFTER THE INITIALIZATION DIALOG HAS BEEN COMPLETED.
        TO USE THESE, SET THE SINGLE-BYTE ARRAY BASE EQUAL
TO THE NUMBER ON THE LEFT.  THEN TYPE "[0]=<EXPR>", WHERE
<EXPR> IS THE NUMBER YOU WANT TO INSERT.

1279    THIS IS THE ADDRESS OF THE ADDRESS WHERE CASUAL
WILL PLACE IT'S STACK POINTER+1.  POKE HERE TO ALLOCATE MORE
OR LESS MEMORY TO CASUAL AFTER INITIALIZATION.  <EXPR>
SHOULD NOT EVALUATE TO AN NUMBER LESS THAN 1700.

1408    THIS IS THE ADDRESS OF THE ADDRESS AT WHICH CASUAL
WILL EXPECT TO FIND THE OPERATING SYSTEM.  REMEMBER:  DECIMAL.


        TO SET THE ADDRESS OF THE MACHINE LANGUAGE ROUTINE
CALLED BY "@" UNDER PROGRAM CONTROL:

        "=3:"="[0]:"[0]= DECIMAL ROUTINE ADDRESS

        APPENDIX O
        ----------
        ARRAY USE

        BOTH THE SINGLE- AND DOUBLE-BYTE ARRAYS OF CASUAL WORK
ON THE SAME PRINCIPLE:  FIRST AN AREA OF RAM IS SET
ASIDE FOR ARRAY STORAGE.  THIS IS USUALLY DONE AT
INITIALIZATION TIME.  THEN THE ARRAY BASE ADDRESS IS SET
SOMEWHERE IN THIS RAM AREA.  WHEN A REFERENCE TO THE ARRAY
IS MADE, THE MEMORY ADDRESS TO USE IS COMPUTED BY ADDING
THE SUBSCRIPT TO THE BASE.  IN THE CASE OF THE DOUBLE
BYTE ARRAY, THE SUBSCRIPT IS DOUBLED FIRST.

        MANY ARRAYS MAY BE KEPT AT THE SAME
TIME BY CHANGING THE BASE ADDRESSES.  CARE MUST BE TAKEN
NOT TO SET THE BASE ADDRESS TO A VALUE LESS THAN
THE NUMBER TYPED IN RESPONSE TO "MEM SIZ?", OR YOU MAY
WIPE OUT ALL OR PART OF CASUAL.

        MULTI-DIMENSIONAL ARRAYS CAN BE SIMULATED LIKE
THIS.  TO SETUP A TWO-DIMENSIONAL ARRAY WITH
DIMENSIONS 4,5:

1.      SET THE BASE ADDRESS AT LEAST 20 BYTES LESS THAN
HIGHEST ADDRESS AVAILABLE FOR ARRAYS FOR THE SINGLE BYTE
ARRAY, 40 FOR THE DOUBLE (4*5*2).

2.      DEFINE THE USER-DEFINABLE FUNCTION LIKE THIS:
        ↑=J*4+I

3.      THE SUBSCRIPTS, I AND J, NOW RANGE FROM 0 - 3
AND 0 - 4, RESPECTIVELY.  REFERENCE THE ARRAY LIKE THIS:
        '[↑]


        THIS PRINCIPLE CAN BE EXPANDED TO AN ARRAY OF ANY
NUMBER OF DIMENSIONS.
```



© David Dameron 1979

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 13 **March 1977** **Volume 2, Issue 3**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

SOFTWARE

Extended Accuracy Arithmetic Routines, M. G. Dineley 15
8080 Disassembler Written in MITS 3.2 BASIC, Jef Raskin 25
M6800 Disassembler, Gordon Stallings 30
A SC/MP Subroutine Supervisor, Richard K. Peterson 36
Tiny Linkage Editor for "relocatable code" on 8080's and Z-80's, Dan M. Shaw 8
Algorithm for computing square roots, James A. Hinds 23

HARDWARE

The Selectric Typewriter as An Output Device, Rod Graves 41

SYSTEMS

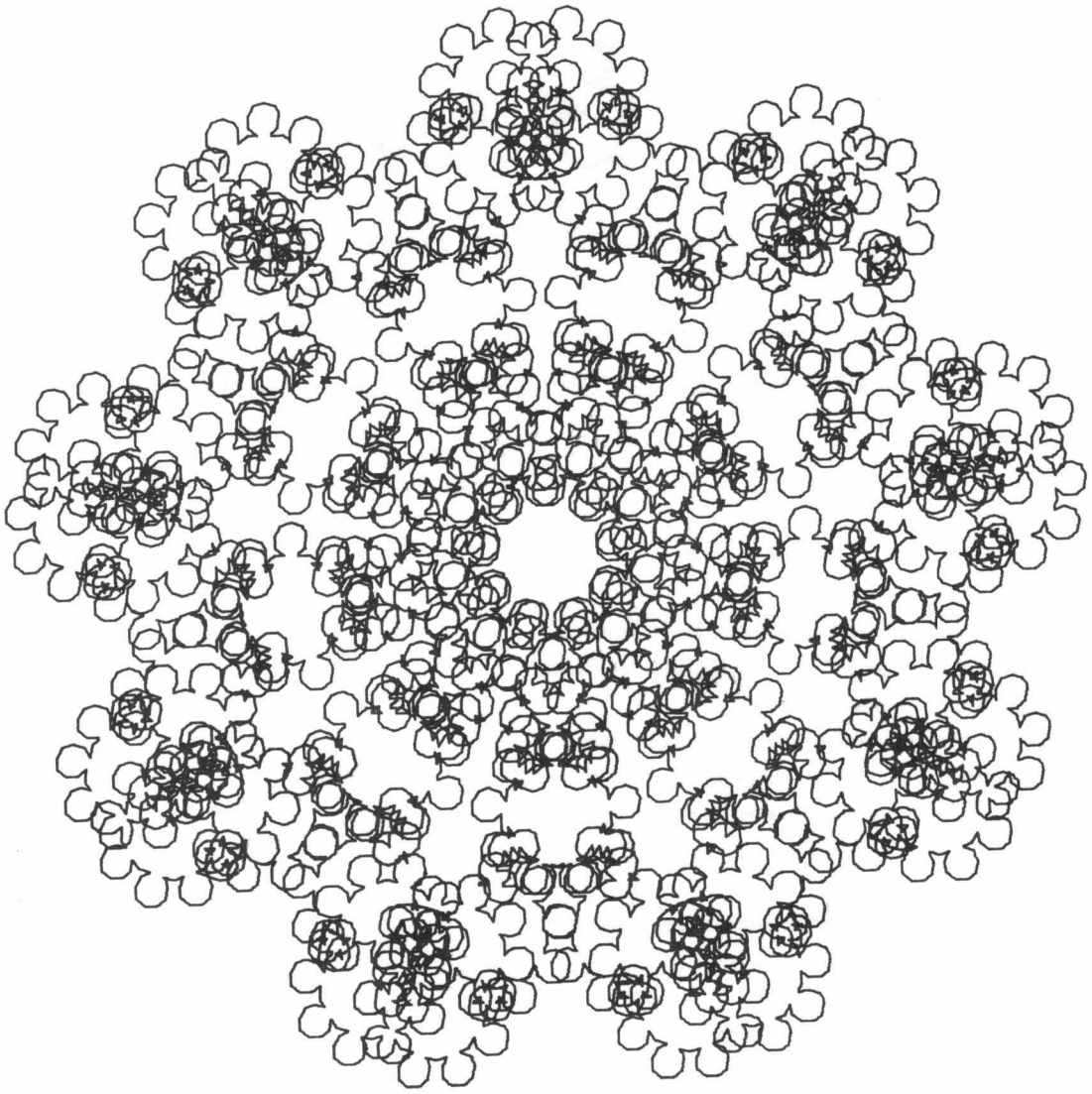
Graphics on the Poly-88, Harvey A. Cohen 12

MODEST PROPOSALS

Proposal for Standard Z-80 Code, Peter G. Formaniak 10
Proposed Tiny PILOT, Dennis Allison 34

AND THE USUAL PROFUSION OF MISCELLANY

notes, remarks, comments, fixes, announcements, new products, computer club notes



© David Dameron 1979

ACM FOUNDER COMMENTS ON COMPUTER KITS STORE

Dear Sir: March 11, 1977

I would like to report an interesting experience. After considering a number of possibilities for acquiring a microcomputer system, we chose Computer Kits (Pete Roberts) 1044 University Ave., Berkeley, CA 94710, and gave them an order on Dec. 20. The order consisted of an Altair 8800A computer with 32K of memory, an ADM 3A cathode ray and keyboard input-output, a cassette recorder and a connecting cable for a teletype, all to be assembled. The promise was 6 weeks delivery, assembled and guaranteed for six months. I was able to operate the system on Feb. 3 and start checking it out. But I found some bugs and some difficulties, including a procedure for removing the case, changing a toggle switch inside, and putting the case back on. On Feb. 10, I left my system unfinished, and went back to Massachusetts not happy.

When I came back on March 4, every part of the system was working excellently. Computer Kits had: arranged to put every operation on switches on the front panel; had modified the monitor and the editor to fit my system with the least amount of trouble; had arranged a cable for teletype input and output; so that a very simple and clear alteration of my teletype in Massachusetts could be accomplished; and had provided half a dozen other conveniences.

What they did was above and beyond the agreement. I am most pleased to recommend them for giving me an integrated and properly working system – and making this customer very happy.

Yours sincerely,
Edmund C. Berkeley
Editor, Computers and People
President, Berkeley Enterprises, Inc.
815 Washington St.
Newtonville, MA 02160

COMPUTER MAGAZINE STARTS MICROSYSTEMS SECTION

Computer Magazine, published by the IEEE Computer Society, has initiated a regular section devoted to small-scale computing, a term which includes microprocessors, minicomputers, hobby and personal computing. This section is a response to the rapidly changing nature of computing due to the impact of cheap memory, inexpensive processors, and a rapidly expanding application and user base.

Potential authors are solicited. Articles should be aimed at a broad technically based audience and should be tutorial or survey in nature, though specialized treatments of topics of general and timely interest are also desired. *Scientific American* is a good stylistic model.

Most any topic related to computers and computation might be suitable, but there is a particular interest in the economics, sociology, psychology and applications of personal and hobby computing. Authors interested in submitting manuscripts may contact the section editor for more information: Dennis Allison, 169 Spruce Ave., Menlo Park CA 94025, (415) 325-2962.

FAIRE CONSUMES EDITOR – DDJ LATE

What more can we say? *DDJ's* entripid editor involved himself as chairperson of the First West Coast Computer Faire and discovered it to be an infinite sink of time. And he was spread much too thin before he started. Xeroxing editors failed. The natural consequence is a late issue.

Because of Jim's overwhelming commitments to the Faire, the March and April issues of *DDJ* are being guest edited by Dennis Allison (who is only slightly less busy than Jim). Hopefully he will be able to maintain the past flavor and quality of *DDJ*.

PITTMAN OFFERS TINY BASIC EXPERIMENTATION KIT

Tom Pittman, the implementator of the $5/copy version of Tiny BASIC for the 6800 and 650X is now offering an experimenter's kit for those folks who want to modify and extend his Tiny BASIC. The kit includes an assembled source listing for the IL (Intermediate Language), and IL assembler written in Tiny BASIC, a detailed description of the virtual machine implemented by the IL, instructions for incorporating a new IL into the Tiny BASIC system, and finally some practical hints about debugging and extending the system. About 40 pages in all. The cost is $10 from Tom Pittman, Itty Bitty Computers, P.O. Box 23189, San Jose CA 95153, (408) 578-4944.

A SPHERE 6800 USER COMMENTS

Dear Jim: Nov. 29, 1976

What a super magazine you publish. The emphasis on software has been verv useful to me.

I thought the Sphere owners amongst your readers would be interested to know that the "Space Voyage" game offered by Technical Systems Consultants, Box 2574, W. Lafayette, IN 47906, can be easily modified to run in a sphere system. The game is an excellent version of Star Trek with very complete documentation (a complete listing with comments, hexidecimal dump and symbol table).

To rewrite the program for a Sphere system.:

1. Move all the temporary storage from locations 20-D4 to 40 F4. Of course change all the references to those sites within the program by adding 20 hex.
2. Move the program origin from 100_{16} to 200_{16} by adding 100_{16} to all program addresses and references.
3. Move the jump table and random number generator to the high end of this program.
4. Write the very simple routines to substitute sphere PDS routines for those in Mikbug. The properties of the Mikbug routines are given with the TSC literature.

By the way, the STRINGOUT described in *Dr. Dobbs* Vol. I, Number 8, Page 33 (1976) but rewritten for 6800 works fine to substitute for the Mikbug PDATA1 (print string). However, I recommend using the ASCII code 04 (message terminated) instead of a null to terminate the string.

This rewrite of the game is quite easy since one can keep all the flags on page zero and hence need not alter any of the commands or relative jumps. I was able to make these changes and track down those I missed on the first pass in four evenings.

I am quite pleased with my Sphere System (20K, built in monitor, keyboard, cassette board, SWTP printer and Oliver tape reader). The system has been up continuously and the Sphere System is no exception. The weak points I have experienced are:

1. Negligible documentation
2. No current user group
3. TV display wavers
4. Mounting hardware is inferior
5. Connecting cables instead of mother-board. Ugh!
6. Not S100 bus compatible
7. No advertising!

. The strong points I've liked are:

1. Excellent repair service (also fairly fast!)
2. The system works reliably. This is a hard point to ignore. (Two months now.)
3. Some independent hardware support (Oliver, SWTP)
4. Some independent software support (TSC, Data Center, 136 N. Maryland Ave., Glendale, CA 91206, Pittman's BASIC)
5. A complete integrated system – not so crucial with other systems S100 compatible.

Yours,

Marshall Hall Edgell
9901 Broad Street
Bethesda, MD 20014

P.T. INTEGER FIX FIX

Dear Jim,

Just a short note and an apology to anyone who tried the "integer fix" to Processor Tech BASIC 5 (*DDJ*, Vol. 1, No. 10).

Here is the corrected version which should clear up any confusion:

```
        XRA A          Existing program
        MVI D,5
AINT3   STAX B
        DCX B          Insert after addr 0BB0
        DCR D
        JNZ AINT3
        STAX B         Existing program
```

Adolph P. Stumpf
5639-A Ute
Glendale, AZ 85307

A FLOATING POINT CHIP QUERY

Jim,

What are the chances of a manufacturer making a microprocessor compatible floating point chip? Pocket calculator chips are available but inelegant to interface with.

Such a chip, of course, should have arithmetic, logarithmic and trigonometric functions. Nice if it had a couple of constants, too (PI,e).

Warren Leach

PERSONAL COMPUTERS & NUTRITION PLANNING ? . . SOMEDAY

Gentleones: Feb. 28, 1977

A few weeks ago on "All things considered," National Public Radio's news show, I heard David Ahl, and various others discussing uses of home computers, among which was its use in nutrition planning and food inventory. Since you do serve as an interchange of much personal level software and software ideas, I was hoping that you could either put me in contact with people who might have such software, tell me where such software might be found, or just publicize my interest in it to perhaps attract other people to work on this field. Thank you very much for your help.

Yours truly,
Reba Silterr
401 S. Coler Avenue
Urbana, IL 61801

[It's a good idea. I think it will happen sooner or later. I know of no one currently working on it. Ask Dave. – Jim W.]

SPHERE BOOTSTRAP FOR SWTPC BASIC

Dear Jim:

The number of SPHERE users among computer hobbyists is very small. SPHERE users seeking BASIC to run on their system may be interested to know that I have written a bootstrap loader which will read and patch ready for execution for SWTPC 8K BASIC Version 1. Since there aren't enough SPHERE users to warrent the space in *DDJ* I will make available documentation and loader on cassette for anyone interested. Send $10 to cover my cost of tape, documentation, postage and handling.

Sincerely,
Henry L. Kee
42-24 Colden Street
Flushing, New York 11355

JEF'S ADDENDA TO NOV/DEC *DDJ*

Jef Raskin January 11, 1977

As happens from time to time even in the best of publications, a few items got garbled between my typewriter and your publication. For example the fourth paragraph originally explained that the switches on the IMSAI are spaced so close together that it is all too easy to hit two at once. The PDP-ll has similar switches with this small difference: the tips are tapered so that there is a small space between them, making hitting two at once less likely. A few gliches (editing late at night again, eh, Mr. Warren?) **[You mean there's another way? – JCW]** made it come out funny in the magazine.

The sentence that reads "More screws than an X-rated movie, and then there are wires screwed onto the board." should have had the word "screwed" replaced with the word "soldered". I wonder what I could have been thinking about to make an error like that. A type that slipped through is at the end of the very next paragraph: "base" instead of "case".

Here's a coincidence. As I was typing this letter I got a call from Polymorphic systems. They tell me that a few of my complaints have been corrected. Especially with regard to the manual. As usual, I'll believe it when I see it. Tune in again next issue. Similarly, the people at IMS admitted (as described in the article under discussion) that I got the cold shoulder when trying to communicate some of the errors in their manuals, and told me that they would be good to me and their other customers in the future. The next time I test one of their products I will also test their response, and use a disguised voice so they won't know it is the fearless equipment reviewer (heh, heh).

Another error (oops) is that in the paragraph on the CROMEMCO Bytesaver, the reference to the 2708-8 PROM came out as a reference to a 2707. At the end of the paragraph the Editor put in the comment that having no sense switches is a possible impediment to man-machine interaction. I claim that I can duplicate and even better sense switch interaction with the keyboard that inevitably goes with a front-panel-less computer. Room for your comment here. . . .

On the differences between the Poly and the Altair and the IMSAI busses. Poly and IMS have put out documents detailing the differences. Bravo folks. I have been both documents and they are avilable from the manufacturers.

Lastly, thanks for the supportive letters and calls about my articles! And also thanks to the manufacturers who, like Mr. Tarbell, respond to constructive criticism from their customers and improve their product and/or documentation.

ECD's MICROMIND yM-65 QUERY

I am looking for an objective opinion on this product and its manufacturer. Can anyone help me?
Rick Fence
811 E. 58th Avenue
Vancouver, B.C.
CANADA V5X 1W6

We also would be interested in your comments. – Editor

COMMUNITY INPUT FOR DESIGN OF A *GOOD* ASSEMBLER

Dear Mr. Warren: February 18, 1977

I read some of the back issues of *Dr. Dobbs Journal* last night and I found a letter from Jim Day dated 6/27/76 calling for a good assembler for the 8080. I too feel that any serious work with a computer is hampered by a crummy assembler and, worse, the lack of a standard set of mnemonics, pseudo-ops and coding conventions. We all know that manufacturer A wants colons after his symbols, mfr B doesn't and mfr C wants comments to start with asterisks etc. This leads to a certain confusion which annoys experienced computniks and frustrated many inexperienced ones.

So, before the contract goes out for a good assembler, let's all conduct a poll as to what we want to see. I will be glad to serve as the collector. Keep in mind that long symbols mean a lot of memory, short ones are frequently too distorted to be meaningful.

Let me know your ideas. I have a rough idea for an assembler which incorporates many of the features of Jim Day's want list. Write a note, preferably in list form (as Jim Day did it) to: A. L. Bender, M.D., 400 Old Hook Road, Westwood, NJ 07675. I promise to tabulate the stuff and write it up for approval. (Please don't send material to my home, an address ambiguity results in loss of mail.)

Very truly yours,
A. L. Bender

O'course, Don Knuth sez we should never have a really GOOD assembler. Otherwise, people will keep using assemblers to do major software development, instead of using good, efficient, high level language compilers . . . like they oughta. (Hope I haven't butchered Master Knuth's views too badly.) – JCW

NIFTY DISASSEMBLER FOR SWTP 6800 WITH TVT-II

Dr. Dobbs: March 27, 1977

Thoroughly enjoy your publication. It definitely provides a media for information exchange among us hobbyists.

That is why I would like to inform your readers of a nifty piece of software I recently developed for a SWTP 6800 with TVT-II. It's a very compact dis-assembler (300 hex in size) that generates output similar to that below:

```
A020 A1 00     CMPA $00,X
A022 26 53     BNE  $A077
A024 BV A004   CPX  $A004
A027 27 03     BEQ  $A02B
A029 08        INX
```

Note that relative addresses are resolved to absolute ones for ease of reference. The program is currently assembled at 1D00–1EDF with opcode tables at 1ED0–1FFF.

For a SASE I will send anyone a more complete description including actual sample output and information on how to obtain a source listing or AC-30 compatible cassette of source and object. For $4.00 I will return a copy of the object program in hex dump format with instructions for memory relocation and other useful changes.

I will have as an extension to this program the capability of defining symbols that will replace the $hhhh addresses in the output. This extension will automatically be sent to all those who are sent the original program.

Robert Dembinski 12 Richard Rd.
Medway, MA 02053

Tiny linkage editor for "relocatable code" on 8080's and Z-80's

Aloha, *February 7, 1977*

I got so tired of reassembling subroutines so that I could move them that I figured out a way to write relocatable code. It works a lot like the relative branch instructions on the Z-80. I also have a "tiny linkage editor" which lets me call subroutines by name instead of by address.

I have a Z-80 system from the Digital Group, but I've written everything in 8080 for compatibility. I use a one-of-a-kind cross assembler, so I've redone source listings in more readable mnemonics. It's all double-checked.

Dr. Dobb's is fantastic. Keep up the good work.

Sincerely,

Dan M. Shaw
2182 Round Top Drive
Honolulu, HI 96822

Have you ever tried to move a subroutine from one part of memory to another? It's a real hassle. You have to find all the branch instructions and all the loads and stores that refer to specific locations and then reassemble the code so it will reflect the changed locations. Even if you have a text editor and an assembler, it's a hassle.

One way out of this mess is to write "relocatable code." What does that mean? To put it simply, relocatable code contains no absolute addresses. The Z-80 has branch instructions which use relative addressing. For example, the instruction 28 DE is an unconditional branch with a relative displacement of DE. The displacement (which in this case is negative) is added to the program counter (which always points to the *next* instruction.) The resulting address is the address that is branched to. This instruction would execute just as well any place in memory, because it is relative, not absolute. Unfortunately, the Z-80 does not have relative load and store instructions. If it did, we could write subroutines that would execute anywhere in memory without modification.

On my Digital Group Z-80 system, I use the "EF" instruction (restart to location 0040) to generate three-byte relative instructions. This is how it is done. The first byte is the restart. The second byte is the opcode of the instruction that I would have used. The third byte is the displacement from the next instruction to the actual address.

Figure 1 shows the code which does the work. The restart instruction is a one-byte subroutine call, so it puts a return address on the stack. This address points to the opcode and the displacement. The displacement is used to calculate an address, which is combined with the opcode to generate an instruction stored at OP. After all the registers and flags are restored, this instruction is executed and control is transferred back to the calling program. By changing the branch at MODE from SLOW to FAST, you may add a section of code that copies the generated instruction back into the calling program (the generated instruction takes three bytes, so it just fits). The next time through, it will be executed directly. Of course once this happens, the instruction is no longer relocatable.

The restart instruction is really a subroutine call, but I prefer to think of it as a new opcode, a new machine instruction which I have added to my system. With it, I can write three-byte relative instructions that give me relocatable code. There's another neat trick: the instruction EF 21 FD (21 is the opcode for "load HL immediate" and ED is the displacement which points to the byte where EF is stored) will load the HL register with the instruction's own address. This enables a subroutine to find out where it is.

Now that I can write relocatable subroutines, I still need a way to call one subroutine from another without knowing its absolute address. You might say I've solved the internal problems but not the external ones. On dinosaurs, they have things called linkage editors to handle the linkage between subroutines. I am proposing a "tiny linkage editor."

On my system, I use the E7 instruction (restart to location 0038) to implement a "call by name" instruction. Figure 2 shows how it works. The first byte is the E7, and the next two bytes are the name or identification of the subroutine. A subroutine can be named anything you want. Each subroutine in the system has a six-byte header containing the length of the subroutine (my cassette read and write routines use this information), the name of the subroutine, and space for the address of the next subroutine. The address pointers are set when the subroutines are loaded, and the last subroutine in the list has zeroes in the address bytes. The restart instruction follows the pointers from one subroutine to the next, looking for the named subroutine. When it finds it, it restores the registers and calls it. Figure 3 shows the code which accomplishes this. Once again, a fast mode is provided, which changes the "call by name" instruction in the calling program to an ordinary call with an absolute address.

I wrote a fancy "Life" program using this scheme. It really encourages modular programming and saves a lot of time. I broke the problem down into a collection of subroutines. I wrote one subroutine at a time, starting at the "bottom" with subroutines that didn't call anything, then the subroutines that called only routines that I'd already written. Each subroutine was relocatable, and after I debugged it, I stored it on tape. The whole thing took about a dozen routines. When I had the program working like I wanted it (a little bit slow, I will admit), I patched the SLOW modes to FAST and ran the program to get all the relative and "call by name" instructions fixed. Now my program runs at normal speed. As an added bonus, some of the subroutines will come in handy for other applications.

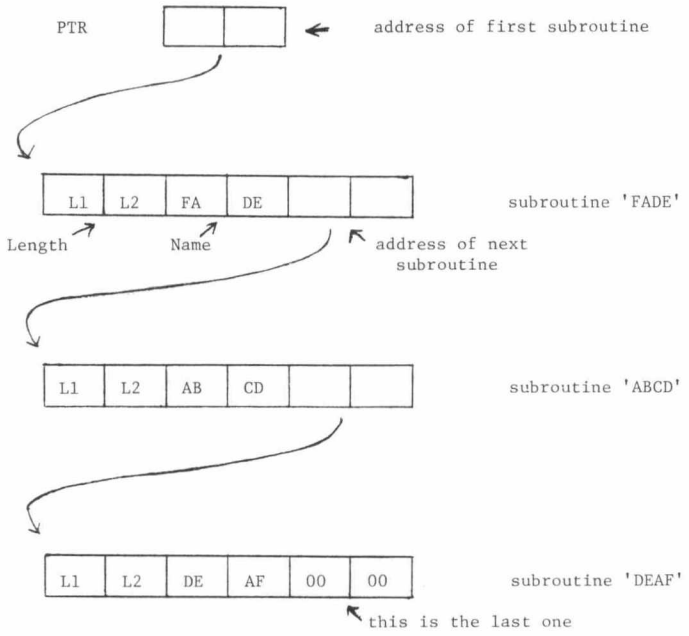
If you have a floppy, you might try this idea: change the code so that the "call by name" instruction looks through memory for the desired subroutine, and if it's not there, fetches it in automatically from the disk. You could also use the same idea to fetch blocks of data by name.

I/O SELECTRIC INTERFACE (*NOT* FOR AN OFFICE SELECTRIC)

Interfacing a Selectric to your microcomputer? The Center for the Study of the Future, a religious information networking organization, has just released a Selectric interface card for any of the Dura of Itel Selectric terminals with 24 volt solenoids (Dura 1021, 1040, etc.) as well as for the Tycom adaptor. Input is RS-232 seven bit ASCII at 110 or 134 baud.

The card is available in a kit form without power supplies for only $325, but is also available with power supplies as well as assembled. Some Dura units are also available as completed systems. Send SASE for brochure – Center for the Study of the Future, 4110 N.E. Alameda, Portland, OR 97212.

E7 AB CD CALL 'ABCD'




```
                *RELOCATABLE CODE GENERATOR
                *
                *EXAMPLE:  EF C3 60  IS A RELATIVE JUMP
                *TO NEXT + 60
                *
                *PATCH LOC 0040 TO BRANCH TO 0600
                        ORG  0600H
0600  E3        REL     XCH  HL,(SP)    SAVE HL, GET RETURN
0601  F5                PUSH AF         SAVE REGISTERS
0602  C5                PUSH BC
0603  E5                PUSH HL
0604  7E                LD   A,(HL)     GET OP CODE
0605  32 30 06          STO  A,OP       STORE
0608  23                INC  HL         POINT TO DISP
0609  06 FF             LDI  B,0FFH     FIRST BYTE FOR NEG DISP
060B  7E                LD   A,(HL)     GET DISPLACEMENT
060C  4F                LD   C,A        MOVE TO C
060D  B7                OR   A          TEST A
060E  FA 12 06          JN   SKP        IF NEGATIVE
0611  04                INC  B          FIRST BYTE FOR POS DISP
0612  23        SKP     INC  HL         POINT TO RETURN
0613  22 34 06          STO   HL ,RET   STORE FOR RETURN JUMP
0616  09                ADD  HL,BC      CALCULATE ADDRESS
0617  22 31 06          STO   HL ,ADDR  STORE
061A  E1                POP  HL
061B  2B                DEC  HL         POINT TO OP CODE
061C  C3 2D 06  MODE    JMP  SLOW       JUMP IF SLOW MODE
061F  3A 30 06  FAST    LD   A,OP       GET OP CODE
0622  77                STO  A,(HL)     STORE IN CALLING PROG
0623  23                INC  HL         NEXT BYTE
0624  3A 31 06          LD   A,ADDR     GET ADDR
0627  77                STO  A,(HL)     STORE
0628  23                INC  HL         NEXT BYTE
0629  3A 32 06          LD   A,ADDR2    GET ADDR2
062C  77                STO  A,(HL)     STORE
062D  C1        SLOW    POP  BC         RESTORE REGISTERS
062E  F1                POP  AF
062F  E1                POP  HL
0630  00        OP      DEF  0          GENERATED OP CODE
0631  00        ADDR    DEF  0          ADDRESS
0632  00        ADDR2   DEF  0
0633  C3        JMP     DEF  0C3H       JUMP FOR RETURN
0634  00        RET     DEF  0          RETURN ADDRESS
0635  00                DEF  0
```

```
                *CALL BY NAME
                *
                *EXAMPLE:  E7 AB CD  CALL SUBR ABCD BY NAME
                *
                *PATCH LOC 0038 TO BRANCH TO 0636
                        ORG  0636H
0636  E3        CBN     XCH  HL,(SP)    SAVE HL, GET RET
0637  D5                PUSH DE         SAVE REGISTERS
0638  C5                PUSH BC
0639  F5                PUSH AF
063A  E5                PUSH HL
063B  56                LD   D,(HL)     GET IDENT
063C  23                INC  HL
063D  5E                LD   E,(HL)
063E  2A 81 06          LD   HL,PTR     ADDR OF 1ST BLOCK
0641  23        CHKID   INC  HL         GET IDENT OF BLOCK
0642  23                INC  HL         INTO BC REG
0643  46                LD   B,(HL)
0644  23                INC  HL
0645  4E                LD   C,(HL)
0646  78                LD   A,B        COMPARE IDENTS
0647  BA                CP   D
0648  C2 50 06          JNZ  NXTPT
064B  79                LD   A,C
064C  BB                CP   E
064D  CA 5F 06          JZ   FOUND      IF IDENTS SAME
0650  23        NXTPT   INC  HL         GET POINTER
0651  4E                LD   C,(HL)
0652  23                INC  HL
0653  46                LD   B,(HL)
0654  78                LD   A,B        TEST FOR ZERO
0655  B1                OR   C
0656  CA 5E 06          JZ   EOL        END OF LIST
0659  60                LD   H,B        MOVE BC TO HL
065A  69                LD   L,C
065B  C3 41 06          JMP  CHKID      TEST NEXT BLOCK
065E  F7        EOL     R30             *SNAP SYSTEM
065F  23        FOUND   INC  HL         FOUND IT
0660  23                INC  HL
0661  23                INC  HL         EFFECTIVE ADDRESS
0662  22 7F 06          STO  HL,JCAL    STORE IN CALL INSTR
0665  E1                POP  HL         RETURN ADDRESS
0666  C3 69 06  MODE    JMP  SLOW       MODE SWITCH
0669  2B        FAST    DEC  HL         GENERATE CALL INSTR
066A  3E CD             LDI  A,0CDH     AND STORE IN
066C  77                STO  A,(HL)     CALLING PROGRAM
066D  23                INC  HL
066E  3A 7F 06          LD   A,JCAL
0671  77                STO  A,(HL)
0672  23                INC  HL
0673  3A 80 06          LD   A,JCAL2
0676  77                STO  A,(HL)
0677  2B                DEC  HL
0678  23        SLOW    INC  HL
0679  23                INC  HL         GET RETURN ADDRESS
067A  F1                POP  AF         RESTORE REGISTERS
067B  C1                POP  BC
067C  D1                POP  DE
067D  E3                XCH  HL,(SP)    SET RETURN
067E  C3        JUMP    DEF  0C3H
067F  00        JCAL    DEF  00
0680  00        JCAL2   DEF  00
0681  00 00      PTR     DEF  0,0        POINTER TO FIRST SUBR
```

COMPUTER CENTER, INC. IS APPOINTED DISTRIBUTOR FOR THE DTC MICRO FILE IN NORTHERN CALIFORNIA

Computer Center, Inc. owners of the BYTE SHOP of San Francisco, 321 Pacific Avenue, San Francisco (415) 421-8686, and the Berkeley BYTE SHOP, 1514 University Avenue, Berkeley, (415) 845-6366 have been appointed as the exclusive distributors for the DTC Micro File in the San Francisco – Oakland Bay Area. The DTC Micro File is a sophisticated word and data processing system based on the 8080 microprocessor.

For additional information: J. Benbow Bullock, Vice President, Computer Center, Inc. 321 Pacific Avenue, San Francisco, CA (415) 421-8686.

PROPOSAL FOR STANDARD Z-80 CODE

Dear Sir: February 14, 1977

MOSTEK has developed a standard for Z-80 object code which is upward compatible with the current Intel hex format. The format handles relocatable and linkable programs. We are suggesting that this format be adopted for Z-80 software products to produce some standardization in the industry. This information is not proprietary and is not copyrighted. You may use the information at your discretion.

Sincerely,

Peter G. Formaniak
Microprocessor Applications Group
MOSTEK, P.O. Box 169, 1215 West Crosby Road,
Carrollton, Texas 75006

3.0 Definitions

An "object module" is the output from an assembler. One or more object modules can be placed into memory by a loader. The output from the loader is called a "load module".

A "relocatable" program is one which has extra information in the object module. The extra information allows the loader to place the program anywhere in memory. A non-relocatable program can be loaded only in one place in memory in order to work properly.

A "position independent" program can be placed anywhere in memory. It does not require relocating information in the object module.

An "internal" symbol is a symbol which is defined in a program and whose location is made known to all other programs. It is also called "public" (Intel), "defined", "global", or "common".

An "external" symbol is a symbol which is used in a program but which is not defined in the program. Such symbols must be defined as internal symbols in other programs.

A "linkable" program has extra information in the object module which defines internal and external symbols. The loader uses this information to connect, or link, external references to internal symbols.

4.0 MOSTEK Object Output Definition

Each record of an object module begins with a delimiter (colon or dollar sign) and ends with carriage return and line feed. A colon (:) is used for data records and end-of-file indicator for Intel compatibility. A dollar sign ($) is used for records containing relocation information and linking information. An Intel loader will ignore such information and allow loading of non-relocatable, non-linkable programs. All information is ASCII.

Each record is identified by a "type". The type appears in the 8th and 9th bytes of the record and can take the following values:

00 - data record
01 - end-of-file
02 - internal symbol
03 - external symbol
04 - relocation information

3.1 Data Record (type 00)

Byte 1	Colon (:) delimiter
2-3	Number of (binary) bytes of data in this record. The maximum is 32 binary bytes (64 ASCII bytes).
4-5	Most significant byte of start address of data.
6-7	Least significant byte of start address of data.
8-9	ASCII zeros. This is the "record type" for data.
10	data bytes
Last two bytes	Checksum of all bytes except the delimiter, carriage return, and line feed. The checksum is the negative of the binary sum of all bytes in the record.
CRLF	Carriage return, line feed

3.2 End-of-file Record (type 01)

Byte 1	Colon (:) delimiter
2-3	ASCII zero
4-5	Most significant byte of the transfer address of the program. This transfer address appears as an argument of the "END" statement of a program. Although we will not be using this facility, it may prove useful in higher level language programs. It is an Intel standard.
6-7	Least significant byte of the transfer address.
8-9	Record type 01.
CRLF	Carriage return, line feed

3.3 Internal Symbol (type 02)

Byte 1	Dollar sign ($) delimiter.
2-7	Up to 6 ASCII characters of the internal symbol name. The name is left justified blank filled.
8-9	Record type 02.
10-13	Address of the internal symbol, most significant byte first.
14-15	Binary checksum.
CRLF	Carriage return, line feed.

3.4 External Symbols (type 03)

Byte 1	Dollar sign ($) delimiter
2-7	Up to 6 ASCII characters of the external symbol name. The name is left justified, blank filled.
8-9	Record type 03.
10-13	Last address which uses the external symbol. This is the start of a link list which is described below. The most significant byte is first.
14-15	Binary checksum.
CRLF	Carriage return, line feed.

The SDB-80 assembler outputs the external symbol name and the last address in the program where the symbol is used. The data records which follow contain a link list pointing to all occurrences of that symbol. This is illustrated in Figure 1.

1. The external symbol record shows the label ('LAB') and the last location in the program which uses the symbol (212AH).
2. The object code at 212A has a pointer which shows where the previous reference to the external symbol occurred (200FH).
3. This backward reference list continues until a terminator ends the list. This terminator is $FFFF_H$.

This method is easy to generate and decode. It has the advantage of reducing the number of bytes of object code needed to define all external references in a program. Intel defines each external reference explicitly.

3.5 Relocating Record (type 04)

The addresses in the program which must be relocated are explicitly defined in these records. Up to 16 addresses (64 ASCII characters) may be defined in each record.

Byte 1	Dollar sign ($) delimiter.
2-3	Number of sets of 2 ASCII
4-7	ASCII zeros.
8-9	04.
10-	Addresses which must be relocated.
Last two bytes	Binary checksum.
CRLF	Carriage return, line feed.

4.0 Assembler Object Output Sequence

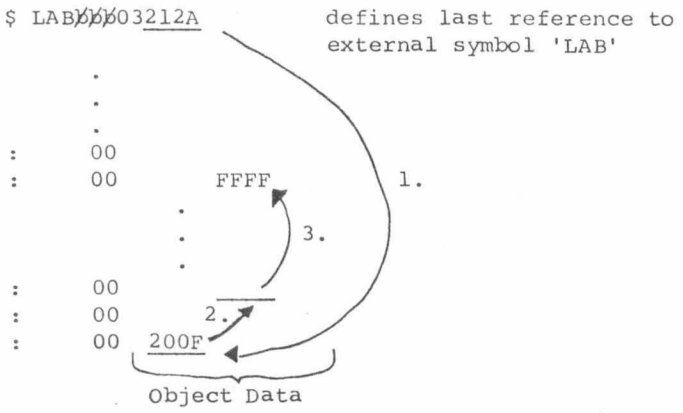
The SDB-80 Assembler will output object records in the following sequence:

1. Internal symbols (type 02) in alphabetical order.
2. External symbols (type 03) in alphabetical order.
3. Data records (type 00), interspersed with relocation records (type 04).
4. End-of-file record (type 01).

Internal symbol records will be output only if internal symbols are defined in the source program. The same is true of external symbol records. Relocation records always will be output by the assembler.

Figure 1

External Symbol Link List



HELP DESIGN A HEX CALCULATOR

Dear Jim: January 3, 1977

As one of several current projects we're working on a hex calculator for program debugging. Before "casting the design in concrete," we would like a little feedback from potential customers on our choice of operations, number of digits, etc. Tentative price for the 1601 complete and tested is $350 with a kit version priced somewhat less (we don't know how much less yet).

Thanks,

Ron Tipton

TDL Electronics
Route 7
Fayetteville, Arkansas
72701 (501) 643-2191

Model 1601 Hexadecimal Calculator — Preliminary Specifications
December 1976

GENERAL

The 1601 is designed around an RCA 1802 microprocessor and the operating program is contained in permanent read only memory (ROM). Preliminary performance specifications are listed on an attached sheet and your comments are invited.

KEYS

The keys are mechanical switches with a 3/16 inch keystroke. Keytops are 3/4 inch square and are permanently marked by being double-shot molded of ABS plastic.

DISPLAY

The display consists of ten characters plus overflow, decimal and negative indicators. The characters are 0.3 inch high easy-to-read 5x7 dot matrix type.

CABINET

The 1601 is housed in a type SEE1-2 cabinet made by The Enclosures Group — see attached drawing. The printed circuit card is 1/16 inch thick glass-epoxy (G-10) material with plated conductor pattern.

POWER

105 to 135 volts AC, 60 Hz, 10 watts.

PRICE

Tentatively priced at $350 each fully assembled and tested. The 1601 will also be available in kit form at a lower price.

LIMITED WARRANTY

One year parts and labor to original purchaser.

1. By our convention the most significant bit stores the sign: logical "1" for negative and logical "0" for positive. Thus the range of the calculator is +7FFFFFFF to −7FFFFFFF in hexadecimal.
2. Performs addition, subtraction and double precision multiplication on two 8 digit hex integers. When the multiply operation ("x" key) is performed the most significant 8 digits of the product are displayed. Pushing the key labeled "N" displays the least significant 8 digits of the multiplication result.
3. Chained operation is provided but nesting is not.
4. A separate 8-digit memory register is provided. Numbers are stored in memory by first pressing the "=" key and then pressing the "M" key.
5. Both clear "C" and clear entry "CE" clears only the number currently being entered.
6. A complement key "CM" is provided to calculate the two's complement of any hex integer within the range of the calculator. The result of performing "0 - n" is identical to the result of entering "n CM" where n is the hex integer to be negated.
7. A hex to decimal conversion key is provided to convert any hex integer of 8 digits or less to decimal. A ten digit display is provided for decimal numbers and a lower case "d" lights at the far left side of the display to indicate that the number is decimal. ($7FFFFFFF_{16} = 2{,}147{,}483{,}647_{10}$) The calculator clears its accumulator after displaying a decimal number so that chained calculations "mixing" decimal and hexadecimal numbers cannot occur.
8. An indicator light (LED) comes on whenever the content of the accumulator is negative (i.e., the leading hex character displayed is 8, 9, A, B, C, D, E or F). An overflow indicator "o" comes on whenever the result of a data entry or an arithmetic operation exceeds the calculator range. If a negative hex number is converted to decimal, the LED remains on (indicating negative); the display indicator flag shows "d" (for decimal) and the decimal magnitude of the number is displayed. (The two's complement of the negative hex value is converted to decimal.)

[Texas Instruments used to make a hex calculator but withdrew it because there was no market. Now everyone wants one! —Ed.]

GRAPHICS ON THE POLY–88

Harvey A. Cohen
OZNAKI Project, La Trobe University, Bundoora Victoria, Australia 3083

In the Nov/Dec 76 issue of *DDJ* Jef Raskin gave a careful review of the Poly–88 personal computer. However this review was rather hardware oriented and did not seek to evaluate the graphics capacity of the Poly–88 which is a bonus (i.e. free) feature of this system.

A Rose by any other name . . .

To start discussing graphics, it helps to distinguish between interactive graphics, dynamic graphics and data graphics (though in particular set-ups the distinction may be blurred). In interactive graphics what's on the screen can be varied substantially (including partial erasure) by an input from the user. Thus interactive graphics requires a facility for updating pictures and a convenient representation of the current state of the screen. In dynamic graphics substantial shapes move about the graphic screen to give a coarse grained cartoon effect. Speed is of the essence in dynamic graphics, and the capabilities of Poly for such performance requires a demonstration. (Read on!) Finally there is what I call data graphics, where a long string of data just gets output to the screen. Of course any device that calls itself a graphic system can perform data graphics.

Background to this evaluation

In the OXNAKI Project we've used the SWTP 6800 System (also a SC/MP and a DEC–10 as mousetrap variations) as a controller for our rudimentary robot ZONKY which is programmed by kids. (Educationists interested should write for my paper "The OXNAKI Robotics Language OZ" for a fuller description. The SWTP 6800 has also been used as the controller for another kids' (i.e. 6 to 12 year olds) programming system – WHAM. In WHAM a pseudo robot called a NAKI crawls about a TV screen, where the robot is represented by ^, V, >, or < depending on whether the NAKI is heading North, South, East or West. As the NAKI moves about a bounded area of the screen (the rest of the screen is used for program storage, etc.) he/she/it leaves a trail of asterisks, forming various patterns. Originally we used the South West Technical Product terminal kit CT1024 to generate the graphics, so that the largest NAKI territory possible was 13 by 13. Clearly the pictures kids can create using the CT1024 are extremely coarse grained, and the finer mesh of graphic characters on the Poly–88 would improve resolution. It is clear that for our purposes the Poly–88 would represent an improvement as far as interactive graphics was concerned. However we plan to implement a dynamics graphics system to extend the creative possibilities for older kids. Would Poly–88 be capable of dynamic graphics? Our answer, as you'll see is a guarded YES.

Interactive Graphics

A first question is whether the Poly–88 is suitable for interactive graphics. In Version 2.2 of the Poly monitor a keyboard stroke generated an interrupt, serviced by the PROM monitor. In other words a single keystroke would stop the program running, and the key depressed would be handled as a monitor command. However the latest Poly-88 monitor, version 4.0, is rather smarter, and a keyboard interrupt causes a jump to a "wormhole" location on CPU board RAM. On RESET the monitor writes into this wormhole a call to the monitor keyboard service routine, followed by a RET. However the user can plug into the wormhole the address of his own service routine. So the answer to the first question is YES.

Data Graphics

On TV graphics screens the screen area is divided into a grid of squares which may be white (STUNG) on black. For instance, the Poly-88 has a 48 by 128 grid, the SWTP graphics system has a 96 by 64 grid so that for both there are 6144 screen squares that may be stung to produce a coarse grained picture. If one loads into the computer a long list of squares to be STUNG to produce a picture, one is indulging in what I call data graphics. Of course even to output "data" to a screen (whether peripheral or on bus) requires some gentle massaging by the processor, but the distinction I'm making is pretty sharp. A prime example of data graphics is given by the program included in an article on the SWTP Graphics System in *Byte*, December, 1976. (In that program the stung points required to outline the USNS Enterprise are listed as data.)

STINGING Poly

The Poly–88 displays 16 lines of 64 alpha-numeric or graphic characters on a screen. Each screen location corresponds to a unique location in memory starting at 8800H (Monitor version 2.2) or F800H (Monitor version 4.0). If bit 7 of the memory byte is 1, the Poly treats the other 7 bits as ASCII and uses a slightly unusual character generating ROM for a conventional dot matrix display at the character position. If bit 7 of memory location is 0, then Poly displays a graphics character based on bits 5 to bits 0. For a graphics character, each character position is divided into 6 squares. Whether a particular square is stung or not depends on whether the corresponding bit is 0 (for sting) or 1 (for off).

bit 5	bit 2
bit 4	bit 1
bit 3	bit 0

TARGET SUBROUTINE

How can one sting a particular square in the 48 by 128 grid on the Poly screen? One needs a subroutine that will take the x, y coordinates of the square of interest, and then determine the screen address. As at that screen address some of the other 5 squares may be already STUNG, one must generate a sort of mask. All of which suggests quite an amount of processing. If one uses standard 7-fold loops for the division (by 3) algorithm then the amount of processing will become quite ridiculous. However the infinite series

1/4 + 1/16 + 1/64 + 1/256 + 1/1024 + . . .

whose sum just happens to be one 1/3, inspired me to devise a far shorter division by 3 algorithm. Pruning code elsewhere I've written a useable subroutine TARGET. TARGET takes x and y in B and C respectively, returns graphic character mask in A, and screen character address in HL. Thus to STING the x, y position in BC requires the minimal code

```
STING:  CALL    TARGET
        CMA
        ANI     ^B01111111
        AND     M
        MOV     M, A
```

Likewise to unsting one would use the 8080 code

```
LISTING:  CALL    TARGET
          ORA     M
          MOV     M, A
```

Now the crux of our discussion is the TARGET subroutine in the box. The TARGET subroutine doesn't include any loops or jumps – it is a piece of in-line coding optimized for run time. In fact execution requires up to 220 machine cycles which takes 110 micro seconds.

IS DYNAMIC GRAPHICS POSSIBLE?

Suppose one utilized the TARGET subroutine to UNSTING 100 points on the TV screen, and then STUNG another 100 points. These operations would take rather more than 200 times 110 microseconds (i.e. just for the TARGET calls alone) which is about 1/40 second. Thus even though TARGET is very economical of processing time, it isn't the whole story. However by careful coding a system could be devised. There are several routes to further economies of processor time.

a) Organize lists of squares to be stung or unstung by row, so that only for the first square in a row would one need to run through the TARGET routine in full.

b)Organize lists of squares to be stung with mask character appropriate.

c) Erase whole screen by very simple routines, rather than particular squares.

d) Update screen more slowly than the screen refresh rate, perhaps updating every 1/15 sec. But one would then need to take great care to avoid big light flashes, so this strategy incompatible with a rapid total clearing of screen as per (c).

All things considered, a crude dynamic graphics could be implemented, with about 50% of processor time available for computation of new positions of screen objects.

SUMMARY

The graphics capabilities of the Poly–88 are a bonus feature with this personal system. Because of the peculiarities of screen addressing, if one used the standard division algorithms anything other than data graphics would be out of the question. However by careful coding – as exemplified by the TARGET SUBROUTINE listed here – a useable graphic system could be developed. I hope I've also indicated at least part of the story as to what must be considered when contemplating graphics: one needs to consider what are the basic macros needed as building blocks.

Footnote: TARGET works on the Intel 8080. Polymorphic systems are however switching to the NEC processor, so some modification is probably necessary. (Someone has "borrowed" my copy of the December *DDJ* so I don't know the difference).

```
                        TITLE   TARGET
0C00                    LOC     ~006000 ;LOCATION  0C00H
                ;;      TARGET SUBROUTINE
                ;;      LOCATABLE ANYWHERE IN MEMORY
                ;;      START CONDITIONS AS FOLLOWS
                ;;      (B) = X    ;  (C) =  Y  ;
                ;;      X,Y BOTH VALID SCREEN POINTS
                ;;      0=<X<48 ;  0=<Y<128  ;
                ;;      START SCREEN ADDRESS 88H
                ;;      REGISTER DE NOT USED
                ;;      END CONDITIONS AS FOLLOWS
                ;;      (HL) = SCREEN ADDRESS (ASSUMING START SCREEN IS 8800H)
                ;;      (A)  HAS EXACTLY 7 ZERO BITS AND
                ;;                   HAS 1 IN  STING LOCATION


0C00    78  TARGET: MOV     A,B
0C01    07          RLC
0C02    07          RLC
0C03    80          ADD     B
0C04    47          MOV     B,A     ;(B)=X + LS.LS.X ;LS=LEFT SHIFT
0C05    1F          RAR
0C06    1F          RAR
0C07    1F          RAR
0C08    1F          RAR
0C09    E6          ANI     ~017    ;0FH ;(A) = RS.RS.RS.RS(B) ;RS=RIGHT SHIFT
0C0A    0F
0C0B    80          ADD     B
0C0C    C6          ADI     ~05
0C0D    05
0C0E    47          MOV     B,A     ;(B) IS IN THREE PARTS AS FOLLOWS
                    ;; BITS 1,0 ARE GARBAGE
                    ;; BITS  7,6,5,4 ARE THE INTEGER PART OF X/3
                    ;;      ARE ROW NUMBER
                    ;; BITS  3,2 ARE 1 + REMAINDER X/3 UNLESS
                    ;;              BOTH ZERO WHEN REMAINDER IS ZERO ALWAYS
0C0F    E6          ANI     ~B11110000
0C10    F0
0C11    6F          MOV     L,A
0C12    26          MVI     H,~B00100010
0C13    22
                ;;FOR SCREEN START ADDRESS F800H REPLACE PREVIOUS LINE
                    ;;BY
                ;;  MVI     H,~B00111110
0C14    29          DAD     H
0C15    29          DAD     H       ;; (HL)=ADDRESS OF COL 0 IN OK ROW
0C16    78          MOV     A,B
0C17    1F          RAR
0C18    1F          RAR
0C19    E6          ANI     ~03     ;03H
0C1A    03
0C1B    3D          DCR     A

0C1C    D6          SUI     1
0C1D    01
0C1E    DE          SBI     ~03
0C1F    03
0C20    2F          CMA
0C21    47          MOV     B,A
0C22    79          MOV     A,C
0C23    37          STC
0C24    3F          CMC
0C25    1F          RAR
0C26    F5          PUSH    PSW     ;STORE CARRY
0C27    85          ADD     L
0C28    6F          MOV     L,A
0C29    F1          POP     PSW
0C2A    78          MOV     A,B
0C2B    D8          RC      ;ALL DONE IF Y IS ODD NUMBER
0C2C    17          RAL
0C2D    17          RAL
0C2E    17          RAL
0C2F    C9          RET
        ;;EXPLANATION OF GENERATION OF STING CHARACTER FROM REMAINDER STUFF
        ;;STEP BY STEP CHANGES IN (A)

        ;;INITIAL    DCR  A      SUI 1   CARRY?   SBI   3  (A)LOWER  CMA
        ;;  0         F           E               FB     ~B1011  ~B100
        ;;  1         0           F       YES     FB     ~B1011  ~B100
        ;;  2         1           0               FD     ~B1101  ~B010
        ;;  3         2           1               FE     ~B1110  ~B001

                END
```

Permission to reproduce for any non-commercial purpose granted provided source quoted. Elsewise copyright retained by author, © *H.A. Cohen 1977*

TIME FOR SUPER VIDEO?

February 26, 1977

Almost any function that can be performed by a few dozen MSI chips can be handled equally well by a single, special-purpose LSI chip. Custom LSI chips are expensive in small lots but fairly cheap when produced in quantity. Now that the hobby market has grown to respectable dimensions, I hope that more manufacturers will begin to heed the needs of that market. Although improvements can and no doubt will be made in MPU design, the immediate need is in the peripheral area. As I see it, there is need for a one-chip video generator, a one-chip floppy controller, a one-chip cassette interface, a one-chip DMA controller, and a real-time vectored interrupt chip. LSI chips for some of these functions already exist and should find their way into the next generation of personal computers.

Perhaps the most urgent need is for a good $25 video chip. Although video displays are often used in personal computer systems, present designs for video boards are a throwback to the Paleozoic era when video displays were used mainly in time-sharing terminals. In that context it made sense to use on-board RAM for the display, but in a personal computer this is simply a nuisance. A video generator could, and should, be designed to access any part of memory directly. Further, it should be possible to select the type font, graphics set, character size, line length, number of lines per display, color, etc. dynamically under program control. No technical breakthrough is needed to any of these, but they would be expensive to add to existing video boards. A custom chip could handle all of these functions at reasonable cost and with silicon to spare. Now, if someone would only convince the people at Intel, Mostek, or Motorola, we might soon see super video chips coming off the assembly lines.

Jim Day

MICROCOMPUTER CLUBS IN JAPAN

Microcomputer clubs are sprung up in Japan, too, these days. I'd like to mention some notable clubs.

(1) *Japan Microcomputer League*

This loosely knit club was organized by student computer-freaks. They started publishing a small hobby-electronics magazine called I/O in October, 1976. To amazement by bookstore owners who understood I/O as "one and zero", the magazine is selling well in Tokyo and the number has reached an astounding (from Japanese standard) 8000 in the second month. The success seems to be that the magazine has a special focus on micro-computers and also carries articles on laser hobby, music synthesizer and TV-game kits, etc. The editor of I/O is Mr. Nishi, a sophomore at Waseda University. He wants to convert a sonosheet (thin plastic) record player into a cheap read-only disc memory and distribute Tiny BASIC on sonosheets. Has anyone tried this?

(2) *Japan Microcomputer Club*

Organized by university professors, research engineers and vendors, this club started in October, 1976, has some 600 members and publishes a newsletter bimonthly. I am complaining that this club is head-heavy – too many committees whose members have never touched microcomputers – and suggesting that they resolve committees and purge figure-head members. The sole activity of this club is to hold lectures monthly by vendor spokesmen and members mostly just listen and have not started their own microcomputers.

(3) *NEC Microcomputer Club*

The NEC (Nippon Electric) is a big computer and semi-conductor manufacturer and is second-soucing Intel 8080A series and other chips. NEC also sells KIM-1-like 8080A kits (TK-80) with hex-heys and 8 digit LED displays (but no peripheral kits). The NEC club was organized as a fee-free club by NEC to promote their own sales. NEC claims they received 3,000 postcard applications, which shows high interest toward microcomputers among Japanese. The only activity NEC is planning is lectures by NEC personnel.

(4) *Microcomputer Classroom*

This is my own no-fee club with 100 members set up at the University of Tokyo to promote the use of microcomputers in experiment control (in physics, chemistry, medicine, etc.) and personal intelligent terminals. We have about ten microcomputers (of all kinds), two teletypes, a jet-ink teletype, an IBM typewriter converted into an intelligent terminal, a floppy disc, a TV display, two cassette interfaces (one is high quality), a PROM writer and an eraser. So my lab is a kind of showcase for Japanese microcomputers with contributions from many vendors. We are holding monthly meetings for exchange of experience.

Of course, there are many small computer clubs in Japanese universities and elsewhere. Although interests are very high, as indicated by the fact that 24,000 copies of my first microcomputer books has been sold, our trouble is that chips and kits are still very expensive in Japan. There are only two or three computer stores in the Tokyo area, where U.S. made kits are sold at the rate of ¥ 500 – ¥ 550 (sometimes ¥ 1,000) for a dollar (regular rate is ¥ 300/$). This means that microcomputers are about twice as expensive here as in the U.S. So we are now trying to design cheap peripherals and pursuading small businesses to sell good kits at low prices.

Haruhisa Ishida
Computer Centre
University of Tokyo
Tokyo, Japan

NEGATIVE NOTES, BASICALLY SPEAKING

Gentlemen: 1977-01-25 **[Top-down Date]**

As I understand from the first volume of the journal, you call yourselves Dragons, **[Not all of us; only the more conservative ones]** and that's about the only sensible notion I managed to find in the entire project. It is true that my acquaintance with *Dr. Dobb's Journal* is merely casual so I don't know about the background, but provided that you don't want to express the opposite of what you print, I have the following principal objections against your handling that Tiny BASIC business, however tiny or basic it may be.

1. There are two kinds of so-called high level programming languages: those making the programmer think as the machine does – and one or two others. BASIC – along with FORTRAN, ASSEMBLER, and COBOL – decidedly belongs to the first category. **[We couldn't agree more.]** It's no more nor less difficult to learn and applicate than home computers' instruction codes which are easy enough. **[True for us pros. Not so for novices.]**
2. The only remaining positive aspect is portability, and I refuse to accept this until there is some constructive method for making different Tiny BASIC interpretive systems (there exist dozens, don't they?) running on different machines (dozens again) react equal to the same program. (You ought to be happy anyway to have to do with feelingless machines. An interpretive routine or compiler for a home computer sounds as disproportionate as a bicycle with automatic gear-shifting, **[Why?]** and I'm sure if your machines could scream, they would.)
3. A language design frequency of almost one Tiny BASIC version per month is certainly impressive. Programmers ought to have realized that God invented many different languages in Babylon not to enjoy but to punish mankind, and, after all, he never intended implementing all of them for the 8080.

So why not write – and if necessary publish – programs in an informal and ad-hoc-language fitting to the problem, not the computer? **[You write 'em; we'll publish (maybe).]** And have every programmer do the compilation by hand in, say, half an hour. By that one would achieve

- a readable program (the original one)
- a fairly effective and small machine program
- and no pre-fabricated code of which nobody – except the author, miles away – knows exactly how it behaves in special (e.g. error) cases.

What I really like about the *Journal* is the Elvis-Presley-era-disc-jockey-style I found in a few lines (e.g. "Do, do, do tell me about . . . "), and some humorous passages (e.g. "Biblical Mathematics"), but there are better ones in the ALGOL68 Report.

Theory of formal languages tells us there are infinitely many of them, so don't despair when publishing the (n + 1)-th and even then not coming to an end!

Thomas Alexander Matzner
HeBstraBe 64
D-8000 Munchen 40
Bundesrepublik Deutschland

Extended accuracy arithmetic routines

Dear Sirs,

Much to my pleasant surprise and amazement I received a complimentary copy of *Dr. Dobbs* and I am very impressed by it.

I enclose the source paper tape and limited documentation for an x "Extended Accuracy Arithmetic" package that I wrote. It's written for an 8080, is 776 DEC. bytes long and will handle positive and negative numbers of up to 10**308 magnitude; the larger numbers being a little slow. It works excellently with numbers in the range 40 – 80 bits.

Maybe you know of enthusiasts who are interested in solving combinatorial and permutation, or any other problems that require large "crunching" power; this package will handle 170! with its capacity for up to 2K bit word lengths.

I'm very impressed with "Tiny Basic" and would like to know more about it.

Any criticisms, bugs, praise or correspondence to me at the following address.

Yours sincerely,

M.G. Dineley

B22, Digital Processes Lab
Dept. E.E.&E.
University of Manchester Institute
of Science and Technology
Manchester M6O 1QD England

```
      EXTENDED ACCURACY ARITHMETIC PACKAGE
      ====================================

ABSTRACT
--------     THERE IS FREQUENTLY A NEED TO MANIPULATE INTEGERS OF A MUCH GREATER
SIZE THAN THE 8 BITS OF CAPACITY THAT MANY MICROPROCESSORS CAN DIRECTLY
HANDLE; THE GENERAL CASE IS CONSIDERED WITH PROCEDURES CAPABLE OF MANIPULATING
MANY BYTES - 1 TO 127, THAT IS INTEGERS OF MAGNITUDE 1 TO 10**308.

MULTIBYTE FORMAT
----------------     ONE EFFECTIVE WAY OF STORING THESE LARGE INTEGERS IN MEMORY
  IS TO USE THE REPRESENTATION WHEREBY THE FIRST LOCATION (ADDRESS) CONTAINS
  A LENGTH AND SIGN INDICATOR WHICH SPECIFIES THE NUMBER,N , OF BYTES
USED TO STORE THE INTEGER, THE NEXT LOCATION (ADDRESS+1) CONTAINS THE LEAST
SIGNIFICANT 8 BITS OF THE INTEGER, AND THE FINAL LOCATION (ADDRESS+N) CONTAINS
THE MOST SIGNIFICANT 8 BITS.

ARITHMETIC OPERATIONS- HIERARCHY
-------------------------------     THERE IS A DISTINCT HERARCHY OF ARITHMETIC
OPERATIONS; THE MORE COMPLEX ARE SQUARE ROOT, DIVISION & MULTIPLICATION,
THE SIMPLER ONES ARE SUBTRACTION AND ADDITION AND THE SUBSIDIARY ONES ARE
COMPARISON, LEFT SHIFTING ( MULTIPLICATION BY RADIX ), RIGHT SHIFTING
(DIVISION BY RADIX) AND INCREMENTING. THESE LATTER OPERATIONS ARE TERMED
SUBSIDIARY BECAUSE THEY ARE PROCESSES REQUIRED BY THE MORE COMPLEX OPERATIONS.
  IT IS NOT TOO SURPRISING THAT THE ALGORITHMIC METHODS USED TO PERFORM THESE
MORE COMPLEX OPERATIONS ARE FUNDAMENTALLY INDEPENDANT OF THE RADIX USED,
THOUGH THEY ARE CONSIDERABLY SIMPLIFIED BY USING A BINARY RADIX BECAUSE OF
THE LIMITED RANGE OF VALUES FOR ANY DIGIT I.E. 0 OR 1.

GENERAL ALGORITHMS
------------------

      LET   D BE ANY DIGIT ; 0<= D < R , WHERE R IS THE RADIX
            N BE ANY ARBITRARY INTEGER

      SQUARE ROOT
      ***********

      LET   A BE THE ARGUMENT
            B BE A WORKING ACCUMULATOR
            C BE A WORKING ACCUMULATOR INITIALLY ZERO

1/    CHOOSE THE GREATEST B : B=D*R**2*N  &  B*(B+C)<= A
2/    IF B=0  THEN GOTO 5
3/    A=A-B*(B+C)
4/    C=C+2*B   GOTO 1
5/    RESULT = C/2

      DIVIDE
      ******

      LET A BE THE DIVIDEND
          B BE THE DIVISOR
          C BE THE ANSWER INITIALLY ZERO

1/    CHOOSE THE GREATEST D AND THE GREATEST N : D*B*R**N<= A
2/    A=A-D*B*R**N
3/    C=C*R+D
4/    IF N=0 GOTO 6
5/    N=N-1 GOTO 1
6/    REMAINDER= A: RESULT=C

      MULTIPLY
      ********

      LET A BE THE MULTIPLICAND
          B BE THE PRODUCT INITIALLY ZERO
          C BE THE MULTIPLIER

1/    D=C MOD R
2/    C=C/R
3/    B=B+D*A
4/    A=A*R
```

```
5/      IF C<>0 GOTO 1
6/      RESULT=B
```

BINARY ALGORITHMS

IN THIS CASE R=2 AND D=0,1 SIMPLIFYING THE ALGORITHMS

SQUARE ROOT

```
     LET A BE THE ARGUMENT
         B BE A WORKING ACCUMULATOR
         C BE A WORKING ACUMULATOR INITIALLY ZERO

1/      CHOOSE THE LEAST B : B=2**2N >A
2/      C=C+B
3/      IF A<C GOTO 6
4/      A=A-C
5/      C=C+B  GOTO 7
6/      C=C-B
7/      C=C/2
8/      B=B/4
9/      IF B<>0 GOTO 1
10/     RESULT=C
```

DIVIDE

```
     LET A BE THE DIVIDEND
         B BE THE QUOTIENT INITIALLY ZERO
         C BE THE DIVISOR

1/      CHOOSE THE LEAST N : A< C*2**N
2/      IF A<C*2**N GOTO 5
3/      A=A-C*2**N
4/      B=B+1
5/      B=B*2
6/      N=N-1
7/      IF N<> 0 GOTO 2
8/      RESULT=B
```

MULTIPLY

```
    LET A BE THE MULTIPLICAND
        B BE THE PRODUCT INITIALLY ZERO
        C BE THE MULTIPLIER

1/      IS THE LEAST SIGNIFICANT BIT OF C SET
          I.E.   IF C MOD 2 <> 1  GOTO 3
2/      B=B+A
3/      A=A*2
4/      IF C=0 GOTO 6
5/      C=C/2 GOTO 1
6/      RESULT =B
```

REPRESENTATION OF ZERO

ZERO HAS BEEN CHOSEN TO BE REPRESENTED BY A MAGNITUDE OF ZERO (NUMBER OF BYTES OF PRECISION = 0), RATHER THAN A REPRESENTATION OF LENGTH OF ONE BYTE AND VALUE OF ZERO. ZERO IS A SPECIAL INTEGER OF NO SIGN AND NO MAGNITUDE.

REPRESENTATION OF NEGATIVE NUMBERS

AS WE ARE CONSIDERING THE GENERAL CASE, THE ROUTINES MUST BE CAPABLE OF MANIPULATING BOTH POSITIVE AND NEGATIVE NUMBERS OF UP TO THE MAXIMUM RANGE, AND ZERO.

THERE ARE TWO USEFUL REPRESENTATIONS OF NEGATIVE NUMBERS

1/ TRUE MAGNITUDE WITH A RESERVED SIGN INDICATOR
2/ TWO'S COMPLEMENT WHERE -X IS REPRESENTED BY 2**N-X, 2**N > X

THE LATTER FORMAT WAS ADOPTED INITIALLY AND WAS PROVED TO BE A BAD CHOICE BY THE CRITERIA OF EFFICIENCY, WE WILL NOW CONSIDER WHY.

EFFICIENCY CONSIDERATIONS

THE OPERATIONS OF ADDITION AND LEFT SHIFTING CAN PRODUCE RESULTS OF GREATER MAGNITUDES THAN THE OPERANDS, I.E. THE AMOUNT OF BYTES USED TO REPRESENT THE NUMBER MAY NEED TO BE INCREASED, SIMILARLY SUBTRACTING AND RIGHT SHIFTING REDUCES THE MAGNITUDES. THE TWO'S COMPLEMENT SYSTEM REQUIRES THAT THE SIGN BIT (8*N-1) MUST BE PROPOGATED UNDER ANY INCREASE OR REDUCTION SIZE; THIS REQUIRES TWO DISTINCTLY DIFFERENT ADJUSTMENTS. ONE FOR POSITIVE AND ONE FOR NEGATIVE NUMBERS. HENCE DOUBLING THE AMOUNT OF "FILTERING" REQUIRED COMPARED TO THE TRUE MAGNITUDE REPRESENTATION.

THERE ARE NO KNOWN ALGORITHMS FOR DIVIDING NEGATIVE NUMBERS. HENCE THE DIVISION OF NEGATIVE NUMBERS REQUIRES TWO CONVERSIONS UNDER THE TWOS COMPLEMENT REPRESENTATION.

THE RESULT OF SUBTRACTING A LARGE POSITIVE NUMBER FROM A SMALLER ONE GIVES A RESULT THAT IS DIRECTLY COMPATIBLE WITH THE TWO'S COMPLEMENT REPRESENTATION, AS DOES THE ADDITION OF MIXED POSITIVE AND NEGATIVE NUMBERS WHEN USING FIXED LENGTHS. HOWEVER IN THE GENERAL CASE OF VARYING LENGTHS OF BYTES THERE ARE PROBLEMS IN THE REMOVING THE SPURIOUS TERMS OF 2**N.

CONSIDER (X) + (2**N - Y) AND (X) - (2**N - Y) X>2**N>Y

WITH STRAIGHTFORWARD ADDITION AND SUBTRACTION WE HAVE

X-Y + (2**N) AND X+Y - (2**N)

WHERE THE TERMS 2**N ARE SPURIOUS AND MUST BE REMOVED.

THIS GREATLY COMPLICATES THE GENERALISED ADDITION AND SUBTRACTION OF TWO'S COMPLEMENT REPRESENTED NUMBERS.

THE MAJOR DISADVANTAGE OF USING A TRUE MAGNITUDE AND SEPARATE SIGN REPRESENTATION IS IN THE CASE WHERE A LARGER NUMBER IS SUBTRACTED FROM A SMALLER NUMBER. THE DIRECT RESULT IS THEN IN TWO'S COMPLEMENT FORM AND MUST BE ADJUSTED TOGETHER WITH THE SIGN TO MAINTAIN THE CONVENTION; THIS IS THE ONLY CASE WHERE ANY COMPLEMENTATION NEED BE PERFORMED THOUGH UNDER THIS CONVENTION.

MODULARITY AND HIERARCHY

BECAUSE OF THE HIERARCHICAL NATURE OF THESE ARITHMETICAL OPERATIONS IT IS EASIER TO IMPLEMENT THEM AS INDIVIDUAL SUBROUTINES. EACH SUBROUTINE HAS BEEN PROGRAMMED TO MAINTAIN THE MULTI BYTE MAGNITUDE AND SIGN STORAGE CONVENTION FOR ANY COMBINATION OF OPERANDS, EXCEPTING OVERFLOW CONDITIONS: WHEN A RESULT BECOMES TOO LARGE TO BE REPRESENTED WIHITN THE CONVENTION THE ROUTINE WILL HALT.

THIS DISPERSAL OF "INTELLIGENCE" AMONGST THE MORE BASIC ROUTINES MAKES THEM MUCH LONGER, SLOWER AND MORE COMPLEX BUT GUARENTEES THAT NO "FILTERING" IS REQUIRED BEFORE SUBMITTING OPERANDS TO THEM AND ENSURES THAT THE HIGHER OPERATIONS REQUIRE ONLY A MINIMUM OF "FILTERING" AND HENCE A TREMENDOUS REDUCTION IN SIZE AND COMPLEXITY.

DESCRIPTION OF ROUTINES

```
      ** LENGTH INDICATOR UTILITIES
      == LDC1 : USES .
         LOADS REGISTER C WITH THE CONTENTS OF THE LENGTH INDICATOR SPECIFIED
          BY HL REGISTERS- STRIPS THE SIGN BIT

      == LDC2 : USES . LDC1
         AS LDC1 BUT RETURNS TO CALLERS CALLER IF RESULT IS ZERO

      == LDB1 : USES .
          LOADS REGISTER B WITH THE LENGTH INDICATOR SPECIFIED BY DE REGISTERS
         - REMOVES SIGN BIT

      == LDB2 : USES . LDB1
         AS LDB1 BUT RETURNS TO CALLERS CALLER IF RESULT IS ZERO
```

```
      == INRM : USES .
         INCREMENTS LENGTH INDICATOR SPECIFIED BY HL REGISTERS - PRESERVES
          SIGN INDICATOR.  HALTS ON OVERFLOW CONDITION

      == DCRM : USES .
         DECREMENTS LENGTH INDICATOR SPECIFIED BY HL REGISTERS - REDUCES IT
         TO ABSOLUTE ZERO EVEN IF SIGN INDICATOR IS SET - RETURNS WITH
         CONDITION CODE OF ZERO IF ZERO.

      ** DEQUE UTILITIES
      == PSHL : USES . LDC1
         INSERTS A BYTE FROM A REG INTO LOW ORDER OF MULTIBYTE DATA SPECIFIED
         BY HL PAIR.  PUSHES ALONG ALL OTHER BYTES.

      == POPL : USES . LDC2
         REMOVES A BYTE FROM LOW ORDER SPECIFIED BY HL INTO A REGISTER, PULLS
         BACK ALL OTHER BYTES.

      == PSHH : USES . LDC1
         INSERTS A BYTE FROM A REG INTO HIGH ORDER OF MULTIBYTE DATA SPECIFIED
         BY HL PAIR.

      == POPH : USES . LDC2
         REMOVES A BYTE FROM HIGH ORDER  BY HL PAIR INTO A REGISTER.

THESE FOUR ROUTINES TOGETHER CAN BE USED TO MANIPULATE MULTIBYTE DATA AS
A DEQUE (DOUBLE ENDED QUEUE) , GIVING TREMENDOUS FLEXIBILITY , E.G.  LIFO
FIFO AND ROTATION.

      ** ARITHMETIC- SIMPLE
      == INCR : USES . LDC1
         INCREMENTS A NUMBER SPECIFIED BY HL PAIR AND EXTENDS IF NECESSARY

      == LEFT : USES . LDC2
         DOUBLES A MULTIBYTE NUMBER SPECIFIED BY HL ( SHIFTS BY ONE BIT)

      == RIGHT : USES . LDC2
         HALVES A MULTIBYTE DATA SPECIFIED BY HL (SHIFTS BY ONE BIT)

      == MOOV : USES . LDB1
         MOVES A MULTIBYTE DATA  FROM DE PAIR TO HL PAIR.  INCLUDING ZERO.

      == PARE : USES . LDC1
         COMPARES A MULTIBYTE DATA FROM DE WITH HL PAIR .  SETS ALL CONDITION
         CODES ZERO IF EQUAL AND CARRY IF GREATER THAN.

      == ADD1 : USES . LDC1 . LDB1
         ADDS MULTIBYTE DATA HL TO DE - RESULT IN DE :: POSITIVE NUMBRES ONLY

      == SUB1 : USES . LDC2 . LDB1
         SUBTRACTS MULTIBYTE DATA HL FROM DE - RESULT IN DE :: POSITIVE ONLY



      ** ARITHMETIC - HIGHER
      == AD1 & SB1 : USES . ADD1 . SUB1
         ADDS AND SUBTRACTS POSITIVE AND NEGATIVE NUMBERS

      == MULT : USES . LDB2 . MOOV . RIGHT . LEFT . ADD1 . / WORK AREAS T1 & T2
         MULTIPLIES DE BY HL - RESULT IN DE

      == MOD . USES . LDC1 . LDB1 . MOOV . PSHL . PARE . SUB1 . INCR .
          RIGHT . LEFT . / WORK AREAS T1 & T2
         REDUCES DE MODULO HL
       * NOTA BENE . -30 MOD 7 = 2  NOT 5
      == DIV
         DIVIDES DE BY HL - RESULT IN DE
      == DIVR
         AS DIV BUT ROUNDS SO THAT 5/3=2 NOT 1

      == SQRT : USES . PSHL . ADD1 . PARE . SUB1 . RIGHT . INCR . MOOV.
         . / WORK AREAS T1 & T2
         SQUARE ROOTS DE - RESULT IN DE
      == SQTRTR
         AS SQRT BUT ROUNDS SO THAT SQRTR(31)=6
```

BENCHMARKS +-10%

N=NUMBER OF BYTES IN AN INTEGER
TIMES ARE IN MILLISECONDS (UNLESS SUFFIXED WITH S =SECONDS)
FOR ROUTINES TO EXECUTE ON AN INTELLEC 8

OPERATIONS ARE ADDITION , SUBTRACTION , MULTIPLICATION , DIVISION AND SQUARE ROOT

N	N+N	N-N	N*N	2N/N	R(N)
1	.25	.28	3.7	8.2	5.7
2	.3	.35	8.4	13	12
4	.36	.5	21	27	27
6	.43	.65	40	45	45
8	.5	.8	63	67	68
10	.55	.95	91	97	94
15	.85	1.2	180	175	175
20	.95	1.5	320	270	280
25	1.2	1.8	470	420	415
30	1.35	2.2	670	570	570
40	1.75	2.7	1.15S	1.0S	.95S
50	2.2	3.6	1.75S	1.45S	1.42S
60	2.5	4.2	2.5S	2.1S	2.0S
100	4.0	6.8	-	-	5.3S

LISTING OF ASSEMBLED EXTENDED ACCURACY ARITHMETIC PACKAGE

```
:100100007E17B71F4FB7C9CD0001CA1C01C91A1706
:10011000B71F47B7C9CD0E01CA1C01C93333C97E09
:1001200034AEF0767EFE81CA2F0135C8AEF0763649
:1001300000C9F506007EB74FCA4501097E23772B1B
:100140002B0DC23C0134CA7E01F123772BC97EB747
:10015000C84FE52346237E2B77230DC25501E13599
:1001600078C9E5F534CA7E014E060009F177E1C988
:100170007EB7C84FE50600097EE1F535F1C976E5A1
:10018000CD0001CA8F012334C297010DC28601231D
:1001900036O1E1CD1F01C9E1C9CD0701E5237E1775
:1001A000770DC29D01DA8F01E1C9CD07010600097 3
:1001B0007E1F77472B0DCABF017E1F77C3B4010591
:1001C000F0F5CD2401F1C9CD0E0177C81A77D5E538
:1001D00013231A7705C2D001E1D1C97E173F1F470B
:1001E0001A173F1FB8C0CD0001E5D5060009EB097D
:1001F000EB1ABEC2FC012B1B0DC2F101D1E1C9CD2E
:10020000701D5E5CD0E0191EBD214022F8E77AF09
:1002100080CA360223131A8E7705CA3B020DC21418
:10022000022237E897705C22102D24202233601D100
:10023000E1CD1F01EBC923131A88770DC23602DA0C
:100240002C02E1D1C9CD0701E5D5CD0E0191D25BDC
:1002500002EB2F8E77AF80EBCA680223131A9E122F
:100260000DCA8E0205C25B022313789E120DC2686E
:1002700002E1E5D57E173F1F77E67F4F37237E2FBC
:10028000887700DC27D02D1C39502131A991205C257
:100290008A02DA7102E11AB9C2A2021BCD2401C29C
:1002A0009602EBE1C9AFC3AA0237F51AAEFAB8025B
:1002B000F1DABC02CDFF01C9F1DAB402CD4502C9C1
:1002C000CD1501E51AAEF5D5E5210010CDC701AF7A
:1002D00012D1210012CDC701D12100127EB7CAF37D
```

```
:1002E00002CDAA01210010D2ED02CDFF01CD99016E
:1002F000C3D902F1E1F01AEE8012C9AFC680C3077C
:10030000C3AF37C30703AFE5F51AAE07D21203F107
:10031000C3F5CD0001CA9F03CD0E0191FA9F03D594
:10032000C474FC51AE67F12EB2100107EE67F772F
:10033000CDC701AF320012D1CD32011DC238034FFB
:1003400042D1D5C5210010CDDB01DA5603CD4502DF
:1003500021001 2CD7F01210010CDAA01C10DF2674D
:10036000030E0705FA7203C5210012CD9901C1C31E
:1003700041 03D1F1FA9C03D28B03F5CDDB01DA8A7C
:10038000 03CA8A03210012CD7F01F1EB110012E6AE
:1003900001CA9803 1AF68012CDC701EBAFE1C9F18B
:1003A00037E1C9B7C3A80337F51AB737F2B103C1AC
:1003B000C90E01210012712371D5573E002BCD3299
:1003C00001 15C2BE0311001012CDFF01EBD1CDDB30
:1003D00001F5D44502F1D5EB210012F5D4FF01F16E
:1003E000DC4502EBCDAA01EBCDAA01CDAA017EB777
:1003F000C2C903E1F1DA0304CDDB01D20304EBCD82
:10040000 7F01EBCDC701EBB7C90000000000000081
```

```
;============= EXTENDED ACCURACY ARITHMETIC PACKAGE ===============

;* THESE ARITHMETIC ROUTINES REQUIRE AND MAINTAIN A CONVENTION OF
;* STORAGE FOR VERY LARGE INTEGERS OF THE FORM :-
;*
;*   [ADDRESS  ], [ADDRESS+1], ... , ... , ... , [ADDRESS+N]
;*   SIGN LENGTH  LEAST SIGNIFICANT                MOST SIGNIFICANT
;*   BIT7 BITS0-6     8 BITS                          8 BITS
;*
;* THESE DATA AREAS ARE GENERALLY REFERRED TO BY INDEX REGISTER
;* PAIRS DE AND HL
;* SIGN BIT=1 IMPLIES -VE INTEGER
;* ZERO IS REPRESENTED BY SIGN & LENGTH INDICATOR BYTE = 0
;*
;* USES TWO WORK AREAS OF UP TO 128 BYTES EACH T1 &T2
;*
;*== AUTHOR M.G.DINELEY , B22 DIGITAL PROCESSES LAB.  U.M.I.S.T.
;*==  JULY '76

;* GROUP OF SUBROUTINES TO ADAPT FOR SIGN INDICATOR BEING INCORPORATED
;* INTO LENGTH INDICATOR BYTE
;* LOADS MULTIBYTE LENGTHS INTO DECREMENT REGISTERS B&C
;* VERSION 2 ABORTS ON LENGTH ZERO , RETURNS TO CALLERS CALLER
;* STRIPS SIGN (BIT 7) FROM LENGTH INDICATOR
;==
        ORG 100H
T1      EQU 1000H
T2      EQU 1200H
LDC1:   MOV A,M         ; GET LENGTH
        RAL
        ORA A           ; CLEAR CARRY
        RAR             ; STRIP SIGN
        MOV C,A         ; LOAD C
        ORA A           ; SET CONDITION CODE
        RET

LDC2:   CALL LDC1
        JZ ABRT
        RET

LDB1:   LDAX D
        RAL
        ORA A
        RAR
        MOV B,A
        ORA A
        RET

LDB2:   CALL LDB1
        JZ ABRT
        RET

ABRT:   INX SP          ; RETURN TO -
        INX SP          ;- CALLERS CALLER
        RET

;* SUBROUTINES TO INCREMENT AND DECREMENT LENGTH INDICATORS ( + SIGN)
;* HALTS ON OVERFLOW , >127 BYTES

INRM    MOV A,M
        INR M
        XRA M           ; SIGN CHANGE?
        RP
        HLT

DCRM:   MOV A,M
        CPI 81H         ; IS IT -VE ZERO?
        JZ LABB
        DCR M
        RZ              ; RETURN WITH CONDITION CODE SET
        XRA M
        RP
        HLT
LABB:   MVI M,0
        RET

;*== GROUP OF SUBROUTINES FOR DEQUE (DOUBLE ENDED QUEUE ) MANIPULATION
;*PSHL INSERTS BYTE FROM A REG.  INTO LOW ORDER OF MULTIBYTE DATA
;=    HL=A(DATA), A=BYTE  : BC,PSW=CORRUPTED
PSHL:   PUSH PSW        ; SAVE BYTE
        MVI B,0
        MOV A,M         ;   GET LENGTH
        ORA A
        MOV C,A
        JZ LX01
        DAD B
L001:   MOV A,M
        INX H
        MOV M,A         ; SHUFFLE-
        DCX H           ;- UP DATA
        DCX H
        DCR C
        JNZ L001
LX01:   INR M
        JZ HALT
        POP PSW
        INX H
        MOV M,A
        DCX H
        RET

;* POPL REMOVES BYTE FROM LOW ORDER INTO A REG.
;=  HL=A(DATA) : A=BYTE , BC,PSW=CORRUPTED
POPL:   MOV A,M     ; GET LENGTH
        ORA A
        RZ
        MOV C,A
        PUSH H
        INX H
        MOV B,M         ; REMOVE BYTE
L002:   INX H
        MOV A,M
        DCX H           ; SHUFFLE-
        MOV M,A         ;- DOWN DATA
        INX H
        DCR C
        JNZ L002
        POP H
        DCR M       ; DECREMENT LENGTH
        MOV A,B         ; RETURN BYTE
        RET

;* PSHH INSERTS BYTE FROM A REG.  INTO HIGH ORDER OF MULTIBYTE DATA
;== HL=A(DATA) A=BYTE : BC,PSW=CORRUPTED
PSHH:   PUSH H
        PUSH PSW
        INR M       ; INCREMENT LENGTH
        JZ HALT         ; OVERFLOW ERROR
        MOV C,M         ; GET LENGTH
        MVI B,0
        DAD B           ; GET A(HIGH END DATA)
```

```
        POP PSW
        MOV M,A       ; INSERT BYTE
        POP H
        RET

        ;* POPH REMOVES A BYTE INTO A REG.  FROM HIGH ORDER OF MULTIBYTE DATA
        ;= HL=A(DATA)  : A=BYTE BC,PSW=CORRUPTED
POPH:   MOV A,M     ; GET LENGTH
        ORA A
        RZ
        MOV C,A
        PUSH H
        MVI B,0
        DAD B         ; GET A(HIGH END DATA)
        MOV A,M       ; GET BYTE
        POP H
        PUSH PSW
        DCR M         ; REDUCE LENGTH
        POP PSW
        RET
HALT    HLT


        ;** SIMPLE ARITHMETIC SUBROUTINES
        ;* INCR INCREMENTS THE MODULUS OF A MULTIBYTE DATA
        ;=  HL=A(DATA)  : A,C,PSW=CORRUPTED
INCR:   PUSH H
        CALL LDC1
        JZ L004
L003:   INX H
        INR M           ; INCREMENT DATA
        JNZ L005        ; O/FLOW?
        DCR C
        JNZ L003        ; END?
L004:   INX H
        MVI M,1         ; EXTEND DATA
        POP H
        CALL INRM       ; INCREMENT PRECISION
        RET
L005:   POP H
        RET

        ;* LEFT LEFT SHIFTS BY ONE BIT (DOUBLES) A MULTIBYTE DATA
        ;=  HL=A(DATA)   A,PSW,C=CORRUPTED
LEFT:   CALL LDC2     ; GET LENGTH,' CLEAR CARRY
        PUSH H
L006:   INX H
        MOV A,M
        RAL
        MOV M,A
        DCR C
        JNZ L006
        JC L004       ; O/FLOW?
        POP H
        RET

        ;* RIGHT RIGHT SHIFTS BY ONE BIT (HALVES) A MULTIBYTE DATA
        ;=  HL=A(DATA)  : A,PSW,BC=CORRUPTED
RIGHT:  CALL LDC2
        MVI B,0
        DAD B         ; GET A(HIGH ORDER BYTE)
        MOV A,M
        RAR
        MOV M,A
        MOV B,A       ; SAVE NEW TOP BYTE
L008:   DCX H
        DCR C
        JZ L009
        MOV A,M
        RAR
        MOV M,A
        JMP L008
L009:   DCR B         ; IS IT ZERO?
        RP            ; NO
        PUSH PSW      ; YES- SAVE CARRY = LOST BIT
        CALL DCRM     ;  DECREMENT LENGTH
        POP PSW
        RET

        ;* MOOV MOVES A MULTIBYTE DATA FROM DE TO HL
        ;=  DE=FROM ,  HL=TO  : A,PSW,B=CORRUPTED
MOOV:   CALL LDB1     ; GET LENGTH
        MOV M,A       ; MOVE ANY ZERO LENGTH
        RZ            ; WAS IT ZERO?
        LDAX D        ; NO- MOVE-
        MOV M,A       ; PROPER LENGTH
        PUSH D
        PUSH H
L011:   INX D
        INX H
        LDAX D        ; MOVE-
        MOV M,A       ;- BYTE
        DCR B
        JNZ L011
        POP H
        POP D
        RET

        ;* PARE COMPARES MULTIBYTE DATA DE TO HL
        ;***  DE>HL -Z-C  : DE=HL Z,-C  : DE<HL -Z,C
        ;== DE=A(FROM), HL=A(WITH)  : A,PSW,BC=CORRUPTED
PARE:   MOV A,M
        RAL
        CMC           ; SET SIGN(HL) TRUE
        RAR
        MOV B,A
        LDAX D
        RAL
        CMC           ; SET SIGN(DE) TRUE
        RAR
        CMP B
        RNZ           ; NON EQUAL LENGTHS
        CALL LDC1
        PUSH H
        PUSH D
        MVI B,0
        DAD B         ; GET A(M.S.BYTE HL)
        XCHG
        DAD B         ; GET A(M.S.BYTE DE)
        XCHG
L014:   LDAX D
        CMP M         ; COMPARE BYTES
        JNZ L015
        DCX H         ; EQUAL SO TRY-
        DCX D         ;- NEXT LOWER
        DCR C         ;- BYTES
        JNZ L014
L015:   POP D
        POP H
        RET

        ;==  DE=AUGEND ,  HL=ADDEND ,  DE SUM
        ;= A,PSW,BC=CORRUPTED
        ;* ADD1 ADDS MULTIBYTE DATA DE TO HL
ADD1:   CALL LDC2
        PUSH D
        PUSH H
        CALL LDB1
        SUB C         ; COMPARE LENGTHS
        XCHG
        JNC L017
        CMA
        ADC M         ; INCREASE AUGENDS
        MOV M,A
        XRA A
        ADD B
        JZ L021       ; AUGEND=ZERO
L017:   INX H
        INX D
        LDAX D
```

```
        ADC M
        MOV M,A
        DCR B
        JZ L022         ; AUGEND EXHAUSTED
        DCR C
        JNZ L017
L018:   INX H           ; ADDEND EXHAUSTED
        MOV A,M
        ADC C
        MOV M,A
        DCR B
        JNZ L018
        JNC L019
L020:   INX H           ; O/FLOW
        MVI M,1         ; EXTEND DATA
        POP D
        POP H
        CALL INRM
        XCHG
        RET
L021:   INX H
        INX D
        LDAX D
        ADC B
        MOV M,A
L022:   DCR C
        JNZ L021
        JC L020
L019:   POP H
        POP D
        RET

        ;*  SUB1 SUBTRACTS TWO MULTIBYTE DATA HL FROM DE
        ;= DE=MINUEND ,HL=SUBTRAHEND ,DE=RESULT : A,PSW,BC=CORRUPTED
SUB1:   CALL LDC2
        PUSH H
        PUSH D
        CALL LDB1
        SUB C
        JNC L024
        XCHG
        CMA
        ADC M           ; INCREASE MINUENDS LENGTH (PRESERVE SIGN)
        MOV M,A
        XRA A
        ADD B
        XCHG
        JZ L025         ; MINUEND=0
L024:   INX H
        INX D
        LDAX D
        SBB M
        STAX D
        DCR C
        JZ L027         ; SUBTRAHEND EXHAUSTED
        DCR B
        JNZ L024        ; MINUEND EXHAUSTED
L025:   INX H
        INX D
        MOV A,B
        SBB M
        STAX D
        DCR C
        JNZ L025
LZZZ:   POP H           ; COMPLEMENT(2'S COMP) THE -VE RESULT
        PUSH H
        PUSH D
        MOV A,M
        RAL
        CMC             ; CHANGE SIGN
        RAR
        MOV M,A
        ANI 7FH
        MOV C,A         ; GET LENGTH
        STC
LXZZ:   INX H
        MOV A,M
        CMA             ; COMPLEMENT BYTES
        ADC B
        MOV M,A
        DCR C
        JNZ LXZZ
        POP D
        JMP LZXX
L026:   INX D
        LDAX D
        SBB C
        STAX D
L027:   DCR B
        JNZ L026
        JC LZZZ
LZXX:   POP H
L028:   LDAX D
        CMP C           ; ANY REDUCTION IN PRECISION?
        JNZ L030        ; NO
        DCX D
L029:   CALL DCRM
        JNZ L028
L030:   XCHG
        POP H
        RET

        ;** GENERAL PURPOSE ADD/SUBTRACT ROUTINE FOR BOTH +VE & -VE NOS
        ;== DE=AUGEND/MINUEND : HL=ADDEND/SUBTRAHEND : DE=RESULT
        ;= BC,PSW,A=CORRUPTED
AD1:    XRA A           ; CLEAR CARRY
        JMP L050
SB1:    STC             ; SET CARRY
L050:   PUSH PSW
        LDAX D
        XRA M           ; DIFFERING SIGNS ?
        JM L051
        POP PSW         ; NO
        JC L052
L053:   CALL ADD1
        RET
L051:   POP PSW         ; YES
        JC L053
L052:   CALL SUB1
        RET

        ;** GENERAL PURPOSE MULTIPLY +VE & -VE
        ;== DE=MULTIPLICAND : HL=MULTIPLIER : DE=RESULT
        ;= A,PSW,BC=CORRUPTED
MULT:   CALL LDB2       ; RETURN IF ZERO
        PUSH H
        LDAX D
        XRA M
        PUSH PSW        ; SAVE SIGNS
        PUSH D
        PUSH H
        LXI H,T1
        CALL MOOV       ; MOVE MULTIPLICAND
        XRA A
        STAX D          ; CLEAR RESULT
        POP D
        LXI H,T2
        CALL MOOV       ; MOVE MULTIPLIER
        POP D
L072:   LXI H,T2
        MOV A,M         ; MULTIPLICATION STARTS
        ORA A
        JZ L070         ; END?
        CALL RIGHT
        LXI H,T1
        JNC L071        ; LEAST SIGNIFICANT BIT NOT SET
        CALL ADD1
L071:   CALL LEFT
        JMP L072
```

```
L070: POP PSW
      POP H
      RP          ; DON'T CHANGE SIGN
      LDAX D
      XRI 80H     ; CHANGE SIGN
      STAX D
      RET


      ;** GENERAL PURPOSE DIVIDE, DIVIDE & ROUND (RESULT=NEAREST WHOLE INTEGER)
      ;* AND MODULO ROUTINE FOR +VE & -VE NO.S (NOTE -32 MOD 3 =2)
      ;== DE=DIVIDEND ; HL=DIVISOR ; DE RESULT(DIV)/REMAINDER(MOD)
      ; = A,PSW,BC=CORRUPTED
      ;*=*=*=*  CARRY SET ON EXIT INDICATES DIVISION IMPOSSIBLE
MOD:  XRA A
      ADI 80H
      JMP L081    ; CLEAR CARRY, SET SIGN
DIVR: XRA A
      STC
      JMP L081    ;SET CARRY
DIV:  XRA A       ; CLEAR CARRY
L081: PUSH H
      PUSH PSW
      LDAX D
      XRA M
      RLC
      JNC L083
      POP PSW
      INR A       ; PUT ANY DIFFERENCE IN SIGNS INTO BIT0
      PUSH PSW    ; SAVE
L083: CALL LDC1
      JZ RETN     ; RETURN DIVISOR ZERO
      CALL LDB1
      SUB C
      JM RETN     ; RETURN DIVISOR>DIVIDEND
      PUSH D
      INR A
      MOV B,A
      MOV C,A
      PUSH B      ; SAVE LENGTH DIFFERENCE
      LDAX D
      ANI 7FH     ; CLEAR SIGN DIVIDEND
      STAX D
      XCHG
      LXI H,T1
      MOV A,M
      ANI 7FH     ; CLEAR SIGN OF PARTIAL DIVISOR
      MOV M,A
      CALL MOOV   ; MOVE DIVISOR
      XRA A
      STA T2
      POP D
L082: CALL PSHL
      DCR E
      JNZ L082
      MOV C,A
      MOV B,D
L090: POP D       ; DIVISION STARTS
      PUSH D
      PUSH B
      LXI H,T1
      CALL PARE   ; PARTIAL DIVIDEND>= PARTIAL DIVISOR?
      JC L092     ; NO
      CALL SUB1   ; YES- SUBTRACT
      LXI H,T2
      CALL INCR
L092: LXI H,T1    ; RIGHT SHIFT PARTIAL DIVISOR
      CALL RIGHT
      POP B
      DCR C       ; REDUCE COUNT
      JP L093
      MVI C,7
      DCR B
      JM L094     ; END
L093: PUSH B
      LXI H,T2
      CALL LEFT   ; LEFT SHIFT PARTIAL RESULT
      POP B
      JMP L090
L094: POP D       ; A(REM)
      POP PSW
      JM L098     ; MODULO
      JNC L096    ; NOT ROUND
      PUSH PSW
      CALL PARE   ; PARTIAL DIVISOR/2 < REM ?
      JC L095
      JZ L095
      LXI H,T2    ; YES
      CALL INCR   ; INCREMENT RESULT
L095: POP PSW
L096: XCHG
      LXI D,T2    ;
      ANI 1
      JZ L097     ; NO CHANGE OF SIGN
      LDAX D
      ORI 80H
      STAX D
L097: CALL MOOV   ; MOVE RESULT TO A(DE)
      XCHG
L098: XRA A
      POP H
      RET

RETN: POP PSW
      STC
      POP H
      RET


      ; SQUARE ROOT ROUTINE FOR +VE NO.S
      ;== DE =ARGUMENT ; DE= RESULT
      ; = A,PSW,BC,HL=CORRUPTED
SQRTR: ORA  A     ; CLEAR CARRY FOR ROUND
      JMP L103
SQRT: STC         ; SET CARRY FOR NOT ROUND
L103: PUSH PSW    ; SAVE OPERATION TYPE
      LDAX D
      ORA A
      STC
      JP L104
      POP B
      RET         ; RETURN UNDONE -VE OR ZERO
L104: MVI C,1
      LXI H,T2
      MOV M,C
      INX H
      MOV M,C     ; INITIALISE T2 TO 1
      PUSH D
      MOV D,A
      MVI A,0
      DCX H
L100: CALL PSHL
      DCR D       ; MAKE T2 >N (T2 MUST BE A SQUARE NUMBER)
      JNZ L100
      LXI D,T1
      STAX D      ; CLEAR T1
L101: CALL ADD1   ; T1=T1+T2
      XCHG
      POP D
      CALL PARE   ; N>= T1?
      PUSH PSW
      CNC SUB1    ; YES- N=N-T1
      POP PSW
      PUSH D
      XCHG
      LXI H,T2
      PUSH PSW
      CNC ADD1    ; YES- T1=T1+T2
```

```
        POP PSW
        CC SUB1      ; NO T1=T1-T2
        XCHG
        CALL RIGHT   ; T1=T1/2
        XCHG
        CALL RIGHT
        CALL RIGHT   ; T2=T2/4
        MOV A,M
        ORA A        ; T2=0 ?
        JNZ L101     ; NO
        POP H
        POP PSW      ; WAS IT ROUND?
        JC L102      ; NO
        CALL PARE    ; N>T1 ?
        JNC L102
        XCHG
        CALL INCR    ; YES SO T1=T1+1
        XCHG
L102:   CALL MOOV    ; PUT RESULT
        XCHG
        ORA A        ; CLEAR CARRY FOR GOOD RESULT
        RET

        ORG 900H
        MVI A,10
        STA 711H     ; INITIALISE WORK AREAS
        MVI A,1
        STA 710H
        STA 720H
L0:     XRA A
        STA B00H     ; RESET WORK LENGTHS
        STA 500H
        STA 600H
        CALL 3CC8H   ; O/P CRLF
L1:     CALL 3F6EH   ; GET CHAR
        CPI 30H
        JC ERR1      ; <0
        CPI 3AH
        JNC ERR1     ; >9
        STA 700H
L9:     LXI D B00H
        LXI H,710H
        CALL 2C0H    ; MULTIPLY T1 BY 10
        LDA 700H     ; GET DIGIT
        SUI 30H
        JZ L1        ; DON'T ADD ZERO
        STA 721H     ; PUT IT
        LXI H,720H
        CALL 2A5H    ; ADD DIGIT TO 10*T1
        JMP L1
ERR1:   CPI 2BH
        JZ L2        ; +
        CPI 2DH
        JZ L2        ; -
        CPI 2AH
        JZ L2        ; *
        CPI 2FH
        JZ L2        ; /
        CPI 25H
        JZ L2        ; % DIVIDE AND ROUND
        CPI 3DH
        JZ L3        ; =
        CPI 23H      ; # =SQRT
        JZ L10
        CPI 40H      ; @ = SQRT AND ROUND
        JZ L11
ERR3:   MVI C,3FH
        CALL 3C50H   ; O/P '?'
        JMP L0
L2:     STA 701H     ; SAVE OPERATION
        XRA A
        STA 500H
        STA 600H
L4:     CALL 3F6EH   ; GET CHAR
        CPI 30H
        JC ERR2      ; <0
        CPI 3AH
        JNC ERR2     ; >9
        STA 700H     ; SAVE DIGIT
        LXI D,500H
        LXI H,710H
        CALL 2C0H    ; MULTIPLY T2 BY 10
        LDA 700H
        SBI 30H      ; GET NUMBER
        JZ L4        ; DON'T ADD PSEUDO ZERO
        STA 721H
        LXI H,720H
        CALL 2A5H    ; ADD DIGIT TI 10*T2
        JMP L4
ERR2:   CPI 3DH
        JNZ ERR3     ; NOT =
        LXI D,800H
        LXI H,500H
        LDA 701H
        CPI 2BH
        CZ 2A5H      ; CALL ADD
        LDA 701H
        CPI 2DH
        CZ 2A9H      ; CALL SUBTRACT
        LDA 701H
        CPI 2AH
        CZ 2C0H      ; CALL MULTIPLY
        LDA 701H
        CPI 25H
        CZ  301H     ; CALL DIVIDE WITH ROUND
        LDA 701H
        CPI 2FH
        CZ  306H     ; CALL DIVIDE NO ROUND
L3:     LXI H,800H
        LXI D,B00H
        CALL 1C7H    ; MOVE IT
        MOV A,M
        ORA A        ; IS IT -VE?
        JP L8
        ANI 7FH
        MOV M,A
        MVI C,2DH
        CALL 3C50H   ; O/P MINUS SIGN
L8:     LXI D,800H
        LDAX D
        ORA A
        JZ OUT       ; CONVERSION FINISHED
        LXI H,710H
        CALL 2FBH    ; CALL MOD 10
        LDA 801H
        LXI H,600H
        CALL 132H    ; PUSH BYTE
        LXI H,800H
        LXI D,1200H
        CALL 1C7H    ; MOVE RESULT BACK TO T1
        JMP L8
OUT:    LXI H,600H
        MOV A,M
        ORA A        ; CHECK NON ZERO
        JNZ OUT1
        MVI C,30H
        CALL 3C50H   ; OUTPUT ASCII ZERO
        JMP L5
OUT1:   LXI H,600H
        MOV A,M
        ORA A
        JZ L5        ; OUTPUT FINISHED
        CALL 14EH    ; GET BYTE
        ADI 30H
        MOV C,A
        CALL 3C50H   ; OUTPUT IT
        JMP OUT1
L5      CALL 3CC8H   ; O/P CRLF
        CALL 3F6EH   ; GET CHAR
        CPI 30H
```

AN ALGORITHM FOR COMPUTING SQUARE ROOTS

Dr. Dobbs: Dec. 21, 1976

I have only recently dug into your exercises and teeth moving and am impressed.

I do have one negative observation (a complaint). Super Classy Text Editor "A" for 8080's is hunky dory (and deserves space, no dung!) but is impossible to translate for a TMS 9900. This is because machine to machine conversions are less than humane. Since your readers often remark with pride about the plethora of computer systems in their club they might be interested in human understandable techniques in addition to machine readable stuff.

A fine example of what I would like to see more of in your journal is on the line of the algorithms such as the somewhat inactive techniques and pracniques [*Dr. Dobb's* No. 1, Vol. 1]. If any of your readers think that they have anything to submit for that forum I urge them to do so. To show that I practice what I preach I am enclosing this algorithm (first shown to me by Dr. Will Brown, Western Washington State College in 1967) for extracting square roots.

Thanks,

James A. Hinds
7240 Davenport Rd, No. 101
Goleta, CA 93017

ALGORITHMS FOR TAKING DECIMAL SQUARE ROOTS

0 is input quantity (all arithmetic is base 10)
Result in N

S1: [initialize and scale 0] N←1, fraction digits ← 0.
R←"first digit(s) of 0". [This is easy to do with a computer but hard to express] If greatest integer ($\log_{10}$ (0) is even
then R←first digit of 0
else R←first two digits of 0
[This is exactly what you do when taking square roots by hand]
GO TO S3.

S2: SET N← N–2.

S3: SET R← R–N, IF R$\geq$ 0 THEN GOTO S2

S4: R← R + N, N← N–1,
If (fraction digits > "ENOUGH") or (R=0 and 0 exhausted) then N← N ÷ Z [no remainder possible] STOP;

S5: SET N← N * 10 + 1, R← R * 100 + "NEXT 2 DIGITS OF 0"
If "THESE 2 DIGITS OF 0 ARE IN THE FRACTION"
then set fraction digits ← fraction digits + 1
GOTO S3

A WORKED-OUT EXAMPLE

Square root of 121 ie 0 = 1'21

Step 1 R← 1; N← 1; fractional prec is none ← 0.

Step 3 R← R–N=1–1=0 (R not less than 0)
Step 2 N← N+2=1+2=3
Step 3 R← R–N=0–3=–3 (R less than 0)

Step 4 R← R+N=(–3)+3=0
N← N–1=3–1=2
Termination test fails

Step 5 N← N * 10+1=(2) * 10+1=21
R=R * 100+Next Digits of 0=(0) * 100+21

Step 3 R← R–N=(21)–(21)=0 (R not less than 0)
Step 2 N← N+2=21+2=23
Step 3 R← R–N=0–23=–23 (R less than 0)

Step 4 R← R+N=(–23)+23=0
N← N–1=23–1=22
Termination test succeeds (R=0 and 0 exhausted)
There when result=N/2 with no fraction digits
$= \frac{22}{2} = 11$

```
TC ERR1
CPI 3AH
JNC ERR1
STA 700H
XRA A
STA B00H        ; CLEAR WORKING REGISTER
JMP L9
L10:  MVI C,3DH
      CALL 3C50H      ; O/P '='
      LXI D,B00H
      CALL 3A7H       ; SQRT NO ROUND
      JMP L3
L11:  MVI C,3DH
      CALL 3C50H      ; O/P '='
      LXI D,B00H
      CALL 3A3H       ; SQRT WITH ROUND
      JMP L3
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      NOP
      END
```

```
LISTING OF ASSEMBLED TEST PROGRAM

:1000000003E0A3211073E013210073220007AF320093
:10010000B320005320006CDC83CCD6E3FFE30D0A0A
:1002000470FE3D02470932000711000B2110079D
:10030000CDC0023A000706306A1A0932210721205A
:10040000C7CDA502C31A09FE2BCA7709FE2DCA7767
:10050000FE2ACA7709FE2FCA7709FE25CA77093B
:1006000FE3DCAE109FE23CA480AFE40CA5E0A0EE5
:10070007FCD503CC300A9320107AF3200053200B4
:10080006CD6E3FFE30DAAE09FE3A02AE0932002A5
:1009000007110005211007CDC0023A0007DE30CA5A
:100A0000107932210721200700A502C38109FE3D1F
:100B0000C26F0911000E2100053A0107FE2BCA5DF
:100C0000023A0107FE2DCA9023A0107FE2ACCC04B
:100D0000023A0107FE25CC01033A0107FE2FCC069F
:100E0000321000811000BCDC7017EB7F2F709E61D
:100F0007F770E2DCD503C11000810A7CA1A0A2174
:10100000107CDFB023A0107821000603D32012107A
:1011000811001200CDC701C3F709210006CEB7CA2D
:10120002E0ACD4E01C6304FCD503CC31A0ACDC058
:10130003CCD6E3FFE30DA4709FE3A02470932001C
:10140007AF32000BC32A030E3DCD503C11000BCDFD
:10150000CDA707C3E1090E3DCD503C11000ECDA342
:10160003C3E109000000000000000000000000000006
```

AN IMSAI GLITCH FIX

Fred Pfafman
353 San Roque Drive
Escondido, California 92025
(714) 747-3630

The IMSAI RAM-4A memory board may generate a minor annoyance on your front panel. When the address is selected for 0000 thru 0FFF the memory protect LED is turned on while performing an input instruction. This problem does not affect the system operation and will be seen on an IMSAI only if the front panel memory protect modification has been incorperated.

Correction of the problem may be accomplished through an unused gate, three jumpers and one circuit cut on the board. An unused section of AND gate C1 is inserted between the output of C4 pin 8 and BDENA BAR signal. The other input of C1 is connected to C4 pin 2. The circuitry to pin 8 of C4 should be cut. This will AND the address decoder output with the NOT input signal and disable the output driver for pin 69, which is the memory protect indicator.

The partial schematic shows how the circuit is modified and will aid in following the above description. If you have any questions please get in touch with me. I am interested in your ideas also.

PERSONAL COMPUTERS & COMMUNICATIONS

Dear Sirs or Madams as the case may be: Dec. 17, 1976

With regard to Lee Felsenstein's idea for 10-watt FM broadcast stations as one-way data buses: I don't think this will work in this form. A 10-watt station has a reliable *in each area* is limited by the FCC. This is for non-commercial as well as commercial. One could get into a situation where the only available channel is in the commercial part of the band, and engineering and program requirements for that frequency are much stricter.

I would give a serious look at two-way data communications in the 1215-mHz amateur band, or in the business communication segments of the 470-890 mHz TV band, or one of the educational closed-circuit TV channels. There's an outfit in the Bay Area that makes reasonably priced ham gear for this band. A good article appeared in the December 1975 issue of *QST*. Similar equipment can be developed as add-on gear to 144 or 432 mHz ham gear or public-service surplus or even police-band scanner receivers. Anyone who can build a Data Handler and have it work with minimum de-bugging can get a Technician Class ham license with very little trouble. There is a nationwide push on not only to get more hams licensed but to expand usage of the amateur bands between 1215 and 10,500 mHz. Since the engineering restrictions are minimal the service area problem is also minimal. You could pilot and debug the system on the ham bands and then (having found appropriate technical standards, move on to a co-op or non-profit installation on a commercial channel. One reason for staying above 1215 mHz is there's more space to paly with, and less chance of interfering with someone else on an adjacent channel with high-baud-rate transmissions.

Sincerely,
Mike Fern 1046 So. Westlake, Apt. 1
Los Angeles, CA 90006

PUSH & POP THE PSW IN P.T.'s ASSEMBLER

Dear D.D.J.,

I was reading through Vol. I, No. 8 and noticed a complaint in the article by Jef Raskin concerning Processor Tack's SYS8:. The missing instructions PUSH PSW and POP PSW caused me to go into a rage too. So I dug into the assembler and discovered that the codes for the double register operations like INX, DAD, etc. assemble the same way as the PUSH and POP instructions. The PSW register pair has the same code as M. Therefore PUSH M and POP M will assemble properly for PUSH PSW and POP PSW.

Raymond F. Jurgens 1421 Curran St.
La Canada, CA 91011

COMMENTS ON LANGUAGES FOR MICROS

Dear Sir: October 12, 1976

I wish to comment on a few things mentioned in recent issues. First, the BASIC controversy: BASIC, in its extended versions is indeed a powerful language. However, for us micro types the extended ones are oft promised and seldom delivered. (ALTAIR is the exception but they have other problems.) Besides even the most powerful implementations I have heard of still retain two of BASIC's most insidious drawbacks: line numbers and NO LOCAL VARIABLES. Line numbers can be lived with, having all variables global is a great pain, as is its concomitant no subroutine parameters. The problem is even worse when, as in most cases, there are only 260 (or 26) variable names.

On the other hand, it is easy to use, though I think ease of learning is overrated. Once you have learned to think in the appropriate manner, the whole FORTRAN-ALGOL-PL/whatever bunch of languages is very similar and easy to move between.

My second comment is that the small memory assumption is becoming questionable. Admittedly, the processor and terminal still represent a big investment for almost all of us. Once you've got that though, the incremental cost of another couple K of memory is no longer that great: $25 for a board and control chips and $15/K byte for 2102's is within reach of most of us. Six and eight K software is certainly reasonable, and sixteen should be reasonable soon.

For similar reasons I don't think much of the suggestions you occasionally publish for timeshared micros: share mass storage, and even power supplies if you're hard up for cash but it's not worth the money, or the performance penalty to timeshare machines this small and cheap.

End of diatribe,
Frank Richards
414 Bradford Way
Norristown, PA 19401

1802 BASIC?

Sirs: Feb. 22, 1977

Do you have, or know of anyone that has, a BASIC system for the RCA 1802 microprocessor? I have a homemade job that could use BASIC. If you have such a system could you inform me of the costs and memory requirements. I would prefer to get a listing of the program. I don't have a paper tape reader and my tape cassette system is non-standard.

Sincerely,
R. Alan Parker 9660D Barrel House Rd.
Laurel, MD 20810

8080 DISASSEMBLER & TRACER . . . FOR $250 EACH

8080 DISASSEMBLER

Designed to run on an 8080 based machine, this program disassembles 8080 object code, producting a "source" listing in format identical to that produced by an Intel Assembler, as well as an ASCII listing of the object code and a symbol cross reference listing. Also produced is source file on floppy disk or punched paper tape which may be input to an assembler with little or no editing. Up to 10 known data areas may be specified which will be treated as DB statements.

8080 TRACE

Designed as a teaching and trouble-shooting aid, this program resides in memory with the subject program, which is executed under trace control. The machine condition is displayed after each subject instruction is executed. Display includes instruction address, instruction in machine code (hexidecimal), and mnemonic, contents of all registers, and latest stack entry. Display may be reduced to only instructions causing branch operations. Modes of operation (single step, branch, continuous, etc.) may be changed during execution by operator.

Both the CYPERGRAFIX Disassembler and 8080 Trace programs are available on iCOM Floppy Disk or punched paper tape in Intel Hex-ASCII object format at the cost of $250 each, or $400 for both. Programs are specialized to run on user's equipment and updates resulting from continuous program improvements are automatically sent to purchasers of CYBERGRAFIX Software.

For further information, contact Sheila Clarke at (213) 243-5179, or write CYBERGRAFIX, P.O. Box 430, Glendale, Calif., 91206.

AN 8080 DISASSEMBLER WRITTEN IN MITS 3.2 BASIC

by Jef Raskin 77-Jan-11

I was pleased to see this program, since it enables one to disassemble all kinds of obscure software. In particular, MITS BASIC itself. The author of this well-written program is rather a perfectionist, and wishes to remain anonymous. I think she or he is altogether too modest – it seems to be well organized and has been tested carefully. The program is designed for a CRT such as the Lear-Seigler ADM-3. You may have to modify the well-marked CRT terminal specific control characters (such as "clear screen" commands) to make it operate on your system. If you are stuck with a teletype, still more modification will be necessary, but the program is quite clear, and readily adapted.

There are some especially interesting features incorporated in this dissassembler. To use it you merely type an (absolute) hexadecimal address. Illegal addresses are not accepted, nor are those above the first 32K. If you type an illegal character, it is merely echoed and then erased, allowing you to put the correct character in its place. This feature should be incorporated in all interactive programs. After the first address is typed, merely pressing the space bar will dissassemble successive lines. For continuous listing replace line 230 with GO TO 500. At any time entry of an address will cause the disassembler to start disassembling at the new address. For conditional and unconditional branches and subroutine calling instructions another option is available. Typing the letter "J" results in the word "JUMP" appearing, and the disassembler will disassemble the instruction referred to in the branch or call rather than the instruction that immediately follows it in memory.

I cannot supply this program in machine readable form for less than an exorbitant fee, since it is a nuisance to have to load it into the computer and then have to wait while a cassette is written. I do not have a paper tape punch. Even if I did, or had time to write lots of cassettes, they probably would be to a different standard or in a different format than yours. Tough darts.

```
100 GOSUB 9000:REM INITIALIZE ARRAYS, CONSTANTS
105 PRINT CS$:REM CLEAR SCREEN
110 PRINT "    8 0 8 0      D I S A S S E M B L E R"
120 PRINT
140 JF=FALSE:REM JUMP FLAG
150 PC=0
160 PF=TRUE
200 REM *** MAIN LOOP ***
210 PRINT " >".
220 GOSUB 5600:REM GET A CHARACTER AND ECHO IT
230 IF C$=" " GOTO 500
235 IF JF THEN IF C$="J" GOTO 450
240 REM LOOK FOR AN ADDRESS (MUST START WITH 0-7)
250 IF C$>="0" AND C$<="7" GOTO 300
255 IF C$=CR$ THEN PRINT:GOTO 210
260 IF PR THEN GOSUB 6200:REM ERASE THE CHARACTER
270 GOTO 220
300 T=C-ASC("0"):N=1
310 GOSUB 5600:REM GET A CHARACTER
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$+EL$;:GOTO 200
350 IF PR THEN GOSUB 6200:REM ERASE THE CHARACTER
360 GOTO 310
400 T=D+16*T
410 N=N+1
420 IF N<4 GOTO 310
430 PC=T
440 GOTO 1020
450 REM JUMP TO ADDRESS IN LAST CALL OR JUMP INSTR
460 PRINT "UMP":PRINT " >".
470 PC=X
480 GOTO 1000
500 REM SPACE MEANS DO THE NEXT INSTRUCTION
```

```
510 PRINT BS$;:REM  GET RID OF THE SPACE
520 IF PC<=32765 GOTO 1000
530 PRINT "ADDRESS TOO HIGH"
540 GOTO 200
1000 REM  PRINT THE ADDRESS AND FETCH AN INSTRUCTION
1010 X=PC:GOSUB 5450:REM PRINT ADDRESS
1020 PRINT ":  ";
1030 GOSUB 5000:OP=X:REM FETCH OPCODE
1040 GOSUB 2000:REM DECODE IT!
1045 PRINT
1050 GOTO 200
2000 REM  DECODE "OP" AS AN INSTRUCTION AND PRINT ITS MNEMONI
C
2010 REM
2020 JF=FALSE
2030 REM  GENERAL NOTE ABOUT "ON N ..." STATEMENTS:
2040 REM   IF N IS 0, CONTROL DROPS THROUGH TO THE
2050 REM   STATEMENT FOLLOWING THE "ON" STATEMENT.
2055 REM  L L M M M K K K   EXTRACT THE FIELDS SHOWN
2060 L=INT(OP/BIT(6))          :REM BITS 6-7
2062 M=INT(OP/BIT(3)) AND 7    :REM BITS 3-5
2064 K=OP AND 7                :REM BITS 0-2
2070 ON L GOTO 3000,3200,3500
2099 REM
2100 REM  0 0 M M M K K K
2110 REM  DISPATCH ON K (BITS 0-2)
2120 ON K GOTO 2200,2400,2500,2600,2600,2700,2800
2129 REM
2130 REM  0 0 M M M 0 0 0
2140 IF M=0 THEN S$="NOP"
2150 IF M>0 THEN S$="- -":REM UNIMPLEMENTED INSTRUCTION
2160 GOSUB 6000:RETURN
2199 REM
2200 REM  0 0 M M M 0 0 1
2210 K=M/BIT(1):REM K IS NOW BITS 4-5 OF OP
2220 IF OP AND BIT(3) GOTO 2300
2230 REM  0 0 K K 0 0 0 1
2240 S$="LXI ":GOSUB 6000
2250 S$=RP$(K):GOSUB 6000
2260 S$=", ":GOSUB 6000
2270 GOSUB 5400:REM FETCH NEXT 2 BYTES
2280 RETURN
2300 REM  0 0 K K 1 0 0 1
2310 S$="DAD ":GOSUB 6000
2320 S$=RP$(K):GOSUB 6000
2330 RETURN
2399 REM
2400 REM  0 0 M M M 0 1 0
2410 S$=O2$(M):GOSUB 6000
2420 IF M<4 THEN RETURN:REM STAX OR LDAX, 1 BYTE
2430 REM SHLD,LHLD,STA,LDA - FETCH NEXT 2 BYTES
2440 GOSUB 6100
2450 GOSUB 5400
2460 RETURN
2499 REM
2500 REM  0 0 M M M 0 1 1
2510 K=INT(M/BIT(1)):REM K IS BITS 4-5 OF OP
2520 M=M AND BIT(0):REM M IS NOW BIT 3 OF OP
2530 IF M=0 THEN S$="INX "
2540 IF M=1 THEN S$="DCX "
2550 GOSUB 6000
2560 S$=RP$(K):GOSUB 6000
2570 RETURN
2599 REM
2600 REM  0 0 M M M 1 0 0   INR
2602 REM  0 0 M M M 1 0 1   DCR
2610 IF K=4 THEN S$="INR "
2620 IF K=5 THEN S$="DCR "
2630 GOSUB 6000
2640 S$=REG$(M):GOSUB 6000
```

```
2650 RETURN
2699 REM
2700 REM  0 0 M M M 1 1 0   MVI
2710 S$="MVI ":GOSUB 6000
2720 S$=REG$(M):GOSUB 6000
2730 S$=", ":GOSUB 6000
2740 GOSUB 5200
2750 RETURN
2799 REM
2800 REM  0 0 M M M 1 1 1
2810 S$=O7$(M):GOSUB 6000
2820 RETURN
3000 REM  0 1 M M M K K K    MOV INSTRUCTION
3010 IF M=6 AND K=6 GOTO 3150:REM MOV M,M IS REALLY HLT
3020 S$="MOV ":GOSUB 6000
3030 S$=REG$(M):GOSUB 6000:REM DESTINATION REGISTER
3040 S$=", ":GOSUB 6000
3050 S$=REG$(K):GOSUB 6000:REM SOURCE REGISTER
3060 RETURN
3150 REM  0 1 1 1 0 1 1 0    HLT INSTRUCTION
3160 S$="HLT":GOSUB 6000:RETURN
3199 REM
3200 REM  1 0 M M M K K K    ACCUMULATOR ARITHMETIC INSTR.
3210 S$=AI$(M):GOSUB 6000: REM M DETERMINES OPERATION
3220 GOSUB 6100
3230 S$=REG$(K):GOSUB 6000: REM K DETERMINES OPERAND
3240 RETURN
3499 REM
3500 REM  1 1 M M M K K K
3510 ON K GOTO 3560,3600,3800,3620,4000,4100,4200
3520 REM  1 1 M M M 0 0 0   RETURN CONDITIONAL
3530 S$="R":GOSUB 6000
3540 S$=CC$(M):GOSUB 6000:REM M IS THE CONDITION
3550 RETURN
3560 REM  1 1 M M M 0 0 1
3570 S$=C1$(M):GOSUB 6000
3580 RETURN
3599 REM
3600 REM  1 1 M M M 0 1 0   JUMP CONDITIONAL
3610 S$="J":GOTO 3650
3620 REM  1 1 M M M 1 0 0   CALL CONDITIONAL
3630 S$="C"
3650 GOSUB 6000
3660 S$=CC$(M):GOSUB 6000 :REM M DETERMINES THE CONDITION
3680 GOSUB 6100
3690 GOSUB 5400:REM FETCH THE ADDRESS
3695 JF=TRUE
3700 RETURN
3800 REM  1 1 M M M 0 1 1
3810 S$=C3$(M):GOSUB 6000:REM PRINT OPCODE
3820 IF M=0 GOTO 3850:REM JMP
3830 IF M=2 OR M=3 GOTO 3900:REM  OUT,IN
3840 RETURN
3850 GOSUB 6100
3860 GOSUB 5400:REM FETCH THE ADDRESS
3865 JF=TRUE
3870 RETURN
3900 GOSUB 6100
3910 GOSUB 5200:REM FETCH THE DEVICE CODE
3920 RETURN
4000 REM  1 1 M M M 1 0 1
4010 S$=C5$(M):GOSUB 6000
4020 IF M<>1 THEN RETURN
4030 REM  CALL INSTRUCTION
4040 GOSUB 6100
4050 GOSUB 5400:REM FETCH THE ADDRESS
4055 JF=TRUE
4060 RETURN
```

```
4100 REM  1 1 M M M 1 1 0
4110 REM  IMMEDIATE ACCUMULATOR ARITHMETIC INSTRUCTIONS
4120 S$=C6$(M):GOSUB 6000:REM PRINT OPCODE
4130 S$=" ":GOSUB 6000
4140 GOSUB 5200:REM FETCH OPERAND
4150 RETURN
4200 REM  1 1 M M M 1 1 1   RST
4210 S$="RST ":GOSUB 6000
4220 X=8*M:GOSUB 5100:REM PRINT ADDRESS CALLED
4230 JF=TRUE
4240 RETURN
4999 REM
5000 REM  FETCH THE NEXT BYTE AND INCREMENT THE "PC"
5010 REM   THE BYTE IS RETURNED IN X
5020 REM   PC IS THE PC, OF COURSE
5030 X=PEEK(PC):PC=PC+1:RETURN
5099 REM
5100 REM  PRINT THE NUMBER X AS TWO HEXADECIMAL DIGITS
5110 PRINT HEX$(INT(X/16))·HEX$(X AND 15);:RETURN
5199 REM
5200 REM  FETCH THE NEXT BYTE AND PRINT IT IN HEX
5210 REM  THE NEW BYTE IS RETURNED IN X
5220 GOSUB 5000:GOSUB 5100:RETURN
5299 REM
5300 REM  FETCH THE NEXT TWO BYTES
5310 REM  RETURN THE FIRST IN XL, THE SECOND IN XH,
5320 REM   AND THE 16-BIT NUMBER (XH,XL)  IN X
5330 GOSUB 5000:XL=X
5340 GOSUB 5000:XH=X
5350 X=XL+256*XH:RETURN
5399 REM
5400 REM  FETCH THE NEXT TWO BYTES, AS ABOVE,
5410 REM  AND PRINT X AS A 4-DIGIT HEX NUMBER
5420 GOSUB 5300:GOSUB 5450
5430 RETURN
5450 REM  PRINT X AS A 4-DIGIT HEX NUMBER
5460 T=X:X=INT(T/256):GOSUB 5100
5470 X=T-256*X:GOSUB 5100
5480 X=T:RETURN
5499 REM
5500 REM  GET A SINGLE CHARACTER FROM THE KEYBOARD
5510 REM  RETURN THE NUMBER IN C, THE CHARACTER IN C$
5520 WAIT TS,KB:C=INP(TD) AND 127
5525 IF C>=97 AND C<=122 THEN C=C-32:REM CHANGE LOWER TO UPPE
R CASE
5530 C$=CHR$(C):RETURN
5599 REM
5600 REM  GET A CHARACTER FROM THE KEYBOARD,
5610 REM  ECHO IT ON THE SCREEN IF IT'S A PRINTING CHARACTER
5620 REM  PR IS TRUE IF THE CHARACTER WAS PRINTED
5630 GOSUB 5500
5640 PR=C>=32 AND C<=126
5650 IF PR THEN PRINT C$·
5660 RETURN
5999 REM
6000 REM  PRINT THE STRING IN S$
6010 REM  UNLESS THE PRINT FLAG, PF, IS FALSE
6020 IF PF THEN PRINT S$
6030 RETURN
6100 REM  PRINT A BLANK UNLESS PF IS FALSE
6110 IF PF THEN PRINT " "·
6120 RETURN
6200 REM  DELETE THE LAST CHAR. FROM THE SCREEN
6210 PRINT BS$;EL$·
6220 RETURN
8999 REM
9000 REM  INITIALIZATION
9010 REM
9020 REM  AN ARRAY OF HEX DIGITS
```

```
9030 DIM HEX$(15)
9040 FOR I=0 TO 15:READ HEX$(I):NEXT I
9050 DATA "0","1","2","3","4","5","6","7","8","9",A,B,C,D,E,F
9060 REM
9070 REM  AN ARRAY OF BITS (POWERS OF 2)
9080 DIM BIT(7)
9090 FOR I=0 TO 7:READ BIT(I):NEXT I
9100 DATA 1,2,4,8,16,32,64,128
9110 REM
9120 REM  THE NAMES OF THE 8-BIT REGISTERS
9130 DIM REG$(7)
9140 FOR I=0 TO 7:READ REG$(I):NEXT
9150 DATA B,C,D,E,H,L,M,A
9160 REM
9170 REM  THE NAMES OF THE REGISTER PAIRS
9180 DIM RP$(3)
9190 FOR I=0 TO 3:READ RP$(I):NEXT
9200 DATA B,D,H,SP
9210 REM
9220 REM  THE ACCUMULATOR ARITHMETIC INSTRUCTIONS
9230 DIM AI$(7)
9240 FOR I=0 TO 7:READ AI$(I):NEXT
9250 DATA ADD,ADC,SUB,SBB,ANA,XRA,ORA,CMP
9260 REM
9270 REM  IMMEDIATE INSTRUCTIONS (SERIES STARTING WITH C6)
9280 DIM C6$(7)
9290 FOR I=0 TO 7:READ C6$(I):NEXT
9300 DATA ADI,ACI,SUI,SBI,ANI,XRI,ORI,CPI
9310 REM
9320 REM  CONDITION CODES FOR JUMP, CALL, RETURN
9330 DIM CC$(7)
9340 FOR I=0 TO 7:READ CC$(I):NEXT
9350 DATA NZ,Z,NC,C,PO,PE,P,M
9360 REM
9370 REM  THE SERIES OF INSTRUCTIONS STARTING WITH C1
9380 REM  AND SPACED 8 APART
9390 DIM C1$(7)
9400 FOR I=0 TO 7:READ C1$(I):NEXT
9410 DATA POP B,RET,POP D,- -,POP H,PCHL,POP PSW,SPHL
9420 REM
9430 REM  THE INSTRUCTION SERIES STARTING WITH C3
9440 DIM C3$(7)
9450 FOR I=0 TO 7:READ C3$(I):NEXT
9460 DATA JMP,- -,OUT,IN,XTHL,XCHG,DI,EI
9470 REM
9480 REM  THE C5 SERIES
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,- -
9520 REM
9530 REM  THE SERIES STARTING WITH 07
9540 DIM O7$(7)
9550 FOR I=0 TO 7:READ O7$(I):NEXT
9560 DATA RLC,RRC,RAL,RAR,DAA,CMA,STC,CMC
9570 REM
9580 REM  THE SERIES STARTING WITH 02
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
9800 REM
9810 REM  SOME HANDY CONSTANTS
9820 TRUE=-1:FALSE=0
9830 REM  TERMINAL STATUS AND DATA PORTS, KEYBOARD AND SCREEN BITS
9840 TS=16:TD=17:KB=1:SB=2
9850 REM  VARIOUS FUNNY CONTROL CHARACTERS
9860 BS=8:BS$=CHR$(BS):REM  BACKSPACE
9870 CS=20:CS$=CHR$(CS):REM CLEAR SCREEN
9880 EL=12:EL$=CHR$(EL):REM ERASE LINE
9890 CR=13:CR$=CHR$(CR):REM CARRIAGE RETURN
9999 RETURN
OK
```

M6800 DISASSEMBLER

Gentlemen: January 20, 1977

Enclosed is a listing which I feel is worth publishing. It is a disassembler for M6800 machine code. It was written primarily to help document programs that were originally done in machine code.

The first page is a listing generator which calls the disassembler as a subroutine.

The disassembler subroutine itself stores enough information on exit that a cross-reference could be performed, if desired. Note that any invalid op code is treated as a one-byte instruction. This subroutine also serves well as the first half of a mnemonic-oriented memory-examine-and-change monitor.

The references to EXBUG are easily adapted to other monitors.

Yours truly,
Gordon Stallings, Research Development & Engineering Division, Applied Automation, Inc., Bartlesville, OK 74004

NEW ELECTRONIC ADAPTER MAKES A SYNTHESIZER POLYPHONIC

A new polyphonic synthesizer adapter has been introduced by Electronic Music Laboratories, Inc. of Vernon, Connecticut.

The EML POLY-BOX is a versatile one octave keyboard which which gives a monophonic synthesizer a unique polyphonic capability. Digital memory allows the programming of any chord or pitch combination into the POLY-BOX which can then be played through a symthesizer. Pitches follow the synthesizer oscillator through portamento, vibrato and keyboard transpositions and can be produced from one octave above to three octaves below the synthesizer oscillator. Manual tuning. variable phasing and a low pass filter help make it possible to play bass lines or harmony on the POLY-BOX keyboard while playing melody on the synthesizer.

The POLY-BOX is a durable, lightweight instrument using digital and analog technology. It weighs 15 pounds and is 19" wide by 12" deep by 5" tall. Output is adjustable from 0 to 5 volts nominal and the frequency range is 5.5Hz to 8324Hz. There are up to 26 pitches at semitone intervals simultaneously available.

The EML POLY-BOX can be added directly to any patchable synthesizer or to smaller instruments with minor modifications and is available directly from the factory and from selected dealers. The POLY-BOX sells for $475. For literature write to EML POLY-BOX, P.O. Box H, Vernon, Connecticut 06066.

For more information contact:
Electronic Music Labs, Inc., P'O' Box H, Vernon, CT 06066
(203) 875-0751

```
                    * THIS PROGRAM WILL DISASSEMBLE M6800 MACHINE CODES
                    * BETWEEN USER-DEFINED LIMITS.
                    *
                    * THIS PROGRAM USES SUBROUTINES IN EXBUG.
                    *
                    * DO NOT SET THE UPPER LIMIT = $FFFF.
                    *
                    * THE ADDRESSES OF THE CODE TO BE DISASSEMBLED MAY BE OFFSET.
                    * THE OFFSET IS ADDED TO THE ACTUAL ADDRESS BEFORE PRINTING.
                    * ALL BRANCHES ARE REFERENCED TO THE OFFSET ADDRESS.
                    *
                    * EXBUG SUBROUTINES AVAILABLE
                    * (EXBUG 1.1)
                    *
F000            EXBUG   EQU     $F000           EXBUG RESTART
F003            INEEE   EQU     $F003           BEGINNING AND END ADDRESS SUB.
F006            OUTHR   EQU     $F006           CONVERT BCD TO HEX
F009            OUTHL   EQU     $F009           CONVERT HEX LEFT TO BCD
F00C            OUTRH   EQU     $F00C           CONVERT HEX RIGHT TO BCD
F00F            ADDRIN  EQU     $F00F           ADDRESS INPUT (STORE @X)
F012            INCH    EQU     $F012           INPUT CHARACTER
F015            INCHAR  EQU     $F015           INPUT CHARACTER-NO PARITY
F018            OUTCH   EQU     $F018           OUTPUT CHARACTER AND EOD
F01B            OUT2HS  EQU     $F01B           OUTPUT TWO HEX AND A SPACE
F01E            OUT4HS  EQU     $F01E           OUTPUT FOUR HEX AND A SPACE
F021            PCRLF   EQU     $F021           PRINT CAR.L/F
F024            PDATA   EQU     $F024           PRINT CAR.L/F.DATA (TEXT)
F027            STRING  EQU     $F027           PRINT DATA (TEXT)
F02A            OUTSP   EQU     $F02A           OUTPUT A SPACE
F034            TAPIN   EQU     $F034           INPUTS CODE FROM LOADER DEVICE
F064            ADDRD4  EQU     $F064           PRINT,EXAMINE,CHANGE 4-DIGIT NUMBER
F071            ADDRD2  EQU     $F071           PRINT,EXAMINE,CHANGE 2-DIGIT NUMBER
F08F            OUTS    EQU     $F08F           OUTPUT A CHARACTER (NO EOD)
F0A4            OUTCH   EQU     $F0A4           OUTPUT TWO HEX
F0B2            HEX4SL  EQU     $F0B2           OUTPUT 4 HEX AND A SLASH
F0F4            IN1H    EQU     $F0F4           INPUT ONE HEX CHARACTER
F0FA            BEGA    EQU     $F0FA
F0FC            ENDA    EQU     $F0FC
                *
0000                    ORG     $1000           DISASSEMBLER SUBROUTINE
1000 CE1287     START   LDX     #MSGLST         ON ENTRY:
1003 8D04               BSR     PDATA           XTORE CONTAINS THE ADDRESS OF THE INSTRUCTION TO BE DISASSEMBLED.
1005 F0F003             JSR     BEGEND          REQUEST BEG/END ADDRESSES
1008 CE1650             LDX     #XTORE
100B BDF024             JSR     PDATA
100E 7E1000             JMP     START           ON EXIT:
1011 CE1650     LOOP    LDX     #XTORE          XTORE IS UNCHANGED;
1014 BDF064             JSR     ADDRD4          STOR1 CONTAINS THE ADDRESS OF THE NEXT BYTE BEYOND THE INSTRUCTION
1017 F0EA04             BEGA    XTORE           OPRND CONTAINS THE VALUE OF THE INSTRUCTION OPERAND.
101A F0F003             JSR     INEEE
101D BD1213             BSR     DISMBL          DISASSEMBLE ONE INSTRUCTION
1020 FE1650             LDX     XTORE           GET NEW LNGTH ADDRESS
1023 A600               LDA A   0,X             ENDA
1025 EB01               ADD B   1,X             ADD LOWER LIMIT
1027 F0F0FC             SUB B   ENDA+1
102A 2403               BCC     LOOP            ENDA+1
102C 7E1000             JMP     START           START
102F 20E0               BRA     LOOP            LOOP
1031                    *
1031                    *
1031                    *
```

```
1030  BD11E5    78          JSR    OFSUB
1033  FF12CD    79          STX    ADDSTO      SAVE USER ADDRESS
1036  CE12CD    80          LDX    #ADDSTO     PRINT USER ADDRESS
1039  BDFA2B    81 DIS2     JSR    HEX4SL
103C  FE12D6    82          LDX    XTOAE
103F  7F12E8    83          CLR    VERFLG
1042  A600      84          LDA A  0,X
1044  B712E9    85          STA A  OPCODE      SAVE OP CODE
1047  BLF01B    86          JSR    OUT2HS      PRINT FIRST BYTE
104A  FF12BA    87          STX    STOR1
104D  E600      88          LDX    0,X
104F  FF12BC    89          STX    OPRAND      SAVE POTENTIAL OPERAND
1052  B612B9    90          LDA A  OPCODE
1055  2B3A      91          BMI    HARD        DECIDE ON TYPE
1057  8140      92          CMP A  #$40
1059  2C03      93          BGE    MEDIUM
105B  7E1109    94          JMP    EASY
                95 *
105E  840F      96 MEDIUM   AND A  #$F          MASK FOR CODE INDEX
1060  CE1200    97          LDX    #TBLM       TABLE OF BAD CODES
1063  C604      98          LDA B  #MEDBAD     LOOP COUNT
1065  BD119D    99          JSR    BADTST      WEED OUT THE BAD CODES
1068  2750      100         BEQ    BAD
106A  8B30      101         ADD A  #OFSM       OFFSET FOR MNEMONIC TABLE
106C  BD11AA    102         JSR    DCODOP      COMPUTE THE MNEMONIC
106F  2704      103         BEQ    MOK1
1071  8520      104         BIT A  #$20
1073  2745      105         BEQ    BAD
1075  5F        106 MOK1    CLR B
1076  8520      107         BIT A  #$20
1078  2706      108         BEQ    MOK
107A  5C        109         INC B
107B  8510      110 MT2     BIT A  #$10
107D  2701      111         BEQ    MOK
107F  5C        112         INC B
1080  5A        113 MOK     DEC B
1081  2C0C      114         BGE    MEDFIN
1083  C641      115         LDA B  #'A
1085  8150      116         CMP A  #$50
1087  2D01      117         BLT    MEDJ2
1089  5C        118         INC B
108A  F712C2    119 MEDJ2   STA B  ABPRNT       PRINT AN 'A' OR 'B'
108D  C6FF      120         LDA B  #$FF        MARK ONE BYTE
108F  2051      121 MEDFIN  BRA    PRNHM
                122 *
1091  840F      123 HARD    AND A  #$F         MASK FOR CODE INDEX
1093  8103      124         CMP A  #3
1095  2723      125         BEQ    BAD
1097  2D01      126         BLT    HJ0
1099  4A        127         DEC A
109A  8B38      128 HJ0     ADD A  #OFSH       OFFSET TO PROPER ENTRY
109C  BD11AA    129         JSR    DCODOP      SET UP MNEMONIC
109F  270A      130         BEQ    HJ1
10A1  C641      131         LDA B  #'A         ADD 'A' OR 'B' SYMBOL
10A3  2540      132         BIT A  #$40
10A5  2701      133         BEQ    HJ2
10A7  5C        134         INC B
10A8  F712C2    135 HJ2     STA B  ABPRNT
10AB  812D      136 HJ1     CMP A  #$2D        LOOK FOR BSR
10AD  2607      137         BNE    HJ3
10AF  C642      138         LDA B  #'B         FIX MNEMONIC
10B1  F712BE    139         STA B  MNPRNT
10B4  2003      140         BRA    BRNCH       TREAT AS BPANCH
10B6  819D      141 HJ3     CMP A  #$9D        THROW OUT 9D
10B8  2601      142         BNE    HJ4
10BA  39        143 BAD     RTS                SKIP BAD CODES
10BB  16        144 HJ4     TAB                SAVE OP CODE
10BC  8887      145         EOR A  #$87        LOOK FOR 87,8F,C7,CF
10BE  84B7      146         AND A  #$B7
10C0  27F8      147         BEQ    BAD
10C2  C8FF      148         EOR B  #$FF
10C4  17        149         TBA
10C5  84CC      150         AND A  #$CC        LOOK FOR C,D,E,F/C,D,E,F
10C7  2609      151         BNE    HJ5
10C9  C502      152         BIT B  #2
10CB  26ED      153         BNE    BAD
10CD  8658      154         LDA A  #'X
10CF  B712C0    155         STA A  MNPRNT+2    CHANGE TO 'X'
10D2  17        156 HJ5     TBA
10D3  5F        157         CLR B
10D4  84B0      158         AND A  #$B0        BX AND FX ARE 3-BYTE
10D6  2709      159         BEQ    HBYT3
10D8  B612B9    160         LDA A  OPCODE
10DB  883C      161         EOR A  #$3C
10DD  842D      162         AND A  #$2D        8,C/C,E
10DF  2601      163         BNE    FRNHM
10E1  5C        164 HBYT3   INC B
10E2  B612B9    165 PRNHM   LDA A  OPCODE      GET THE OP CODE
10E5  84E0      166         AND A  #$E0
10E7  8120      167         CMP A  #$20        LOOK FOR IMMEDIATE MODE
10E9  270B      168         BEQ    IMMED
10EB  8430      169         AND A  #$30        LOOK FOR INDEX MODE
10ED  8120      170         CMP A  #$20
10EF  2603      171         BNE    PJMP
10F1  7D12B8    172         TST    VERFLG      MARK INDEX MODE
10F4  2040      173 PJMP    BRA    PRNTOP
10F6  8603      174 IMMED   LDA A  #'#         MARK IMMEDIATE MODE
10F8  B712C4    175         STA A  ABPRNT+2
10FB  2039      176         BRA    PRNTOP
                177 *
10FD  CE11F6    178 EASY    LDX    #BADTBL
1100  C614      179         LDA B  #EGYBAD     LOOP COUNT
1102  BD119D    180         JSR    BADTST
1105  27B3      181         BEQ    BAD
1107  BD11AA    182 EJ0     JSR    DCODOP
110A  2708      183         BEQ    EOK
110C  16        184         TAB
110D  C401      185         AND B  #1
110F  CB41      186         ADD B  #'A
1111  F712C2    187         STA B  ABPRNT
1114  C6FF      188 EOK     LDA B  #$FF
1116  8120      189         CMP A  #$20
1118  2D1C      190         BLT    PRNTOP
111A  8130      191         CMP A  #$30
111C  2C18      192         BGE    PRNTOP
111E  5F        193 BRNCH   CLR B               BRANCH INSTRUCTION
111F  FE12BA    194         LDX    STOR1
1122  A600      195         LDA A  0,X
1124  2A01      196         BPL    EJ4
1126  53        197         COM B
1127  FE12CD    198 EJ4     LDX    ADDSTO      GET USER ADDRESS
112A  08        199         INX
112B  08        200         INX
112C  BD11E5    201         JSR    OFSUB
112F  FF12BC    202         STX    OPRAND
1132  7C12E8    203         INC    VERFLG      MARK TWO BYTES OF OPERAND
1135  5F        204         CLR B
                205 *
1136  5D        206 PRNTOP  TST B               HOW MANY BYTES?
1137  2C05      207         BGE    TWOBYT
1139  CE129D    208         LDX    #MSGSSP     SINGLE BYTE INSTRUCTION
113C  2014      209         BRA    PRNSPC
113E  FE12BA    210 TWOBYT  LDX    STOR1
1141  8620      211         LDA A  #$20        ENABLE '$' PRINT
1143  B712C3    212         STA A  ABPRNT+1
1146  5D        213         TST B
1147  2E0E      214         BGT    THRBYT
1149  BDFA14    215         JSR    OUT2H       PRINT SECOND BYTE
114C  FF12BA    216         STX    STOR1       SAVE NEW X
114F  CE129F    217         LDX    #MSG3SP
1152  BDF027    218 PRNSPC  JSR    STRING      FILL WITH SPACES
1155  2006      219         BRA    PRNMNE
1157  BDF01E    220 THRBYT  JSR    OUT4HS      PRINT WORD
115A  FF12BA    221         STX    STOR1
115D  CE129C    222 PRNMNE  LDX    #MSG8SP     PRINT SPACES
1160  BDF027    223         JSR    STRING
1163  CE12BE    224         LDX    #MNPRNT     PRINT MNEMONIC STRING
1166  BDF027    225         JSR    STRING
1169  CE12BC    226         LDX    #OPRAND
116C  7D12B8    227         TST    VERFLG      IS IT A BRANCH STATEMENT?
116F  2E0E      228         BGT    PRN2B
1171  5D        229         TST B
```

```
117E 2D1C      220          BLT    PRNOUT
1180 2E09      231          BGT    PRN2B
1182 A600      232          LDA A  0,X          FIX OPERAND FOR
1184 A701      233          STA A  1,X          FOR AT SIGN AND
1186 6F00      234          CLR    0,X          POSSIBLE CROSS-REFERENCE
1188 08        235          INX
1189 2003      236          BRA    PRN1B
118B BDFA14    237 PRN2B    JSR    OUT2H        TWO-BYTE NUMBER
118E BDFA14    238 PRN1B    JSR    OUT2H
1191 7D12E8    239          TST    VERFLG       INDEX MODE?
1194 2C06      240          BGE    PRNOUT
1196 CE12A3    241          LDX    #MSGNDX      PRINT ',X'
1199 BDF027    242          JSR    STRING
119C 39        243 PRNOUT   RTS
               244 *
119D A100      245 BADTST   CMP A  0,X          IS THIS NUMBER IN TABLE?
119F 2F08      246          BLE    BADRET       SET Z-BIT IF BAD
11A1 4A        247          DEC A
11A2 08        248          INX
11A3 5A        249          DEC B
11A4 26F7      250          BNE    BADTST
11A6 7D12A6    251          TST    ALLONE       MARK GOOD
11A9 39        252 BADRET   RTS
               253 *
11AA CE120E    254 DCODOP   LDX    #MNETBL      LOOK UP MNEMONIC
11AD 48        255          ASL A
11AE BD11E4    256          JSR    OFSUB1       INDEX INTO MNEMONIC TABLE
11B1 E600      257          LDA B  0,X          GET COMPRESSED MNEMONIC
11B3 A601      258          LDA A  1,X
11B5 B712CA    259          STA A  ASTO
11B8 CE12BE    260          LDX    #MNPRNT      POINT TO PRINT LIST
11BB 86F2      261 NXTCH    LDA A  #$F2         INITIALIZE A
11BD 7812CA    262 FIVLP    ASL    ASTO
11C0 59        263          ROL B
11C1 49        264          ROL A                GET A BIT
11C2 25F9      265          BCS    FIVLP        FIVE BITS
11C4 A700      266          STA A  0,X          SAVE CHARACTER
11C6 08        267          INX
11C7 8C12C1    268          CPX    #MNPRNT+3    THREE CHARACTERS
11CA 26EF      269          BNE    NXTCH
11CC 37        270          PSH B
11CD 8620      271          LDA A  #$20         SPACE
11CF C604      272          LDA B  #4           EOT
11D1 A700      273          STA A  0,X
11D3 A701      274          STA A  1,X          SET UP SKELETON
11D5 E702      275          STA B  2,X
11D7 A703      276          STA A  3,X
11D9 1B        277          ABA
11DA A704      278          STA A  4,X
11DC E705      279          STA B  5,X
11DE 33        280          PUL B
11DF B612B9    281          LDA A  OPCODE
11E2 5D        282          TST B                FLAG BIT
11E3 39        283          RTS
               284 *
11E4 5F        285 OFSUB1   CLR B                EIGHT-BIT ENTRY
11E5 FF12CB    286 OFSUB    STX    OFSETX       SIXTEEN-BIT ENTRY
11E8 CE12CB    287          LDX    #OFSETX
11EB AF01      288          ADD A  1,X
11ED E900      289          ADC B  0,X          ADD THE OFFSET
11EF A701      290          STA A  1,X
11F1 E700      291          STA B  0,X
11F3 EE00      292          LDX    0,X          X=X+(B,A)
11F5 39        293          RTS
               294 *
11F6 00        295 BADTBL   FCB    0,1,1,1,1
11F7 01
11F8 01
11F9 01
11FA 01
11FB 0D        296          FCB    $D,$D,$D,$D
11FC 0D
11FD 0D
11FE 0D
11FF 0F        297          FCB    $F,$10,$11,$11,$11
1200 10

1201 11
1202 11
1203 11
1204 11        298          FCB    $11,$12,$28,$29,$3A,$2A
1205 12
1206 28
1207 29
1208 2A
1209 2A
120A 01        299 TBLM     FCB    1,1,3,8
120B 01
120C 03
120D 08
120E 73E0      300 MNETBL   FDB    $73E0        NOP
1210 A060      301          FDB    $A060        TAP
1212 A402      302          FDB    $A402        TPA
1214 4BB0      303          FDB    $4BB0        INX
1216 2170      304          FDB    $2170        DEX
1218 1B2C      305          FDB    $1B2C        CLV
121A 990C      306          FDB    $990C        SEV
121C 1B06      307          FDB    $1B06        CLC
121E 9946      308          FDB    $9946        SEC
1220 1B12      309          FDB    $1B12        CLI
1222 9952      310          FDB    $9952        SEI
1224 9882      311          FDB    $9882        SBA
1226 1882      312          FDB    $1882        CBA
1228 A044      313          FDB    $A044        TAB
122A A082      314          FDB    $A082        TBA
122C 2042      315          FDB    $2042        DAA
122E 0882      316          FDB    $0882        ABA
1230 1482      317          FDB    $1482        BRA
1232 1212      318          FDB    $1212        BHI
1234 1326      319          FDB    $1326        BLS
1236 10C6      320          FDB    $10C6        BCC
1238 10E6      321          FDB    $10E6        BCS
123A 138A      322          FDB    $138A        BNE
123C 1162      323          FDB    $1162        BEQ
123E 1586      324          FDB    $1586        BVC
1240 15A6      325          FDB    $15A6        BVS
1242 1418      326          FDB    $1418        BPL
1244 1352      327          FDB    $1352        BMI
1246 11CA      328          FDB    $11CA        BGE
1248 1328      329          FDB    $1328        BLT
124A 11E8      330          FDB    $11E8        BGT
124C 130A      331          FDB    $130A        BLE
124E A4F0      332          FDB    $A4F0        TSX
1250 4BA6      333          FDB    $4BA6        INS
1252 8559      334          FDB    $8559        PUL A
1254 8559      335          FDB    $8559        PUL B
1256 2166      336          FDB    $2166        DES
1258 A626      337          FDB    $A626        TXS
125A 84D1      338          FDB    $84D1        PSH A
125C 84D1      339          FDB    $84D1        PSH B
125E 9526      340          FDB    $9526        RTS
1260 9512      341          FDB    $9512        RTI
1262 B852      342          FDB    $B852        WAI
1264 9DD2      343          FDB    $9DD2        SWI
1266 714E      344 MNE4     FDB    $714E        NEG
1268 1BDA      345          FDB    $1BDA        COM
126A 64E4      346          FDB    $64E4        LSR
126C 93E4      347          FDB    $93E4        ROR
126E 00E4      348          FDB    $00E4        ASR
1270 00D8      349          FDB    $00D8        ASL
1272 93D8      350          FDB    $93D8        ROL
1274 2146      351          FDB    $2146        DEC
1276 4B86      352          FDB    $4B86        INC
1278 A4E8      353          FDB    $A4E8        TST
127A 5361      354          FDB    $5361        JMP
127C 1B24      355          FDB    $1B24        CLR
127E 9D45      356 MNE8     FDB    $9D45        SUB
1280 1B61      357          FDB    $1B61        CMP
1282 9887      358          FDB    $9887        SBC
1284 0B89      359          FDB    $0B89        AND
1286 1B69      360          FDB    $1B69        BIT
1288 6103      361          FDB    $6103        LDA
128A 9D03      362          FDB    $9D03        STA
```

```
12B0  2BE5      363          FDB   $2BE5          EOR
12B2  0907      364          FDB   $0907          ADC
12B4  7C83      365          FDB   $7C83          ORA
12B6  0909      366          FDB   $0909          ADD
12B8  1C30      367          FDB   $1C30          CPX
12BA  54E4      368          FDB   $54E4          JSR
12BC  6126      369          FDB   $6126          LDS
12BE  9D26      370          FDB   $9D26          STS
12C0  20        371 MSG6SP   FCB   $20
12C1  20        372 MSG5SP   FCB   $20
12C2  20        373 MSG4SP   FCB   $20
12C3  20        374 MSG3SP   FCB   $20
12C4  20        375          FCB   $20
12C5  20        376          FCB   $20
12C6  04        377          FCB   4
12C7  2C58      378 MSGNDX   FCC   /,X/
12C9  04        379          FCB   4
12CA  FF        380 ALLONE   FCB   $FF
12CB  202040    381 MSGLST   FCC   /  LIST/
12D1  04        382          FCB   4
12D2  4F4646    383 MSG0     FCC   /OFFSET/
12D9  20        384          FCB   $20
12DA  04        385          FCB   4
                386 *
                387 *        RAM VARIABLES
                388 *
12DB            389 XTORE    RMB   2
12DD            390 VERFLG   RMB   1
12DE            391 OPCODE   RMB   1
12DF            392 STOR1    RMB   2
12E1            393 OPRAND   RMB   2
12E3            394 MNPRNT   RMB   12
12EF            395 AEPRNT   EQU   MNPRNT+4
12F3            396 ASTO     RMB   1
12F4            397 OFSETX   RMB   2
12F6            398 ADDSTO   RMB   2
12F8            399 OFFSET   RMB   2
12FA            400          END
```

```
MSGLST   12A7   381    42
MSGNDX   12A3   378   241
MSG0     12AE   383    45
MT2      1087   110
NXTCH    11BB   261   269
OCHAR    F9CF    34
OFFSET   12CF   399    47    76    77
OFSETX   12CB   397   286   287
OFSH     0038    74   128
OFSM     002C    73   101
OFSUB    11E5   286    78   201
OFSUB1   11E4   285   256
OPCODE   12B9   391    85    90   160   165   281
OPRAND   12BC   393    89   202   226
OUT2H    FA14    35   215   237   238
OUT2HS   F01B    25    86
OUT4HS   F01E    26   220
OUTCH    F018    24
OUTSP    F02A    30
PCRLF    F021    27    51
PDATA    F024    28    43    46
PJMP     1100   173   171
PRN1B    118E   238   236
PRN2B    118B   237   228   231
PRNHM    10EE   165   121   163
PRNMNE   1169   222   219
PRNOUT   119C   243   230   240
PRNSPC   115E   218   209
PRNTOP   1142   206   173   176   190   192
START    1000    42    59
STOR1    12B9   392    53    56    57    87   194   210   216   221
STRING   F027    29   218   223   225   242
```

```
TAPIN    F8A4    31
TBLM     1203   299    97
THRBYT   1163   220   214
TWOBYT   114A   210   207
VERFLG   12B8   390    83   172   203   227   239
XTORE    12B6   389    50    75    82
ABPRNT   12C2   395   119   135   175   187   212
ADDRD2   F971    33
ADDRD4   F964    32    48
ADDRIN   F00F    21
ADDSTO   12CD   398    79    20   193
ALLONE   12A6   380   251
ASTO     12CA   396   259   262
BAD      10C6   143   100   105   125   147   153   181
BADRET   11A9   252   246
BADTEL   11F6   295   178
BADTST   119D   245    99   180   250
BEGA     FF0A    38    49
BEGEND   F003    17    44
BRNCH    112A   193   140
CBCDHX   F006    18
CHEXL    F009    19
CHEXR    F00C    20
DCODOP   11AA   254   102   129   182
DIS2     1045    81
DISMBL   1033    75    52
EASY     1109   178    94
EJ0      1113   182
EJ4      1133   198   196
ENDA     FF0C    39    54    55
EOK      1120   188   183
ESYBAD   0014    71   179
EXBJGA   F000    16
FIVLP    11BD   262   265
HARD     109D   123    91
HBYT3    10ED   164   159
HEX4SL   FA2B    36    81
HJ0      10A6   133   126
HJ1      10B7   136   130
HJ2      10B4   135   133
HJ3      10C2   141   137
HJ4      10C7   144   142
HJ5      10DE   156   151
IMMED    1102   174   168
IN1H     FAA5    37
INCH     F012    22
INCHNP   F015    23
LOOP     1018    50    58
MEDBAD   0004    72    98
MEDBIN   109B   121   114
MEDIUM   106A    96    93
MEDJ2    1096   119   117
MNEM     1266   344
MNES     127E   356
MNETBL   1205   300   254
MNPRNT   12EC   394   139   155   224   260   268   395
MOK      10E0   113   108   111
MOK1     10E1   106   102
MSG3SP   129F   374   217
MSG4SP   129E   373
MSG5SP   129D   372   201
MSG6SP   129C   371   200
```

PROPOSED TINY PILOT

by Dennis Allison December 3, 1976

169 Spruce Avenue, Menlo Park, CA 94025 (415) 325-2962

0. INTRODUCTION

This note describes a proposed syntax and semantics for a programming language based on PILOT 73 but suitable for a minimal resources implementation on a microprocessor. The language is embedded in a program creation and editing facility suitable for interactive program development.

1. PROGRAMS: INFORMAL DESCRIPTION

A program in TINY PILOT (TP) is a collection of numbered lines. The line numbers impose an ordering on program statements and must be in the range 1. . 32767. Line numbers serve as names for program statements in exactly the same fashion as they do in BASIC.

All program statements begin with a single letter keyword which determines the statement type. The statements supported are:

T	Type text from terminal
A	Acquire answer from terminal
M	Test answer for match
C	Compute
U	Use as subroutine (GOSUB)
J	Jump (GOTO)
E	Return (or terminate) (RETURN/END)

All statements have an identical prefix syntactical form:

line-number keyword condition? colon

where the condition is either a parenthesized expression or the special conditions Y or N. The special conditions test the condition of a global flag which indicates whether the last match was successful. A condition is true if it is non-zero; statements with conditions are executed only if the condition is true.

TP provides 26 numerical variables and 26 string variables. String variables are dynamically allocated storage from a common pool.

Numeric variables are represented as #A . . #Z. String variables are represented as $A . . $Z

There are two kinds of strings in TP: open strings and closed strings. An open string is simply a sequence of characters. A closed string is enclosed in quotes. There are significant differences in interpretation.

In an open string any instance of a variable (string or numeric) is replaced by its current value. Default values are zero for numeric variables and the null string for string variables. In a closed string, no replacement takes place; all characters are printed literally. A quote mark in a closed string is represented by two immediately adjacent quote marks.

Closed strings may be catenated to open strings by simply writing them adjacent. For example:

```
NOW IS THE TIME TO PRINT "$A" WHICH
  CONTAINS $A
```

would print as

```
NOW IS THE TIME TO PRINT $A WHICH
  CONTAINS HI
```

if the current value of $A was "HI".

Type Statement

The type statement has the form:

line-number T condition? colon string carriage-suppress?

and causes the string to be output to the terminal. The carriage-return line-feed which is usually produced may be suppressed by using a terminating at-sign. (The at-sign was chosen because it will probably be little used in most TP programs)

Examples:

```
100 T:HI THERE
200 TY:"$A$A$A$A$A$A"ABCDEFGHIJ
300 T: WHAT'S YOUR NAME @
400 TN: I DON'T UNDERSTAND, TRY "$A"@
```

Answer Statement

The answer statement has the form:

line-number A condition? colon variable

and causes either a string value to be read from the terminal and stored in the string variable or a numeric value to be read and stored in the numeric variable.

Syntactic errors on numeric entry cause a diagnostic ("I NEED A NUMBER") and a retry.

Examples:

```
100 A: #S
200 A: $Q
300 AY: #T
400 A(X=) : #U
```

Match Statement

The match statement has the form:

line-number M condition? colon pattern-strings

It causes each of the pattern-strings (see below) to be tested for substring match against the most recently entered answer. If a match is found the flag is set true (=1), if not, it is set false (=0).

Pattern-strings are lists of open and closed strings; pattern strings are delimited by commas, therefore a comma cannot appear in a pattern except in a closed string. Leading blanks are ignored in open strings.

Examples:

```
100 MY: DOG, PUPPY, KITTY, HORSE, LION, PIG
200 MY: "DOG" , "PUPPY", "KITTY"
300 M:: , . , ' , ; , " , "
```

Compute

The compute statement has the form:

line-number C condition? colon numeric-variable equals expression

The expression is evaluated and the value stored in the numeric-variable. All arithmetic is integer with values in the range –32768 . . 32767. Overflows result in an error message ("NUMBER OUT OF RANGE") and a return to program mode.

All the usual operators (+, –, *, /, AND, OR, XOR, NOT, . =, =, =) are provided with the usual precedence. Partial evaluation depth caused by parenthesis or operator precedence is limited to 6.

Examples:

```
100 C: #X = #X + 5
200 C(#A  0): #A = –#A
300 C: #A = #A+#B+ (#C+#D) / (#E+#G*8976)
400 CY:  #B  =  (#A=3)  or  (#V=9)
```

Use

The use statement has the form:

line-number U condition? colon line-number

Execution of a use statement causes the line-number of the current statement to be placed into a subroutine return stack. Program execution continues at the line number specified.

An attempt to use a non-existent statement number is a run-time error and causes a diagnostic to be output and control to revert to program mode.

Examples:

100 UY: 25
200 U: 750
300 UY: 7

Jump

The jump statement has the form:

line-number J condition? colon line-number

Execution of a jump statement causes the program to begin executing statements at the line-number specified.

An attempt to use a non-existant statement number is a run-time error and causes a diagnostic to be output and control to revert to program mode.

Examples:

100 JY: 79
200 J: 1000
300 J (#X = #Y): 88

End

The end statement has the form:

line-number E condition? colon line-number

Execution of an end statement causes the program to begin executing statements at the statement following the statement whose number appears at the top of the subroutine return stack. That number is deleted from the stack. If the stack is empty, the system reverts to program mode.

2. EDITING AND CONTROL: INFORMAL DESCRIPTION

TP programs are built and modified in program mode. They are invoked by commands available in program mode which initialize and convert to run mode.

The several commands which are available are:

RUN	Run a TP program
LIST	List the TP program
LOAD	Load a TP program from external media
SAVE	Save a TP program on external media
DEL	Delete lines specified from TP program
SEE	Display value of a TP variable or expression
CONT	Continue after interruption

Lines with line numbers are inserted or appended to the program.

At line entry time, the usual backspace and line delete functions are available just as in BASIC.

RUN takes an optional line-number parameter to indicate the starting line.

LIST and DEL both have the possibility of specifying a range or line numbers or a starting line-number.

SEE displays a single string variable or an arbitrary numeric expression. It is provided as a debug aid.

3. PROGRAM INTERRUPTION

The program can be interrupted (returned to program mode) by typing a break-character at any time. This produces a diagnostic message ("STOPPED BEFORE LINE nnn"). The program can be edited and continued following an interruption.

4. IMPLEMENTATION

One needs a slightly different interpreter mechanism than that used for Tiny BASIC because of the presence of strings, but the same interpretive intermediate language approach would be a plausible implementation strategy. Alternatively, one might use a procedural approach or a table-driven approach which would be somewhat faster.

More on this later as the language firms up.

5. CONCLUSIONS

A very simple version of PILOT has been defined together with some simple editing facilities for program development. Some of the features of PILOT 73 have been eliminated in favor of what we feel is a more consistent and simpler approach.

Comments and counter proposals are solicited. **[See PCC Newspaper Vol. 5, No. 3 and Vol. 5, No. 4 for more PILOT information.]**

ATTOFILE IT, PLEASE

March 2, 1977

From time to time, over the past twenty years or so, there have been predictions that we will soon have inexpensive mass storage devices capable of holding the entire contents of the Library of Congress in a $19 desk-top unit. These chimerical devices are usually based on some far-out technology involving proton resonance, magnetic bubbles, or holograms. Well, I'm still waiting patiently for such a device to materialize but I'm not holding my breath. Still, it's fun to speculate on the possible applications of a truly inexpensive storage device of Brobdingnagian capacity.

One possibility is the gradual conversion of all library books to the new medium, let's call it an Attofile ("atto" implying a reduction factor of 10 to the -18th power). Assuming that copyright problems could be worked out, there might be no need to have more than one copy of any any document, or more than one Attofile, since presumably any number of people could access the same Attofile copy simultaneously.

Obviously, there are certain hazards in such a lack of redundancy, since a single accident or act of sabotage could conceivably wipe out the totality of recorded human knowledge. The consequences of such a calamity are dealt with in Hal Draper's delightful SF story "MS FND IN A LBRY," published in the December 1961 issue of *The Magazine of Fantasy and Science Fiction.*

Use of a machine-readable Attofile could facilitate the development of some novel forms of literature. Computerized document retrieval and display is not new, but in most cases the static format of the original printed matter is retained. With an Attofile it would be possible to create and reproduce text, line drawings, halftones, and animated illustrations, as well as accompanying sound effects, music, or narration. For example, children could learn to read by viewing an Attofile version of *Go Dog, Go* with simultaneous narration, text display, and full-color animation, perhaps with computer-generated background music.

If an Attofile document included a word outside the reader's vocabulary, the reader could stop the display temporarily, indicate the unknown word with a light pen, and get an appropriate definition. A reader who wanted to browse could request an abstract of any document or could ask for a digest version of a book rather than the entire text. The computer accessing the Attofile could provide a coherent condensation of any desired length or could generate an expanded or annotated version of the original.

A full range of bibliographic and cross-indexing services could also be provided on demand. For example, in the case case of scientific journals, the reader could request a list of references cited in a specific paper and could also ask for lists of references cited in *those* publications, etc. Perhaps of even greater utility would be the ability to compile and list *forward* citations referencing a given paper.

An Attofile computer could perform other, less passive, tasks as well. For example, it could produce a version of *Pilgrim's Progress* rewritten in the style of Art Buchwald or James Joyce, or perhaps an iambic tetrameter version of *War and Peace.*

Doubtless many other possibilities will occur to the reader reader. But when are we going to get that $19 Attofile with Altair interface? Sooner than we think, perhaps, but I'm still not holding my breath. In fact, I'm willing to settle for a good $300 floppy disk.

Jim Day

A SC/MP SUBROUTINE SUPERVISOR

Dear Dr. Dobbs: February 5, 1977

Having enjoyed your fine magazine for a year now, I am finally getting around to making a contribution. Attached is a subroutine supervisor for the SC/MP which provides the jump-to-subroutine capability enjoyed by users of the 8080 and other CPU's. (It goes them one better by allowing unlimited depth of indirect addressing.) Please note that the storage locations shown are completely arbitrary as all addressing is program-or register-relative.

My system (SC/MP board, TV Typewriter, cassette interface, and 4K memory) is built into an attache case, which simplifies the logistics of computer club meetings. I would like to hear from anyone else who is using this CPU. My first project will be an interactive assembler, followed (possibly) by a BASIC compiler (not interpreter).

I am intrigued by the concept behind the FORTH language, if not the actual implementation being marketed. The ability to teach a computer gradually and interactively sounds like an ideal approach for the hobbyist. However, I have found very little written on this subject, other than a rather brief article in *Interface Age* which apparently was never followed up as promised. Do you know of anything being done in this area?

Keep up the good work. I still consider *DDJ* the only computer hobbyist magazine for grown-ups.

Richard K. Peterson
323 W. 63rd Street
Davenport, Iowa 52806

```
*****************************************************************
*    "SCMPCALL" IS A SUBROUTINE CALL/RETURN SUPERVISOR FOR THE SC/MP. *
* IT SUPPORTS INDIRECT ADDRESSING, NESTING, AND RECURSION.  REGISTERS *
* AND STATUS ARE NOT AFFECTED. MEMORY REQUIREMENT IS 75 BYTES, PLUS   *
* 2 BYTES FOR EACH LEVEL OF NESTING.  REGISTER P3 IS RESERVED.        *
*    A SUBROUTINE IS CALLED BY AN "XPPC P3" INSTRUCTION, FOLLOWED BY  *
* A 2 BYTE ADDRESS SPECIFYING THE SUBROUTINE ENTRY POINT, OR, IF THE  *
* HIGH ORDER BIT IS 1, THE LOCATION OF ANOTHER DIRECT OR INDIRECT     *
* ADDRESS. IF THE FIRST ADDRESS BYTE IS ZERO, CONTROL IS RETURNED TO  *
* THE MOST RECENT CALLER.  THEREFORE THE "RETURN" INSTRUCTION IS 2    *
* BYTES LONG INSTEAD OF 3.  "CALL" ADDRESSES MUST BE IN THE RANGE     *
* '0100' TO '7FFF'.                                                   *
*    THE STACK POINTER MUST BE INITIALIZED WITH ITS OWN ADDRESS, AND  *
* P3 WITH THE CALL/RETURN ENTRY POINT MINUS 1.  P3 RETAINS THIS VALUE *
* AT ALL TIMES OUTSIDE THE CALL SUPERVISOR.  "SCMPCALL" RESIDES IN    *
* HIGH MEMORY, AND THE STACK EXTENDS DOWNWARD AS NECESSARY.           *
*    EXAMPLES:                                                        *
*      3F 04 EC   (CALL A SUBROUTINE WHOSE ENTRY POINT IS '4EC')      *
*      3F 84 EC   (CALL A SUBROUTINE WHOSE ADDRESS IS AT '4EC')       *
*      3F 00      (EXIT FROM SUBROUTINE & RETURN TO CALLER)           *
*****************************************************************

0000            SCMPCALL CSECT
0000                     ORG    *+X'FA1'  POSITION BELOW MONITOR WORKSPACE
                *
0FA1 0000       STACK    DC     X'0000'   STACK POINTER
                *
0FA3 00         CALLAC   DC     X'00'     ACCUMULATOR SAVE
0FA4 00         CALLEX   DC     X'00'     EXTENSION SAVE
                *
0FA5 C601       CALLRET  LD     @1(P2)    UNSTACK RETURN ADDRESS
0FA7 37                  XPAH   P3        INTO LINK REGISTER
0FA8 C601                LD     @1(P2)    REPEAT FOR LOW BYTE
0FAA 33                  XPAL   P3
0FAB C7FF       CALLGO   LD     @-1(P3)   ADJUST LINK ADDRESS FOR XPPC
0FAD C0F3                LD     STACK     GET SAVED P2 CONTENTS
0FAF 36                  XPAH   P2        RESTORE P2 HIGH BYTE
0FB0 C8F0                ST     STACK     SAVE NEW STACK POINTER
0FB2 C0EF                LD     STACK+1   REPEAT FOR LOW BYTE
0FB4 32                  XPAL   P2
0FB5 C8EC                ST     STACK+1
0FB7 C0EC                LD     CALLEX    GET SAVED EX CONTENTS
0FB9 01                  XAE              RESTORE EX REGISTER
0FBA C0E8                LD     CALLAC    RESTORE ACCUMULATOR
0FBC 3F                  XPPC   P3        LINK TO PROGRAM & RESTORE P3
```

```
0FBD C8E5      CALL      ST     CALLAC     ** CALL/RETURN ENTRY POINT
0FBF 40                  LDE               SAVE EX REGISTER
0FC0 C8E3                ST     CALLEX     SAVE EX REGISTER
0FC2 C0DE                LD     STACK      GET STACK POINTER HIGH BYTE
0FC4 36                  XPAH   P2         P2 IS STACK REGISTER
0FC5 C8DB                ST     STACK      SAVE PREVIOUS P2 CONTENTS
0FC7 C0DA                LD     STACK+1    REPEAT FOR LOW BYTE
0FC9 32                  XPAL   P2
0FCA C8D7                ST     STACK+1
0FCC C301                LD     1(P3)      GET SUBROUTINE HIGH BYTE
0FCE 98D5                JZ     CALLRET    IF ZERO, RETURN REQUEST
0FD0 01                  XAE               SAVE HIGH BYTE IN EX
0FD1 C703                LD     @3(P3)     ADJUST RETURN ADDRESS
0FD3 C3FF                LD     -1(P3)     GET SUBROUTINE LOW BYTE
0FD5 33                  XPAL   P3         SET LINK REGISTER
0FD6 CEFF                ST     @-1(P2)    STACK RETURN LOW BYTE
0FD8 40                  LDE               GET SUBROUTINE HIGH BYTE
0FD9 37                  XPAH   P3         SET LINK REGISTER
0FDA CEFF                ST     @-1(P2)    STACK RETURN HIGH BYTE
0FDC 40        CALLSUB   LDE               GET SUBROUTINE HIGH BYTE
0FDD 94CC                JP     CALLGO     IF HIGH BIT ZERO, GO
0FDF D47F                ANI    X'7F'      INDIRECT ADDRESS IN P3
0FE1 37                  XPAH   P3         ZERO HIGH BIT & REPLACE
0FE2 C300                LD     0(P3)      GET INDIRECT ADDRESS
0FE4 01                  XAE               SAVE HIGH BYTE
0FE5 C301                LD     1(P3)      GET LOW BYTE
0FE7 33                  XPAL   P3         SET INDIRECT LINK
0FE8 40                  LDE               GET HIGH BYTE
0FE9 37                  XPAH   P3         SET INDIRECT LINK
0FEA 90F0                JMP    CALLSUB    GO CHECK NEW ADDRESS
```

HI? . . . NOT YET

HI to all on my mailing list, January 20, 1977

Enclosed are TINY HI specs for your use (not for publication – I want the next appearance of TINY in *DDJ* to be in the form of real live executable, testable code – KERNEL).

These specs aren't worth $3. I'm sending them free to my first customers and returning their money. I'll charge a buck for expenses if I get many queries.

Sorry I'm 5 days late on these specs.

I am not going into great detail on how to build a dynamic allocator or link programs with an OS call, because I'll have a working model for publication in a couple of months.

This is the second anniversary of my first real encounter with computers, and the first anniversary of my obsession with programming languages. It feels good to start coding after a year of design (not that design stops, but is has reached the point where it needs data from a working system and real users). Also waiting to be coded are a preprocessor to translate NICE BASIC (my design – structured programming – mnemonic names – more flexible comments) into BASIC, and a tiny LIST interpreter.

TINY has been simplified in several ways. Programs can no longer be nested (no real reason since data cannot be inherited from your surrounding block as in ALGOL or PL/I), and program references must include an argument list, even if empty. Programs may be directly or indirectly recursive (something I knew only after I designed the allocator). Input has been drastically simplified. Deletion has been simplified. My vocabulary has changed somewhat. One concept I'm proud of is *program oriented* text-editing functions which make indenting to show structure, and also commenting, easier. One design problem I'm leaving open (I haven't found a solution that satisfies me) is program entry – how lines are ordered, listed, deleted, inserted, etc.

I want to back down on a couple of points. TINY will have a virtual memory in that everything will appear to the user to be in main memory – but in actuality, the storage available to the active process (in main memory) limits the size of file that can be processed. I have no paging scheme, and I don't believe microcomputers are the right place for that sort of scheme. My dynamic allocator also does not keep a record of who owns storage that it allocates, counting on processes to return it. One consequence is that the allocator cannot eliminate fragmentation by compacting allocated areas. The second point I was wrong on is my offhand attack on the "multilevel approach" in systems software – which can also be interpreted as top-down development, hierarchial layering of virtual machines, etc., which I favor.

I want to make clear that some of the weaknesses of TINY (no arrays or real variables) are not inherent, but simply complications I wish to avoid in my first attempt at such a compiler/interpreter. These considerations may not worry more experienced programmers who will want to add these features.

All feedback is appreciated. Special thanks to *Dr. Dobbs Journal* and my correspondents for previous help.

Martin Buchanan
2040 Lord Fairfax Rd.
Vienna, VA 22180

As requested, we are deferring publication of the specs, and eagerly await KERNAL. –Editor

THE SELECTRIC TYPEWRITER AS AN INPUT/OUTPUT DEVICE

Rod Graves
Anderson Jacobson Inc., 521 Charcot Ave., San Jose CA 95131
(408) 263-8520

The first true Input/Output device based on the electric typewriter was introduced in the 1940's. Small solenoids were developed so that forty-two of them could be placed beneath a standard 8½" keyboard to operate the forty-two letters on the standard electric typewriter. For transmitting there were also forty-two switch contacts struck by the keys so that each key triggered a signal from the typewriter to the reader of the tabulating machine. Logistic problems such as cables large enough to house 56 connecting wires were not important because the distances were never more than a few feet. Transmitting data from place-to-place around the country was done either by mail or by teletype, then reproduced at the tabulating center. This concept was successful until the mid-fifties when the first UNIVAC computer was introduced. This event spurred IBM to introduce its first truly electronic computer which was released at about the same time as their first "Selectric" typewriter. Actually, the Selectric was developed *as* an I/O device for the System 360. In this case it's the Selectric we're interested in because of its basic operating theory of "single element" typing.

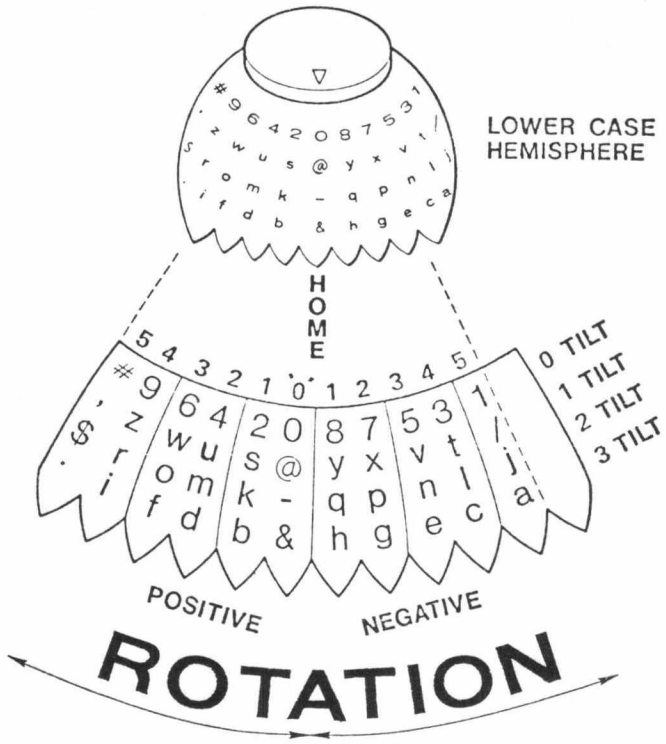
For the first time an electric typewriter was logically coded to produce a single character from a spherical element containing 88 characters. Each keyboard position was mechanically coded to operate a selection mechanism capable of moving type element into position for the desired character. The keyboard was a MECHANICAL input device to a BINARY LOGIC system accessing the forty-four characters in each hemisphere of the type element. Rotating the element 180 degrees made another forty-four characters available in "upper case". To those people seeking a computer terminal, this provided access to and from the unit on 38 fewer cables and was adaptable to encoding in an Extended Binary Coded Decimal (EBCD) system.

Let us now get a look at the mechanics of the Selectric typewriter to better understand what's needed to operate it electrically to and from remote locations.

Aside from printing data characters, the terminal must be capable of performing five additional functions, viz: Carrier Return, Tab, Line Feed, Backspace and Shift. Each of these functions must be coded as a single character for transmitting. For receiving, a single solenoid is used to operate each of the various functions. Each function uses one or more switches to encode its operation for transmitting. Either of two types of switches are used (by Anderson Jacobson, Inc.). They are MICRO switches or REED switches. Micro switches are used wherever direct physical contact will not interfere with the mechanical operation being monitored. Reed switches are used wherever direct mechanical contact would possibly interfere with an operation. They are operated magnetically and so we avoid direct mechanical connection. Beyond this, they are hermetically sealed, extremely reliable and quite small.

Each of the operational functions requires more time for completion than a single typed character requires. Because of this we must stop incoming data (or store it somewhere) until the longer function has ended. The upshot of this is that we DO require feedback data to signal every completed operation. It's time now to take on the more complex idea of encoding and decoding the character selections mechanically.

To gain a basic idea of how characters are selected and printed by a Selectric typewriter, lets observe Figure 1. Depicted there is a type element (or sphere). Note that each



band around the element contains eleven characters per half. The normal or "HOME" position of this element is such that the center character of each band is directly opposite the next typing position on the page. By tilting the sphere we may move any of these character bands to the writing line for printing. Consider the TOP band of characters as "BAND ZERO" and succeeding bands as "ONE", "TWO" and "THREE". The mechanical selection was evolved this way to reduce the number of possible selections to THREE. Making NO mechanical TILT SELECTION will always cause a character from the top band (BAND ZERO) to be printed. There is one mechanical tilting mechanism which provides ONE increment of TILT MOTION. A second tilt selection provides TWO increments of TILT MOTION. Operating these selections *together* provides THREE increments of TILT. Hence, TWO mechanical inputs provide FOUR mechanical selections. We can now see that this is compatible with a BINARY ENCODING SYSTEM.

In the same manner, the sphere may be ROTATED to any of FIVE positions on either side of the zero-rotate (HOME) position. To do this, the rotational position of "zero" is "balanced" so that a single rotate selection will rotate the element FIVE character positions in the CLOCKWISE direction. This is accomplished by spring tension. It is called "NEGATIVE" rotate.

Positive rotate is accomplished in steps. There are three mechanical selections available for positioning the element in the positive direction. The first provides ONE unit of character rotation. Each of the other two selections provide TWO units of positive rotation. As with tilt, these selections ADD, providing THREE, FOUR, and FIVE units as combined.

Algebraically, the machine will add any value of positive rotate selected to the negative-five selection to provide any negative rotate character between zero and minus five.

For speed and control purposes it was advisable to operate the "spacebar" as a non-printing character, but all 88 character selections were accounted for. The character ZERO (on EBCD machines) occupies the zero-tilt, zero-rotate position of the element and is normally selected by making "no mechanical selection". By re-coding this character as a

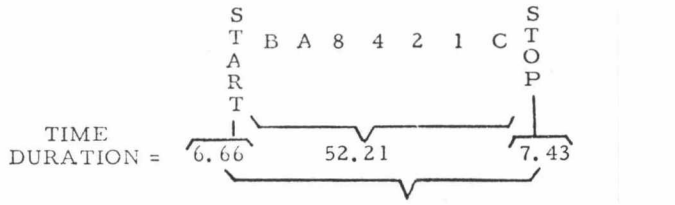
"plus-five, negative-five rotate combination" the "no selection" combination was available for the character zero as a non-print selection. Hence we have expanded the character selection ability to 89 combinations.

Selecting these 89 characters and five mechanical functions is accomplished remotely by electrical solenoid operation. There are six solenoids to make the selection and one to cycle the printer for each printed character. The print cycle and all selection solenoids may be operated simultaneously – as happens when the keyboard is the printer's source of operation, but practical timing makes this inadvisable. Getting all these solenoids to operate at nearly the same time requires parallel wiring and is called "parallel data" when the seven signals are transmitted to the terminal at the same time.

When the printer is operated from the keyboard, each of the mechanical selection components is monitored by moving a miniature magnet in close proximity to a magnetic "reed switch". The switch opens or closes indicating which selection component is being used. When the mechanical selection is near the apex of its operation, a sample switch closes and electrical current passes through each closed switch directly to the parallel transmission lines.

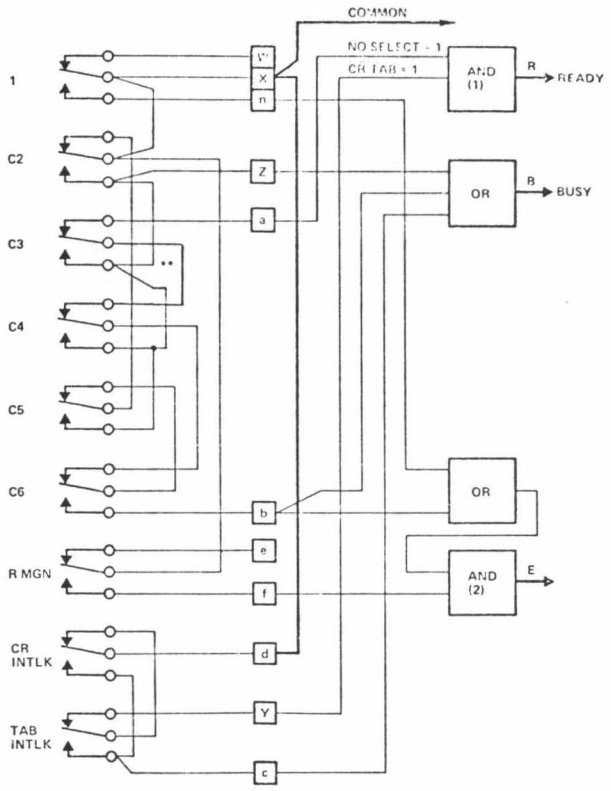
To handle such data required encoding it to a standard system for recognition. Extended Binary Coded Decimal values were assigned to the bit configuration of each character, function and control signal required. This was a seven-bit code.

To transmit the data over a single pair of wires, the dimension of time was added and each bit comprising a character was "clocked" out of a storage buffer. This conversion serializes the data for transmitting. A start and stop bit were then needed to demark the beginning and end of each character transmitted. We now have a nine-bit code. The nine "bits" of each character code are (Figure 2):



BITS	FUNCTION
1	START: This bit is defined as a transition from a "MARK" (+) condition to a "SPACE" (-) condition on the DATA TRANSMIT LINE.
2-7	DATA BITS: The next six bits are the encoded character bits. They are expressed in the hexidecimal format as B A 8 4 2 1.
8	CHECK: An ODD-BIT PARITY is employed wherein all data must be comprised of an ODD number of bits. Whenever an even number of bits comprise a character, the printer adds the "check" bit with the encoding to insure the validity of transmitted data. The check bit is derived mechanically along with the character selection.
9	STOP: The ninth bit is the "stop" bit, signalling the end of one character by restoring the transmit line to a "mark" condition. The minimum time duration of a stop bit is one character time. (Same as one bit time.)

There are two types of operation that have been used for I/O printers. They are "CLOSED LOOP" and "OPEN LOOP" (synonyms are "Asynchronous" and "synchronous"). Let's get a look at a "closed loop" as recommended by IBM for their Selectric I/O. (Figure 3)



* = CORRESPONDENCE WIRING
** = EBCD WIRING

COMMENTS ON THE CONDITIONS ILLUSTRATED IN FIGURE 3: In this diagram, "TRUE" conditions of READY and BUSY exist when a logical "ONE" appears at the output of a logic block. A "LOGIC 1" is generated when a logic block INPUT is conditioned by I/O contact closure BETWEEN the block and power common. (Anderson Jacobson, Inc. uses the absence of a "READY" condition to indicate "NOT READY" for character transmission and "BUSY" to indicate the operation of the various functions. This is possible because we've incorporated DTL logic into the printer base and use simpler "single-pole, single-throw" switches.)

READY (R): Defines the period of time when any of the I/O magnets may be addressed. Those addressed should be held operated for the full duration of the "R" condition. This allows the selected operation to control the duration of the solenoid pulse. The "R" condition obtains when all selection interlock switches are in their normal (non-transferred) condition.

BUSY (B): Defines a period of time during which NO I/O SOLENOIDS ARE TO BE ADDRESSED. The break-before-make contacts used here are not used by Anderson Jacobson. Our Reed switches used in conjunction with "filter" switches and DTL logic allows us to control "BUSY" and "NEXT CYCLE" by degree of camshaft rotation, providing much more reliable operation.

FEEDBACK and CHARACTER SELECTION CONTACTS: During an INPUT (transmit) operation, the purpose of contacts C-1 (and C-3 through C-6) is to GATE the sampling of the transmit

contacts. The N/O contacts shown here should make AFTER the transmit contacts CLOSE and should open BEFORE the transmit contacts re-open. This control of transmit contacts provides the condition where voltage is never applied to contacts while they are being transferred.

FIGURE 4: Once a print command is issued to the terminal, its cycle shaft begins to rotate. When it has rotated 10° the first feedback switch (FILTER 2) opens indicating the terminal is now "BUSY". It will remain busy until this contact closes again near the end of the character cycle.

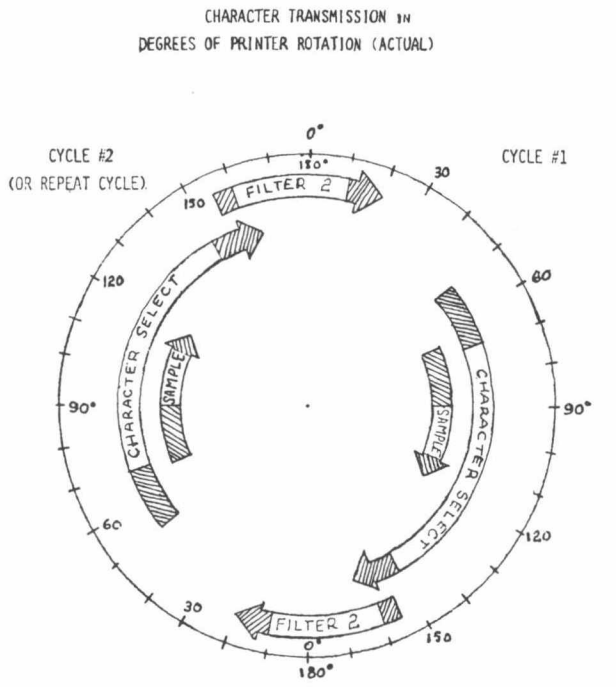
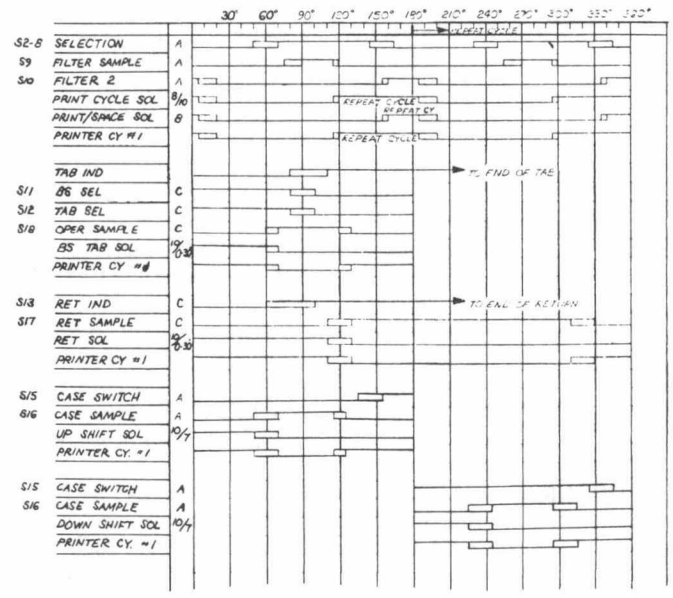


FIGURE 5: If a FUNCTION command is first issued, the operational shaft is set in motion. Depending on which function is selected, the feedback contacts open at approximately 60° and "busy-out" the terminal. The duration of the "BUSY" line is determined by the completion of the selected operation.

CONTROL DEVICE COMMANDS: Commands from the control device are released to the I/O in relation to the degrees



of camshaft rotation rather than on a time-interval basis. This prevents the time required for any given operation from being a factor in reliable operation.

FEEDBACK CONTACTS: These are timed to permit initiation of "next cycle" prior to the conclusion of "current cycle". This avoids stopping the I/O between succeeding characters or repeat functions. It has the effect of increasing terminal longevity, providing maximum throughput speed, relieving "full-power" stop and start of printer clutches.

EXPLANATION OF "CLOSED-LOOP" PREFERENCE: Operating an I/O device in a system where the control device issues commands on a time basis is NOT RECOMMENDED. Such an approach is referred to as "Open Loop" and is shunned because an I/O device of the Selectric nature should receive its commands in relation to the dynamic location of the terminal as measured in the number of degrees rotation of its current mechanical operation. A closed loop provides TOTAL INTERLOCKING between terminal and control device. Without total interlocking, the control device must assume the ability to synchronize the I/O at line speed. Such an approach requires that the time necessary to complete each individual machine cycle be an unvarying constant.

SELECTRIC I/O TIMES ARE NOMINAL, NOT CONSTANT. They can be maintained in a closed loop system because of the total interlocking provided. Occasional losses of machine cycles would be *inherent* in an open loop operation.

PARITY CONTACTS: Parity provides a method of checking the character selection to assure it is valid. This does not insure that the *correct* character has been selected or printed; only that a *valid* selection was made. The check is mechanically encoded by keyboard interposers for all characters comprised of an EVEN number of selection latches. When EVEN, the parity latch is operated, closing the parity contacts so the bit configuration transmitted will be comprised of an ODD number of bits. Logic for Odd-Bit Parity is standard. Even parity may be provided.

SOLENOID/MAGNET COIL SPECIFICATIONS

WHERE USED	RATED Voltage	RESISTANCE IN OHMS (LOW)	RATED	(HIGH)	MAX M.V.*
KBD**	48	329	358	397	146
Lck	24	100	105	110	240
UPPER	48	221	240	259	217
SHIFT	24	62	65	68	387
NO	48	345	395	455	139
PRINT	24	125	137	150	192
ALL	48	432	475	518	111
Others	24	122	128	133	197

*Theoretical maximum current (in Milliamperes) at rated voltage--computed as a ratio of Rated Voltage to Low Resistance.
**100% Duty Cycle. The Duty Cycle of all other magnets is described as "sufficient to provide continuous machine operation when magnet pulses are gated by the Feedback and Interlock Contacts.

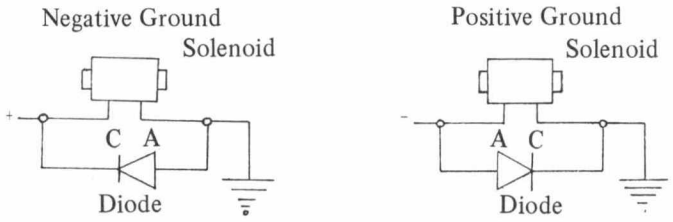
COMMENTS ON SOLENOIDS:

Solenoid coils may be found in both 24 and 48 volt operating ranges. Solenoids used for UPPER CASE, NO PRINT, and KEYBOARD LOCK operate within 12 milliseconds when used at their rated voltage. They may be operated at 10% above this rating but the operating speed will vary.

ARC SUPPRESSION:

All magnets and solenoids use (for best results) the diode method of arc suppression. The standard for I/O

wiring is NEGATIVE GROUND, therefore arc suppression requires that the ANODE of such diodes be common. Both methods are illustrated below.



ARC SUPPRESSION

CONTACT SPECIFICATIONS (As recommended by IBM):

GENERAL: Contacts must operate within 10 to 300 milliamps at 10 to 48 volts, DC.

WIRE CONTACTS:

200 to 300 mA only if used in steady, non-switching state.
40 to 200 mA if adequately suppressed.
10 to 40 mA if not arc-suppressed.
10 mA (or less) NOT RECOMMENDED.

REED SWITCHES:

200 to 300 mA, non-switching.
40 to 200 mA if not arc-suppressed.

Contact bounce is not a problem with magnetic reed switches. Open switching contacts and wire contacts bounce. Switch-bounce may be present for a period of one to five milliseconds on either of these. The fast acting solid-state circuitry of any control device must be protected against contact bounce or discontinuities greater than 200 MICRO-seconds. Bi-stable switches (or reed switches) must be used.

CHOOSING A PRINTER: To provide output and/or input you must first have a printer adaptable to Binary Encoding. The most compatible printer available is IBM's Selectric mechanism. There are two basic choices: The OFFICE MODEL or the I/O (Heavy Duty) MODEL. Office models are available in one configuration only – Correspondence. I/O's are provided in either Correspondence or EBCD.

NECESSARY COMPONENTS TO OPERATE A SELECTRIC AS AN I/O

OUTPUT (Universally needed)	INPUT (add to basic needs)
SOLENOIDS for:	SOLENOIDS for:
Character Selection (6ea)	Keyboard Lock (2ea)
Print Cycle (start)	Non Escape (optional)
No Print Selection	
Upper Case Shift	
Lower Case Shift	
INITIATE Lower Case	
Carrier Return	
Index (Line Feed)	
Tab	
Backspace	
TOTAL OUTPUT SOLENOIDS= 15	
TOTAL INPUT SOLENOIDS= 3	

SWITCHES, FEEDBACK	SWITCHES, TRANSMIT
Terminal Ready	Character Select (6ea)
Terminal Busy	Parity Check
Busy (Function Interlock) 3 ea.	Character Sample
Case Sample (upper/Lower)	Function Sample (3ea.)
	Shift Sample
	End of Line (optional)

The next decision to be made is which of these two printers should you select as an I/O device. To bias your selection, let me say now that for price, effort of conversion, longevity and reliability, I recommend the Selectric I/O mechanism. It is designed with the following features, and to devise your own for adaption to an office selectric you must design similar features and create them yourself.

FEATURES

OFFICE SELECTRIC	I/O SELECTRIC
MOTOR/DRIVE	
SHADED POLE MOTOR (torque)	CAPACITOR START MOTOR
CENTRIFUGAL CLUTCH	DIRECT DRIVE
CYCLE SHAFT	
3 SELECTION CAMS	3 SELECTION CAMS + 2 Txm Cams
SMALL BEARING AREAS	LARGE BEARING AREAS
UN-TAPPED SHAFT	TAPPED FOR TIMING EXTENSION
CHARACTER SELECTION	
ONE PIECE LATCHES	LATCHES + EXTENSIONS for TXM
KEYBOARD	
MECHANICAL LOCK ONLY	MECH + ELECTRO-MECH LOCK
SHIFT/CHARACTER INTERLOCK	SHIFT/CHAR + CHAR/SHIFT INTERLOCK
ALL SELECTIONS PRINT	NO-PRINT SELECT PROVIDED
NO PARITY ENCODING	PARITY ENCODED
MECH. TAB, POWER SPACE	POWER TAB, NO-PRINT SPACE
OPERATIONAL SHAFT	
LIGHT (small) BEARINGS	HEAVY (large) BEARINGS
LIGHT TORQUE LIMITER (CR)	HEAVY TORQUE LIMITER
COMMON SPACE/ BACK-SPACE CAM	COMMON TAB/BACKSPACE CAM
SHIFT	
MECHANICAL ONLY	MECH & ELECTRO-MECH SHIFT
MECH SHIFT LOCK ONLY	MECH & ELECTRO-MECH SHIFT LOCK
SMALL SHIFT CLUTCH	LARGE SHIFT CLUTCH

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 14 **April 1977** **Volume 2, Issue 4**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

(BE IT EVER SO LATE)

Washington Club Offers Free Access to Their On-Line Journal 3

Pros & Cons of Modems for Cassette Recording 6

Using an Acoustic Coupled Modem as a Bulk Storage Interface 6

SOFTWARE

MITS Basic, Poly Basic, and NIBL – *Product Review by Jef Raskin* 8

Screen Image Hard Copy for the VDM-I, *Dan S. Parker* 10

Here, Now: A Micro APL for 8080's, *Erik T. Mueller* 12

Guide to 8080 PILOT, Version 1.1, *John A. Starkweather, Ph.D.* 17

Complete User's Manual (complete source code to be published in the May issue)

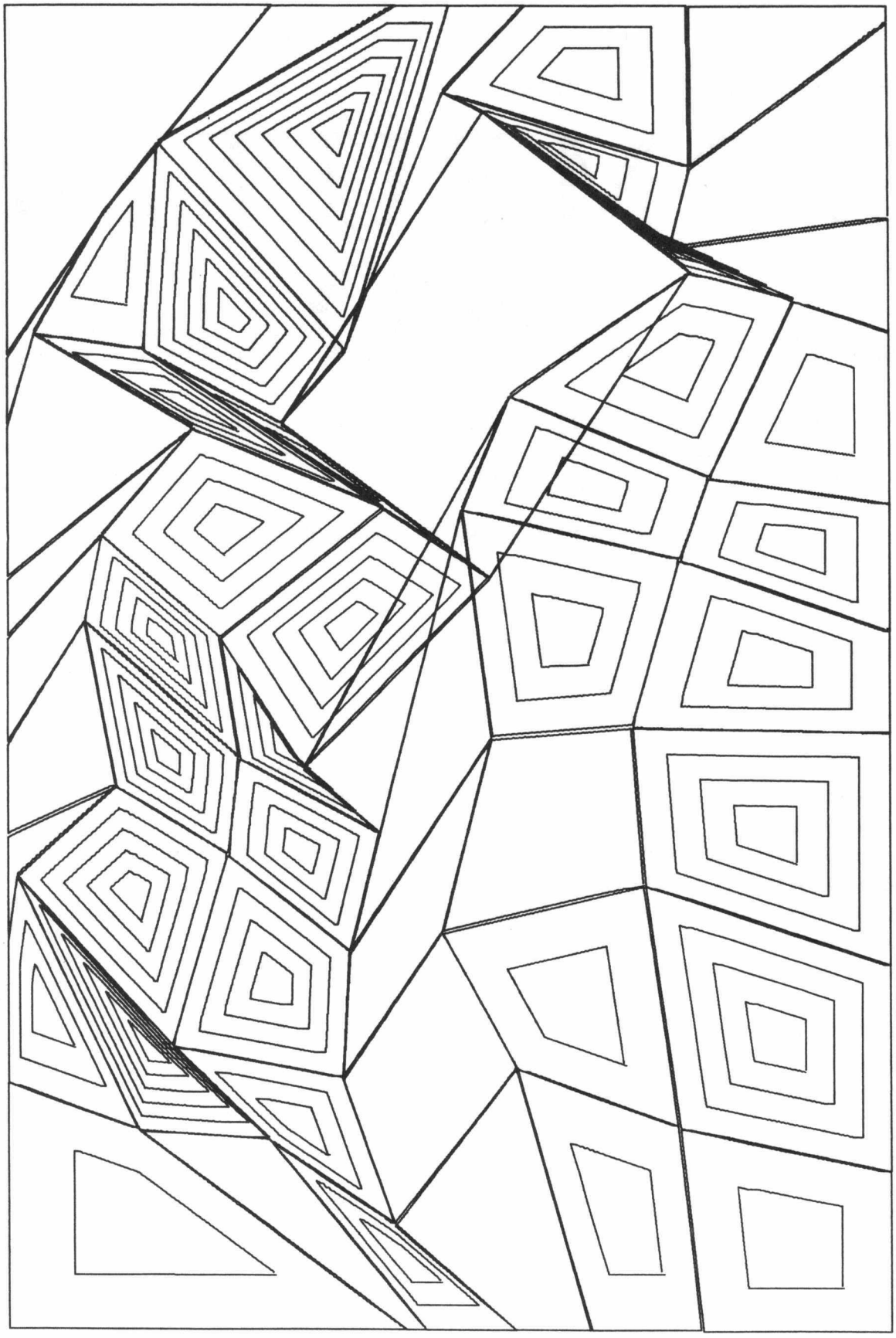
A 650X Program Relocater, *Ralph Sherman* 30

A KIM-I TV Sketch Program, *Stan Ockers* 32

DESIGN CONCEPTS

The Characteristics and Decodability of the Universal Product Code Symbol, *D. Savir & G. J. Laurer* 36

. . . and, a more-than-usually confused potpourri of interesting tidbits



© David Dameron 1979

WASHINGTON CLUB OFFERS FREE ACCESS TO THEIR ON-LINE JOURNAL

The Washington D.C. Amateur Computer Society is proud to announce the publication of the first "Electronic Journal" available to the general public in the United States.

The Journal of the Washington Amateur Computer Society, now in its second year of publication, is available via a computer link, to anyone possessing a computer terminal and telephone coupler.

NO "PASSWORD" OR ACCOUNT NUMBER IS NECESSARY.

Our Society Journal is written, edited and published once a month on the DECSystem-10 of the Catholic University of America, and recently we were granted permission to store the Journal as a "Free Access" text file. Your readers may obtain it by dialing 202-635-5710 for a 110 Baud line, or 202-635-5730 for a 300 Baud line. The DECSystem-10 will respond with its answer tone. Activate your modem and then type the following:

C

(i.e., "Control-C": Hold down the control key and type a "C"). The computer should echo an Up-arrow and a "C". If it does not, try doing it several times, if nothing happens you may have picked a bad time to call, when the computer is either "Down" for maintenance, or has "Crashed" because of some problem. If this is the case, try calling again later. If you get the proper echo (The Up-arrow and the "C"), type:

I

followed by a carriage return. The System Monitor will respond and tell you who, what, when, and where it is; something like this:

CATHOLIC U. 507B20 19:39:01 TTY60 SYSTEM 95

which is something like a Radio Amateurs Callsign. After typing this, the monitor will type a period on the next line, indicating that you are conversing with the Timesharing Monitor, and that it is waiting for your instructions. To get the Journal, type:

HELP WACS

followed by a carriage return, and the computer will proceed to type it out.

The Journal is rather lengthy (The April issue came to 16 pages), and is set up to fill pages 60 characters wide with 66 lines per page. If you are going to read it on a TVT display instead of printing it on a hardcopy device, we recommend that you tape the telephone output so that you can replay the Journal at your convenience. You can stop the output at anytime by typing a "Control-O" (Hold the control key down and type an "O"), which instructs the Monitor to stop output.

We believe that this is the first such electronic "Mail Service" or Journal. You will not be working with a recording, but will be running a program on the Timesharing System that moves information stored in a Disk "File" to a serial communications port. We would not be able to share this new adventure in communication with you, if it were not for the kind assistance of the Computer Center of the Catholic University of America. Since this service uses computer time, slowing the system down slightly, and ties up a communications port, we ask that you use it only during non-business hours; late at night or weekends would be ideal.

Our *Journal* is also available in printed form, that differs from the "On Line" version only slightly (If we have "Snoopy" on the cover page it is omitted from the electronic version).

We hope that this will be the first of many such computer hobbyist communication Nets, and that the age of electronic mail and newsjournals will not be far off. We appreciate the fact that Long Distance telephone rates will make our service rather expensive for the individual, let's say, in California, but perhaps that is where hobbyist clubs can show their worth, and make use of unused business "WATS" lines late at night or on weekends. We wish you all luck in your endeavors, and hope that a second such "Electronic Journal" is in the works out there. Perhaps this is just the kind of justification for the adoption of "ASCII" as a legal transmission code on the Amateur airwaves that the FCC will recognize. We computer hobbyists need access to quick, efficient and up-to-date communications media to enable us to grow and exchange ideas. Why can't we have access to radio communications in the code that is a national standard not only to our hobby, but also to ANSI? Computer hobbyists unite! Petition the FCC to allow us to communicate in our "native tongue". Meanwhile, find those fallow WATS lines and make them serve the hobby.

Sincerely,

Robert Jones, Director — Thomas Crone, Asst. Director
Richard Bonanno, Editor — William Stewart, Assoc. Editor
Washington Amateur Computer Society
4201 Massachusetts Avenue, #168
Washington, D.C. 20016

FIRST MICROPROCESSOR-COMPATIBLE ANALOG INPUT COMPONENT ANNOUNCED

With Burr-Brown's new MP20, microprocessor system designers now have a complete, bus-compatible solution for analog data acquisition tasks.

This hybrid, quad-in-line package consists of a 16-channel analog multiplexer, high-gain instrumentation amplifier, 8-bit A/D converter, plus all necessary address, data- and control-bus interfaces.

The MP20 is timing and logic level compatible with 8080A and 8008 type microprocessors. No external logic is needed. And gain and offset are internally laser-trimmed, eliminating the need for external adjustments while providing absolute accuracy better than ±0.4% (1 LSB) on the ±5V or 0 to +5V ranges.

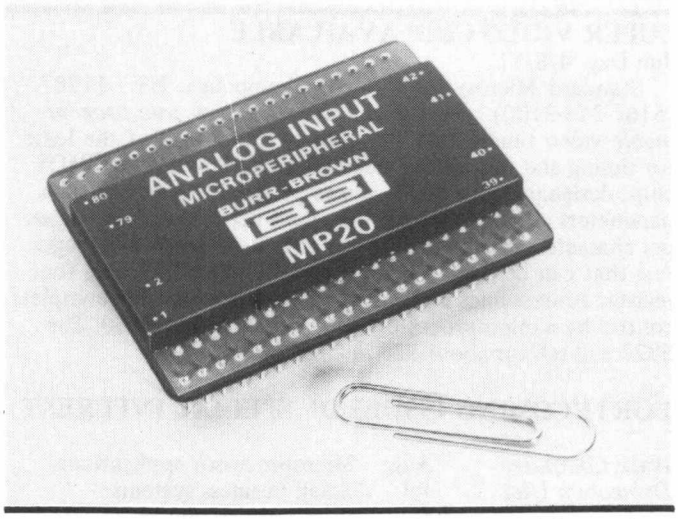
Low-level signals such as thermocouple outputs can also be handled directly with reduced accuracy. The instrumentation amplifier can be programmed with a single external resistor to provide input signal ranges as low as ±10 mV FS.

To simplify programming and allow for unlimited channel expansion, the MP20 is treated as memory. Each analog input channel occupies one memory location. Any memory reference instruction can be used to access data. Thus, one LDA instruction will input data from one channel to the accumulator. Two adjacent input channels can even be acquired with one LHLD instruction. Alternately, the MP20 can be interfaced as I/O.

Housed in a 1.7" x 2.1" x 0.15" ceramic package, the MP20 is specified over a 0° to +70°C temperature range. Power requirements are ±15VDC and +5VDC.

Price of the MP20 is $195 (1-9) and $140 (100's). Delivery is from stock.

For more information, contact C. R. Teeple, Product Manager, Burr-Brown, International Airport Industrial Park, Tucson, Arizona 85734. Phone (602) 294-1431.



LETTER TO THE EDITOR ABOUT AN EDITOR

Dear Sir: December 9, 1976

Enclosed is a preliminary copy of the operating instructions for EDIT. **[Indeed it was. And a reasonably hefty rough draft, at that. –Ed.]** EDIT is a versatile text editor which I have developed for the 8080 microprocessor. EDIT is very similar to 'big machine' text editors, but more powerful than existing editors for hobby and small business applications. EDIT may be used with any type console, and simple patches may be made to interface video module handlers. The input to, and output from, EDIT can be directed to any peripheral with appropriate handlers. Complete peripheral interface details are included in the EDIT manual. I plan to sell this software for around $25 a copy (not including source), but would like to hear your comments, suggestions, offers, etc. before proceeding.

EDIT is now up and running and available for demonstration. I should soon have a simple Tarbell cassette interface handler available. I hope to have floppy handlers for the more popular systems.

The EDIT manual *enclosed is not for public distribution or reproduction.* **[How sad. - Ed.]**

Regards,
Lorin S. Mohler
219 N. Sweetwater St.
Anaheim, CA 92806

SUPER VIDEO CHIP AVAILABLE

Jim Day 4/8/77

Standard Microsystems Corp., Hauppauge, NY 11787 (516/ 273-3100) recently announced a new *user-programmable* video timer-controller chip containing all of the logic for timing and formatting video displays. This 40-pin NMOS chip, designated the 5027, allows program control of such parameters as characters per line, lines per frame, raster scans per character, etc. Control of the chip is via seven 8-bit registers that can be loaded from a parallel I/O port, using four register address lines and a chip-select line to provide complete control by a microprocessor such as the 6800 or 8080. The 5027 will sell for about $26 in OEM quantities.

FORTHCOMING ISSUES OF SPECIAL INTEREST

IEEE *Computer*	Aug:	Microprocessor applications
Datacomm User	Jul:	Small business systems
	Sep:	Intelligent terminals
	Dec:	Specialized terminals
Data Management	Dec:	Mini/Micro update
Datamation	Jul:	Personal computers
Digital Design	Oct:	Microcomputers
EDN	Nov:	Microcomputer directory
Electronic Business	Jun:	Microprocessors
Electronic Design	Jul:	Microprocessors
	Oct:	Microprocessors
Electronic Engineering Times	Aug:	Microprocessor update
Electronic Products	Oct:	Microprocessor power supplies
	Nov:	Microprocessor software
IEEE *Spectrum*	Jun:	Microprocessors
	Dec:	Microprocessors
Infosystems	Jun:	Word processing
	Aug:	Voice recognition
Minicomputer News	Jun:	Personal computers
	Sep:	Word processing
	Oct:	Small business systems
Mini-Micro Systems	Jun:	NCC issue
The Office	Jul:	Small business computers

FASBOL, ANYONE?

Dear Dr. Dobbs:

I'm looking for a hard copy of both the program and documentation for FASBOL II, a SNOBOL compiler for the PDP-10. This is available from DECUS only on magtape (at a price of $40.). I don't have a PDP-10 anyway and my interest in the program is to rewrite it for a smaller system which I'll share with *DDJ* readers, if successful. I hope someone can help me and I'll answer all letters.

Thanks,
Ron Tipton
Route 7
Fayetteville, Arkansas 72701
(501) 643-2191
P.S. Permission is granted, of course to publish this letter!

PRAISE FOR WAVE-MATE & RAINBOW COMPUTING

Dear Mr. Warren: February 19, 1977

As a recent convert to computer mania, I have read and heard many stories of real horror concerning people who have waited months for delivery, spent weeks troubleshooting, and who have gathered grey hair and ulcers corresponding with uncooperative manufacturers.

Perhaps my experiences, rather different, will be of interest to you and to others, and at least will give credit where it is due. After visiting several computer stores, where, in the main, I was accorded treatment usually reserved for a mentally deficient, I started correspondence with several manufacturers, and finally made the acquaintance of Dennis Brown of Wave-Mate in Gardena. Alone of the persons I contacted, he answered patiently and fully my many questions, showing me anything I wished to see, from parts to assembled boards, from manufacturing facilities to operating systems, even introducing me to customers who had computers up and running. I learned a lot fast, and relatively painlessly.

After helping me get my rig going, mostly by building it and doing all the troubleshooting, he turned me over to Gene Sprouse of Rainbow Computing, Granada Hills. Gene, Dick and the gang are enthusiastic, immensely knowledgeable, and about the most helpful people I have ever run into. They took about a month to bring me from benighted ignorance to the point where I am writing useful programs for my business. I am also enjoying what could easily be an expensive and frustrating hobby, and associating with people who are alive and enthusiastic. Everyone associated with Wave-Mate and Rainbow Computing has been helpful "far beyond the call of duty" and has earned my gratitude and recommendation. A list of the things they have done would prolong this letter unbearably, but for example, they have stayed late at night to tutor me, come to my shop to deliver a memory board, and answered at length questions that would have taxed a saint's patience. Frankly, I would rather eat a copy of *DDJ* than write a letter, but such integrity certainly deserves comment.

I am more than pleased with my Jupiter II. It will be a useful and labor-saving adjunct to my business, as well as an interesting and educational avocation. I hope to use it not only to do paperwork, but also to carry out some experiments in computer music.

I will be glad to answer any questions concerning either my system or experiences, and to talk with businessmen or musicians concerning applications.

Sincerely,
M. J. Kreinik
Aeolian Music
4323 Melrose Avenue
Los Angeles, CA 90029

"SERIOUS APPLICATIONS IN PLEASANT HILL"

November 1, 1976

Diablo Valley Junior College in Pleasant Hill, California, is home base for an unusual microcomputer group, an association of professionals interested in "serious" applications for small computer systems. Serious means wanting to make some system in society work better, smoother, more safely, with less loss, pain, frustration, energy expenditure, or delay. It can also mean any computer application which increases one's personal income.

We hereby extend an invitation to like-minded people across the country to join in a network of professionals dedicated to information exchange re small systems and their "serious" applications.

We are not particularly interested in organizational clap-trap but rather in DISSEMINATION OF BOTH HARD DATA AND INNOVATIVE FANTASY, REGARDING TECHNICAL FEATURES OF SMALL COMPUTER SYSTEMS – HARDWARE AND SOFTWARE.

As an example of innovative fantasy, let me tell you about Dick Maus' project. He's a computer science teacher I met in Minneapolis this summer. His project: to regard the game of "Life" as a subset of what Dick calls "Superlife". This algorithm will treat growth and decline phenomena in cell arrays. Dick sees applications for his simulation studies in such diverse fields as auto traffic flow, patient queueing lines in a medical clinic, interaction of arrays of businesses in a given market, wave phenomena analysis, and nerve impulse transmission in neural tissue, just for starters. (Dick plans to communicate soon with Dr. Dobbs re the above. Of special interest is his plan to offer a $100 prize for the best program readers come up with.)

In conclusion, let me happily report that we've got Jim Warren's interest in this network thing, so please use this publication as a clearing house and sounding board. Do NOT write to me – I'm up to my ears in (computerized) alligators. Address all correspondence to: PROFESSIONAL USERS GROUP, Dr. Dobb's Journal.

William J. Schenker, M.D.

POTPOURRI FROM ZHAHAI

Dear *DDJ:*

1. I like the IL approach. It is very similar to the method that has been evolving for some software I'm working on (a monitor and assembler for the 6502), and not I begin to wonder if I cannot make the two compatible, i.e.: use the same IL interpreter for TB and the monitor/assembler. There are some different ML calls, but there is plenty of room to define more in TB's IL (and almost unlimited room in Whipple & Arnolds' version). Will let you know how this works out. It seems to me that a well developed master IL (MIL) might be possible which would allow developement of editors, executives, assemblers, languages, maybe loaders, etc. Is this blue-sky?

2. How much slower should the two level approach be as compared to a one level approach? (Factor of 1.5, 3, 10, 40?) Any rough ideas?

3. Calculator chip extended function units are too slow!

4. I agree: 'nuff BASIC, SMALL PASCAL would be much interest. CASUAL and TINY HI are intriguing.

5. More references to trade & professional sources (literature).

6. Re using an acoustic coupler for cassette tapes: this may not work very well. See the analysis in *The Computer Hobbyist*, No. 5 (Box 295, Cary, NC 27511). Briefly, the "scuzziness" of a cassette recorder and of a phone line are of different natures. Also, who needs another 300 bps cassette standard? One major problem with the KC standard is that it's too slow, so everyone wants to use something faster for themselves (and two cassette interfaces is awkward to ask of folk). And for higher bit rates, you can be pretty sure that high speed modems (IF they work on a cassette at all) will be much more expensive than other (self-clocking) techniques, e.g.: Digital Group & Tarbell. And a final note for those who go ahead anyway (which I encourage as long as nobody forks out *too* much dough *until* this is tested): standard acoustic couplers transmit on one frequency pair and receive on another pair, i.e.: they won't read what they write. On *some* modems, the "self-test" mode will allow transmitting and receiving on the same pair. I believe the Motorola MC 14412 universal modem chip will allow this, also. (Likewise, the MC 6860 of the 6800 family has self-test). Good luck! It should work (at 300 bps on a decent cassette unit. Exchange may be pretty hard tho.

7. Considering the number of mods, errata, and replies in *DDJ,* an index is badly needed. Is there any plan for one? **[Yup.]** Also, how about deliberately prefixing articles with some keywords? **[Good idea. We'll try. In the meantime, use the headlines.]**

8. Good articles on double density for floppies in Sept., Oct., and Dec. issues of *Computer Design.* Re HLL, see *A general-Purpose Macro Processor as a Poor Man's Compiler-Compiler* by Andrew S. Tanenbaum in IEEE trans. on software eng., Vol. SE-2, No. 2, P. 121 (June, 1976). Of special interest is his description of SAL, a PASCAL-like systems programming language. I think it could be Tinied cleanly.

P.O. Box 1637
Zhahai Stewart
Boulder, CO 80306

RASKIN REJECTION, COPYRIGHT COMMENT & PRODUCT TESTING PROPOSAL

Dear Jim: January 5, 1977

Please do not "serialize" (Jef Raskin's product review) or use the stream of consciousness style of writing that is apparently "in" in computer magazines. As a literary work, such articles are atrocious. As a source of information, they are even worse, forcing the reader to wade through oceans of words to find the pearls of wisdom.

Please do not print any more about copyrights ("Postscript to 'Copyright Mania' ", p. 11). There is too much free interpretation of the copyright law around already, e.g. an unfortunate article in a recent *BYTE.* Copyright protection for software is a force fit at best anyway. The result is that there are as many opinions as there are lawyers or pseudo-lawyers. While I sympathize with the victims of corporate supposed fail-safe mechanisms, I think the ideas should not be spread around.

As a matter of interest, the National Commission on New Technological Uses of Copyrighted Works (CONTU) has issued a preliminary report. If you want information or would like to testify at or attend their hearings, contact Robert W. Frase, Assistant Executive Director, National Commisision on New Technological Uses of Copyrighted Works, Washington, D.C. 20558.

Concerning your new product testing program, instead of a voucher from the manufacturer, would you accept a certified check for the price of a piece of equipment chosen by a reader (and not previously evaluated)? **[We would certainly consider it.]** After testing, the equipment is forwarded to the reader. What if the equipment were in kit form? **[Ditto . . . if we can find someone we feel is qualified, whom we can afford.]**

Very truly yours,

165 Hurd Road
Paul F. Wille
Aurora, OH 44202

AN 8080 RESIDENT FORTRAN COMPILER . . . FOR $750

A resident Fortran compiler for 8080 microprocessors has been introduced by Unified Technologies, Inc. of Islington, Ontario.

The compiler runs on most 8080 systems using 16K bytes, handles a subset of ANSI Fortran IV. The program produces a formatted output, handles one to thirty-one character symbolic names and will perform single or double byte integer arithmetic. A floating point package is also available.

Priced at $750, the compiler is available on an IBM format floppy disc, paper tape or PROM. Documentation including both reference manual and user's manual, is available.

Linkage may be made between the firm's FORT-80 programs and assembly language subroutines. In addition, functions such as ABS, RANDU, and SQRT and included.

The compiler is being distributed in Canada by Datamex, Inc. U.S. customers may contact UTI directly.

RASKIN RANTS: ACOUSTIC COUPLER & CASSETTE RECORDERS

In Volume 1, Number 10 (you know, the one two issues after the famous mystery issue) Steve Moore suggested, through our beloved editor, that an Acoustic Coupler be used as a cassette interface. Steve's right, of course. But here I am in the middle of reviewing the Pennywhistle Modem for this very rag. Said Pennywhistle incorporates a feature handy for using the Bell 103 standard as a cassette interface. To wit: a pair of mini phone jacks from which and into which may be placed audio signals. First of all I have been happily using modems as cassette interfaces since long before the hobbyist computers were in gestation. The scheme works with a wide variety of acoustic couplers and cheap cassette recorders. I have often read in the hobbyist publications that the reason the Bell 103 FSK conventions are *not* used is that inexpensive tape recorders will not hack it. Well, they will. Anything over $50 that I have tried over the last six years does the job.

Now, as is my custom, for some snags in the grass. If the batteries in your tape recorder are dying and its speed is slipping a bit then the 103 recording won't work. Nor will any of the other methods even if they have a bit more speed tolerance. The Pennywhistle modem is particularly good in accomodating frequencies off standard. It tracks. But the 103 convention is really two conventions. One pair of frequencies goes one way and another pair of frequencies goes the other way. This is fine when you want to full duplex (go in both directions at once) over a phone line. But it causes a problem in that we must all agree as to whether our cassettes are recorded in ORIGINATE or ANSWER mode. I used to have an old Prentiss Electronics coupler that could handle both. The Pennywhistle is a bother in this regard. But see the article on that gadget for more details.

ACOUSTIC COUPLERS & CASSETTE RECORDERS: FANTASY REALIZED

Dear Jim Warren: January 9, 1977

I have already realized one of your fantasies – using an ordinary modem to generate tape bulk storage. It represents the fact that I had (and still use) a modem to access a few IBM 360/91 computers, a pair of tape decks, an XR210 that didn't work out in a related project and no money. Together, these spelled either no storage or your fantasy. While I don't think it's a great solution, it fit my budget since I already had everything. The enclosed pages should allow anyone to more or less duplicate my effort. With the wide variety of surplus modems around, it would be hard to be more specific about the modem connections. The modulator could also be connected at the computer end of the link and this might be easier since most hobbyists probably have circuit diagrams and drawings to locate voltages and signals. Also, some noise problems would be reduced if the playback signal were electrically connected rather than acoustically as outlined but this requires finding a suitable spot in the modem circuit and may also disable the telephone usefulness. One more detail, computers and modems are usually both wired with a DB25S connector so to connect them together, you need to make an adapter cable with DB25P connectors on each end wired at least for the following connections (by pin numbers): 1-1, 2-3, 3-2, 7-7. Note the connections can also be made: 6-20, 20-6, and 8-8. This crosses the two ready lines and allows the computer to sense carrier detect if it has the signal (MITS 2SI0 does). With this cable, both the computer and modem will be connectable to the terminal as well as to each other by preserving all the standard wiring connections to the plugs and sockets. For 20-ma current loops the same kind of thing can be done, again wiring a cable to crossover the data lines when connecting computer to modem.

Helpfully yours,

Darrel J. Van Buer (and Hymie, my computer)
1516 Electric Ave., Apt. R
Venice, CA 90291
(213) 823-6016

USING AN ACOUSTIC COUPLED MODEM AS A BULK STORAGE INTERFACE

There are only a few problems with doing this. The first problem is that the technique used to store the information was designed for telephone circuit characteristics. For all the bad things about a telephone link, it gives very little frequency distortion. The data is encoded by frequency shift keying between two frequencies less than 10% apart so that a cheap recorder may have enough flutter and wow to deceive the demodulator. Depending on the circuit details of the demodulator, it may not reliably tolerate more than 1 or 2% errors. The solution of this problem is to use a good recorder. A good discussion of this issue can be found in *The Computer Hobbyist*, Vol. 1, No. 5.

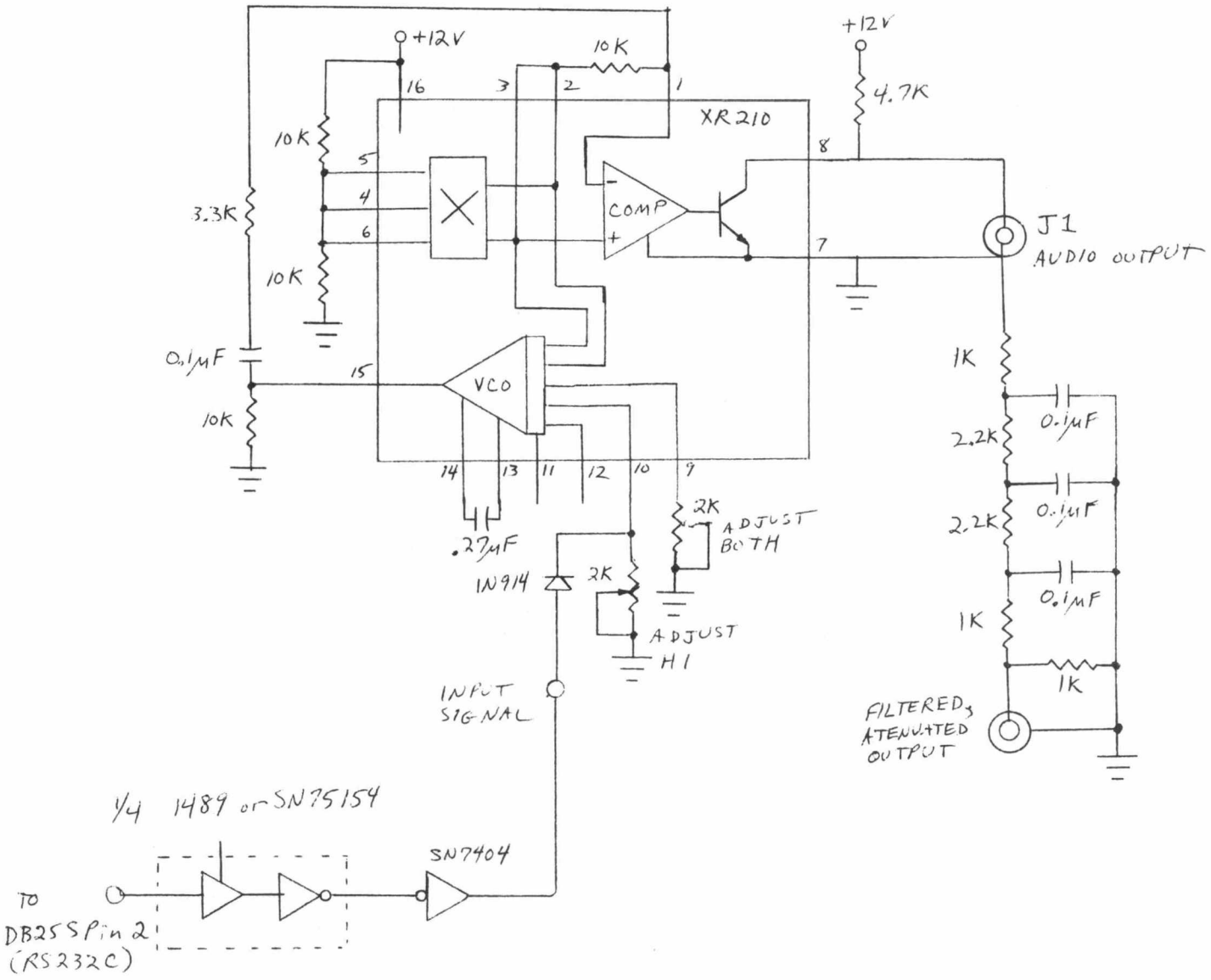
The second problem is that the modem transmits on one set of frequencies and receives on another. It would be possible to retune the transmitter frequencies, but this would disable the usefulness over the phone, and still require disabling the answer-back delay. The easiest solution is to build build the modulator shown in the attached circuit diagram (for about $10).

This modulator circuit is taken from the Exar application notes for the XR-210 IC used. Opening the case of your modem, you should look around for a place to put the circuit (it easily fits on a two inch square bit of perf board). A little bit of poking around in the neighborhood of the power supply should yield a place to steal the +12 volts) and +5 volts for the RS232 receiver and inverter. If the modem uses TTL, DTL or RTL, the signal for the XR210 may also be there but might be hard to find. Both ground and the transmit data are easy to find as both are on the 25 pin socket. Ground is on pin 7 and the data is on pin 2. In most

cases, the filter/attenuator will be needed because the raw output is supply voltage square wave which will badly overdrive most tape decks. In any case add an audio output jack to a handy spot on the modem case. This output will be active whenever the modem is turned on, and should be connected to the recorder input. The simplest way to get the recorder output back to the modem is to wire a small speaker to the earphone jack of the recorder, then set the speaker in the modem earpiece cup.

Tuning the modulator can be done either of two ways. By far the easiest way is to connect a frequency counter to the output. The output frequencies are keyed by using your computer or terminal, high frequencies normally, low frequencies when you send a break condition. The frequencies should be within 10 cycles of 2025 Hz and 2225 Hz. An alternate adjustment method is to use the tape recorder monitoring circuit to feed the modulator output to the demodulator and adjust the frequencies until they activate the carrier detect circuit and give the same data received as sent. Note that one of the adjusting potentiometers adjusts both frequencies so the low frequency should be set first, then the high frequency set separately.

General comments: If you don't have (or want) a modem to talk to remote computers and a good deck already, your money would be better spent on one of the speed-tracking interfaces (Tarbell, TCH, Kansas city standard). The original version of the Altair ACR interface used this 2025/2225 modulation but they have since widened the shift to better discriminate them despite speed errors. The data rate possible will vary somewhat with different modems but most were designed to operate reliably at 300 baud. Any higher rate would depend on a top rate modem and recorder. I have been getting adequate results at 300 baud but I would recommend generation and checking of parity on all data recorded and the playback of new recordings to ensure their readability, at least until you have a good feeling for the reliability of your setup. If you use a three head deck, it is possible to use a full duplex modem setup to verify the data a fraction of a second after it is recorded by reading and writing the computer port simultaneously. It is also a good idea to record at least enough leader of no data to activate the carrier detect circuitry as some modems inhibit the reception of data until there is a carrier for several seconds. I have successfully used this method for about six months, mostly due to the usual student poverty situation, but a high speed alternate is slowly nearing the top of my priorities.



MITS BASIC, POLY BASIC, AND NIBL
A Review by
Jef Raskin

It takes a while to do a software review. Many pieces of hardware can be built and checked out in a few hours. The instruction manual has to be gone through but once, and any problems or good points noted. The device can be objectively tested against the published specifications. The reviewer can note any conveniences and difficulties in operating the device. The case size can be measured with a ruler and reported. All nice, sensible tests. But software is another matter entirely.

To be truthful there is software that *is* easy to review. Software that doesn't work at all is trivial to report on. A short program that does floating-point division can be tested quite readily for example (although there are tricky points). But a compiler or interpreter, an operating system or a text editor – these require having the system up and running for a while. Sitting around and *using* the program for a while. There are sure to be corners of the software under test that just don't get observed. A decent BASIC manual is not a pocket size pamphlet, and testing every feature, in every combination with every other feature, is just *not* possible. Even just testing every feature may be prohibitively time consuming.

My method has been to get the software under consideration up and running, and use it for a month or more. Occasionally I jot down a comment or two, and gather a general impression. After familiarity has bred some contempt I start to get nasty, and pick at the weak points. Some software can be brought crashing to its knees by doing operations that seem to be permitted by the manual, but in fact don't work that way at all. Sometimes software is just wrong.

After looking at the software from the user's point of view, I try to look (at least a little) at the implementation itself. Occasionally a few things will reveal themselves at that point. Such as obscure coding that might make it difficult to modify a system for future use, such as bad documentation, such as clever system design that makes it easy to add your own flavors to the basic recipe.

In reviewing personal computing languages, one would be negligent in not paying part of the debt we all owe to MITS BASIC. Written by a firm called MICRO-SOFT, this was the first widely available BASIC for personal computers. It wasn't (and isn't) perfect but was very well thought out for personal computer use, and brought with it many significant innovations that have set the tone for all following BASICs for home use.

MITS BASIC is not slow. One of the first systems I sold to a commercial user [I do systems integration and software design for microcomputers professionally] was an original ALTAIR running MITS BASIC. One day a customer wanted to run a benchmark on it against the IBM 5100 (at $20,000) and the similar Tektronix machine (at about $12,000). "Fine," I said, not knowing whether I was going to win or lose, having never run such a test myself. The customer typed in the 200-odd line program that did statistical analyses of stock market data. The IBM machine had run the job in 112 seconds. The Tektronix in 113 seconds. When the program was entered I made a small handful of changes to make it run on the MITS computer. It took a few minutes. No longer than it took, according to the customer, to change from IBM to Tektronix's BASIC. So I was still in the running. With stopwatch in hand he typed "RUN" and the program began. He sat back to wait and was startled when results started printing out. The answers were correct. And the program had run in 57 seconds, beating the more expensive competition very handily. I had a customer.

This alone should establish that MITS BASIC is no toy. But it was some of its unique features (at that time) that originally attracted my admiration. Improving on most earlier BASIC's, Micro-Soft had invented (with wonderful aptness and memorability) the PEEK and POKE keywords. These allowed direct memory access in BASIC. For the first time, one could write an assembler in BASIC! In fact we quickly wrote a disassembler in BASIC to find out what the BASIC interpreter looked like. Some other BASIC's use "EXAM" and "FILL" for these functions, but MITS' terms are the better and should be adopted universally. Polymorphic System's BASIC originally used "EXAM" and "FILL" but by their released version they had decided to use the de facto standard. Bravo. NIBL, a version of Tiny Basic concocted by National Semiconductor for their SC/MP system uses a strange notation for this function. An at sign (@) before an expression means the contents at the location whose address is the value of the expression. (An expression is a variable, a constant or some arithmetical combination of variables and/or constants that can be evaluated to a single value). While this accomplishes the same goal as PEEK and POKE it is *not* in keeping with the design of BASIC. BASIC is a keyword style language, and does not use obscure symbols to do functions. In this reviewer's opinion NIBL makes a very wrong choice at this point.

For memory mapped I/O, where putting a value in a certain location in memory automatically sends it to a peripheral device (and the device transmits to the computer by having a word or two in memory that it can control) PEEK and POKE would serve for explicit control of input and output. On less elegant architectures (such as the 8080) there are separate I/O instructions and thus MITS BASIC has the IN and OUT instructions. These are excellent additions to BASIC, and Micro-Soft did it well. On the other hand the WAIT instruction (which waits for a device to be ready, as indicated by a bit going on or off in a status byte) is rather clumsy. But it is useful, and MITS was the first to provide it, again doing the personal computer field a true favor. In Polymorphic BASIC the user has to write the wait loop themselves. Considering the obscurity of the WAIT instruction (when three arguments are required) it is often a good idea to write out waiting loops explicitly in MITS BASIC to improve readability and to make them interruptible!

Polymorphic BASIC, which came out much later than MITS BASIC has the advantage of hindsight and is generally more powerful in consequence, having many useful features such as plotting and a real-time clock. On the other hand one of the most important features of MITS BASIC was copied only superficially and is a major problem to the serious user of Polymorphic's version. One of the most useful innovations in MITS BASIC was the bit-wise AND, OR and NOT. These treat a number in terms of its *binary representation.* For example 9 (base 10) in binary is 1001. 10 (base 10) translates to 1010 in binary. So, 9 AND 10 evaluates to 8 which is merely the bitwise ANDing of 1001 and 1010. Similarly 9 OR 10 yields 11.

```
    1001        1001
    1010        1010
AND ====    OR  ====
    1000        1011
```

This may seem a bit strange, and useless: unfortunately for some readers (and fortunately for others) this article is not the place for a tutorial. The other operation NOT merely turns all ones into zeros and vice versa. These operations are vital for many control applications. Polymorphic BASIC, while it has AND, OR and NOT, does not do them properly bit-wise, thus destroying much of their utility. This reviewer called up

Polymorphic Systems and they have the change to true bit-wise operation "on the back burner". NIBL *does* do these operations bit-wise. But there is a disastrous error in NIBL. False relationals (such as 4⟩9) evaluate to 0. True relationals evaluate to 1. This seems reasonable, but in a bit-wise system with twos complement arithmetic, true *must* (as it does in MITS ville) evaluate to –1 (which in binary is all 1's). Thus, in NIBL, while negating something false gives something true – which is correct – NOT (true) gives *true* instead of *false*. Oops. I called up National Semiconductor about it, and they didn't think that it was important enough to make the necessary changes, although one of the implementers said that it was "an easy patch". So, NIBL users beware. The statement

```
IF NOT (2 = 2) THEN PRINT "TWO DOES NOT EQUAL
TWO"
```

prints

```
TWO DOES NOT EQUAL TWO
```

just as though the computer believes its. Poly and MITS BASIC both give the right answer. [Kudos to Doug Wyatt for this neat formulation of the problem with NIBL]

MITS' BASIC has proved to be fast and reliable, with very few bugs. On the debit side is the manner in which it was coded. The programmers (Bill Gates et. al.) were into tricky coding. This has come back to haunt them as MITS BASIC is one of the few programs known that will *not* run on the newer Z-80 CPU's. Also their I/O routines are not localized. They tend to be here, there, and everywhere. This makes it quite difficult to modify the code to work with input and output devices not specifically provided for by the designers. Poly BASIC is just the opposite, being extremely clean in regard to I/O handling, and the coding (what I have seen of it) seems quite straightforward and not at all convoluted.

Another advantage Poly's BASIC has is arithmetic precision. Besides a few more decimal places, the routines were done with their numerical analysis straight. There are many occasions when MITS' BASIC loses precision unnecessarily. Your faithful watchdog called MITS on the phone, and followed it up by a letter to inform them of the problem. The only reply I got back was that the loss of precision is "normal." The examples were additions of four significant digit numbers, two digits of which were to the right of the decimal point. The sum was non-zero in the third decimal place. Another example was three to the fourth power, which was a smidgen over 81. Sure, in digital arithmetic there will be rounding error due to the finite precision, but many algorithms published quite a number of years ago do better than MITS' BASIC, given the same internal precision. Poly BASIC does it right. NIBL only has integers, so there is no problem. Now for something Poly BASIC has got that everybody oughta have got. Bad English to introduce some Good Computer Language. Many full flowered BASICs have the ability to define functions with a DEF statement. So does Poly BASIC. But! The functions are genuinely re-entrant. This means that they can call themselves. Is this useful? Sometimes. Consider this function that calculates the factorial of an integer. (Old Hands at computer science will recognize an old standard example here).

```
100 DEF FNF(X)
110 IF X=1 THEN RETURN 1
120 RETURN X*FNF(X-1)
130 FNEND
```

It works like this. One factorial is one. So if X=1 line 110 will provide the correct answer. Say X=2. Then line 120 comes into play. There, X (which is 2) is multiplied by this very same function applied to X-1 (which is 1). But FNF applied to 1 is 1, and 1 times X (which is 2) is 2, which is returned (and is the correct answer). Try it yourself for X=5. 5 factorial should be 5*4*3*2*1 or 120.

Avoiding the temptation to make this a tutorial (there, that was easily done) I will just mention that variables used as arguments are *local* to the function. Any other variables are global, which means that they behave just like the BASIC variables you are used to. Thus a program that will work in non-Poly BASICs will work in Poly BASIC unchanged. But the astute programmer knows the advantages of having local variables in a subprogram. If you are not so astute then ask someone who is. Local variables in subroutines would be a nice addition as well, but that has yet to be implemented in any BASIC that I know about.

Poly BASIC has an INPUT statement that allows printing a literal string instead of the usual automatic question mark. That is a fine feature, but their keyword is "INPUT1". Not very mnemonic. It is also unnecessary: INPUT "MESSAGE", X should suppress the question mark. INPUT X would behave as always. The quote would tell BASIC what to expect.

PLOT is another feature that I like, but it depends on having the Polymorphic Systems video interface, with its graphic feature. The statement

```
PLOT X, Y, 1
```

puts a small rectangle at location X, Y on the screen.

```
PLOT X, Y, O
```

deletes such a rectangle. PLOT can be used to position text anywhere on the screen as well.

Both MITS and POLY have a good selection of string handling features. The CHR$, VAL, STR$ and ASC functions are handy. CHR$ returns the character corresponding to an ASCII code. ASC returns the ASCII code of a given character. VAL returns the value of a character string that represents a number, and STR$, given a number returns the character string representing that number. These form a necessary basis for serious string handling in BASIC.

The Poly-88 includes in every system (unlike the other manufacturers) a real-time clock. BASIC can access it with the TIME function. Thus BASIC can be used for doing accurate timing. The clock is useful in computer assisted instruction, games and real-time control applications. It is also easy to write a digital clock that appears in big numbers on the screen when the computer isn't doing anything else.

Ever since MITS showed the way, BASIC has had to have a way of linking to a machine language subroutine. But Poly's format for the linkage is neater, and is easy to describe: CALL(X,Y). The second argument is optional and is converted to an integer and put into the DE register. X is the address of the machine language subroutine. When the machine language *return* is executed the current value of the HL register is returned to the program.

One of the greatest deficiencies of BASIC is the unmnemonic quality of variable names–A through Z and these letters followed by a single digit. Yech. MITS' BASIC allows names up to eight characters in length albeit that only the first two are checked (that is the names NEAT and NECTAR refer to the same variable). This is better than one-letter names, and makes programs easier to write and read. Longer names *should* be allowed in all future BASICs. Note that existing BASIC programs would still run *verbatim* if this change were made. But this article isn't supposed to proselytize for new BASIC features, just report on existing ones.

To sum up, then: MITS' BASIC was a valuable addition to the personal computist's arsenal, and some of its better features have yet to be picked up by the other companies. Poly BASIC has most of the advances that MITS introduced, added some of its own, and missed a few. NIBL, as a Tiny Basic (National Semiconductor's manual references *Dr. Dobbs* as the original source of Tiny Basic–even the big companies read this magazine!), is nowhere near as powerful as big BASICs. It also has one annoying flaw as mentioned. To be

fair to NIBL it has an innovation that might become popular—a DO . . . UNTIL construction. Whatever statements are between the DO and the UNTIL instructions get executed repeatedly until the expression following the "UNTIL" becomes non-zero. The manual has this neat program for finding prime numbers:

```
10 PRINT 2
20 I=3
30 DO
40    J=I/2:N=1
50    DO
60       N=N+2
70    UNTIL (MOD(I,N)=0) OR (N J)
80    IF N J PRINT I
90    I=I+2
95 UNTIL 0: REM THIS REPEATS OUTER LOOP
     FOREVER
```

This is presented mostly without REMarks so that the avid puzzle solver can figure out how it works.

So even a Tiny Basic can have something that the bigger versions can learn from. And the full microcomputer BASICs are already better than the prototypes from the large computers and the minicomputers. The personal computer movement is not only proving how much less expensive computer power can be, but is showing the way in new language ideas as well. Look out, IBM, look out, DEC. Or better still: Look and learn.

4K BYTE MEMORY CARD AVAILABLE FROM NATIONAL SEMICONDUCTOR

Santa Clara, Calif.— A compact, low power series of NMOS semiconductor memory cards offering system engineers a ready-made alternative to the expensive design-prototype-test cycle is now available from National Semiconductor Corporation.

Known as the NS 400-N series, this self-contained 4K-byte memory utilizes the model MM2102 1,024-bit static RAM as storage medium. According to Don Miller, OEM memory systems marketing director, these storage elements eliminate the need for clocks and refresh signals, and keep the power consumption low. The cards are available in 8, 9 or 10 bit versions and in 550 or 300 nsec. access times.

"Simple interface and high performance make the NS400 series ideal for microcomputer and microprocessor applications where overall cost is an important consideration," Mr. Miller said. The space-saving (3.93 x 6.3 x 0.5 in.) 4K x 10-bit, 550-nanosecond version of the NS 410 sells for $200 in quantities of 100, and is available for immediate delivery from National Semiconductor Corporation, Santa Clara, California, or from National's franchised distributors.

All that is needed to use the NS 400-N cards are memory enable, address, read-and-write, and data, all of which are TTL compatible, and a single 5-volt power supply with one Ampere for each card. Independent 3-bit addressing for each card permits plug-in expansion to an 8 card, 32K word memory system. The card address is programmable via the connector wiring.

Data in and data out have the same polarity and the read operation is non-destructive. The TRI-STATE® outputs simplify busing by keeping impedance high when the card is not being addressed, and allow easy expansion of bus interfacing.

Mr. Miller explained that all of National's memory systems are subject to the same stringent quality and test standards, and are designed to guarantee fast, adjustment-free maintenance and high reliability. All components are burned in, and the systems undergo extensive diagnostic testing before shipment.

According to Mr. Miller, the NS 400-N series complements National's expanded family of static memory systems, which provide a wide choice of memory speed, size and power consumption, including the ultra low power NS 400-C, CMOS series, which is available with 4,096 words of 8, 9, or 10 bits each.

NATIONAL WINS MICROCOMPUTER CONTRACTS IN EXCESS OF $267,000

SANTA CLARA, CA—The Microcomputer Systems Group of National Semiconductor Corporation, here, announced the receipt ot three IMP-16L microcomputer contracts, totalling $267,500, from Sono-Mag Corporation, Bloomington, IL, from Princeton Applied Research, Princeton, NJ, and from 3M Company, Minneapolis, MN.

The Princeton Applied Research contract, for $100,000, includes IMP-16L/300 cpu application cards and IMP-16L/006P random-access memory/programmable-read-only memory (RAM/PROM) cards for use in their medical electronics equipment. The contract runs for ten months with renewal options; deliveries begin in April.

In unrelated events, Sono-Mag Corporation purchased, for $72,500, IMP-16L/300 cpu application cards and IMP-16L/006B RAM cards for use in remote radio broadcast stations controls. The $95,000 3M contract is for microcomputer cards used in the firm's facsimile equipment.

National's IMP-16L/300 cpu card is a general-purpose 16-bit microcomputer designed expressly for the OEM user. Using a bit-slice architecture, the cpu application card has a control read-only memory chip which provides an extended instruction set for convenient programming. A high-speed asynchronous bus and control logic affords direct memory access.

The IMP-16L/006P memory cards have 2048 by 16 bits of static RAM and 4096 by 16 bits of ultraviolet-eraseable PROM. All cards include timing and control logic for direct interfacr to the IMP-16L cpu application card.

National Semiconductor Corporation, 2900 Semiconductor Drive, Santa Clara, CA 95051.

For more information:

Mktg:	Bob Pecotich	(408) 737-6115
PR:	Roy Twitty	(408) 737-5287
or	Jim Rose	(213) 883-9884

SCREEN—IMAGE HARD COPY FOR THE VDM—1

Dear Dr. Dobbs, March 18, 1977

I have found the short screen image copy routine to be very handy and perhaps the method of calculating the top screen line's position in memory might be useful to your readers.

Thanks,

Dan S. Parker
1007 3rd St. No.3
Davis, CA 95616

P.S. I'm presently working with John Moorhead, also of Davis, CA, to submit a software scroll VDM-1 driver based on LiChen Wang's driver that does everything PT's VDM-1 driver does (except turn cursor off completely) in about half the number of program bytes that PT's takes: approx 240 compared to PT's 450 byte program. We have it working, just need to complete the documentation. Hope you'll be interested when we submit it. **[yup]** Our driver should also work directly with Solid State Music's VB-1 board. We will check to make sure when I finish building a VB-1.

SCREEN-IMAGE HARD COPY FOR THE VDM-1

Dan Parker
March 18, 1977

A convenient feature of using both a video display device such as Processor Technology's VDM-1 and a hard copy device in an operating system is the ability to edit text, list, or run a program using the high speed and versatility of the video device and then produce a hard copy of the final product. Unfortunately, producing a hard copy usually means rerunning the program, or reassembling it with a listing, this time with a sense switch in another position to get the hard copy.

The following program called SCRPY, which is modified slightly to "stand alone", serves as a nice subroutine addition to an 8080 monitor to produce a hard copy "image" of whatever is currently displayed on the screen. A nonprinting control character, which doesn't modify the contents on the screen, produces a call to the subroutine.

A few notes about SCRPY: The software assumes the VDM-1 is being driven by Processor Technology's VDM-1 driver with the screen memory addressed at C000-CFFF. PT's VDM-1 software takes advantage of the hardware scroll capa-

bility of the VDM-1. As a result, the topmost line of text as seen on the TV screen is not necessarily stored in the lowest portion of the screen display memory. (Of course, if a software-scroll type of driver is used, such as Lichen Wang's driver, *DDJ*, Volume 1, No. 6, then the task is much simplified to merely scanning the screen memory and outputting each character.)

The program listed below uses the variable BOTL (Bottom of Text Line) generated by PT's VDM-1 driver (located at 09B5HEX in PT's VDM-1 driver source listing) to calculate where the top line on the TV screen is stored in memory. The program then continues to output the screen memory one character at a time, with the B and C registers keeping track of the 64 characters by 16 lines, respectively.

```
0000                    0010 *
0000                    0015 *     SCREEN IMAGE HARD COPY FOR THE VDM-1
0000                    0020 *
0000                    0025 *     BY DAN PARKER, 1007 3rd ST. #3
0000                    0030 *        DAVIS, CALIFORNIA  95616
0000                    0035 *
0000                    0040 *              " SCRPY "
0000                    0045 *
0000 31 FF 0F           0050 SCRPY    LXI   SP,0FFFH    ;SET STACK POINTER
0003 21 00 CC           0055          LXI   H,CC00H     ;SCREEN MEMORY ADDRESS
0006 3A B5 09           0060          LDA   BOTL        ;GET BOTL FROM VDM-1 DRIVER
0009 01 40 00           0065          LXI   B,0040H     ;PUT CHARS./LINE IN B&C
000C 09                 0070 CONT1    DAD   B           ;ADD TO H&L
000D 3D                 0075          DCR   A           ;CONTINUE UNTIL START OF
000E                    0080 *                          ;TEXT LINE IN MEMORY
000E CA 14 00           0085          JZ    CONT        ;START OF TEXT IN H&L?
0011 C3 0C 00           0090          JMP   CONT1       ;
0014 01 11 40           0095 CONT     LXI   B,4011H     ;CHARS & LINES+1 IN B&C
0017 CD 36 00           0100 NXTLN    CALL  CRLF        ;GET CARRIAGE RET & LF
001A 7C                 0105          MOV   A,H         ;
001B FE D0              0110          CPI   0D0H        ;STILL WITHIN SCREEN MEM?
001D CC 32 00           0115          CZ    RESET       ;RESET H&L IF NOT
0020 0D                 0120          DCR   C           ;DECREMENT LINE COUNT
0021 CA 00 C0           0125          JZ    MONITOR     ;RETURN TO USER'S MONITOR IF DONE
0024 06 40              0130          MVI   B,040H      ;RESET CHARACTER COUNT
0026 7E                 0135 CONT2    MOV   A,M         ;GET CHARACTER
0027 CD 41 00           0140          CALL  WRITE       ;AND OUTPUT IT
002A 23                 0145          INX   H           ;INCREMENT MEMORY POINTER
002B 05                 0150          DCR   B           ;DECREMENT CHARACTER COUNT
002C CA 17 00           0155          JZ    NXTLN       ;START ON NEW LINE IF DONE
002F C3 26 00           0160          JMP   CONT2       ;OTHERWISE CONTINUE OUTPUT
0032 21 00 CC           0165 RESET    LXI   H,CC00H     ;RESTORE H&L TO START OF SCREEN
0035 C9                 0170          RET               ;MEMORY
0036                    0175 *
0036                    0180 *   USER'S I/O UTILITIES
0036                    0185 *
0036 3E 0D              0190 CRLF     MVI   A,0DH       ;"CR"
0038 CD 41 00           0195          CALL  WRITE       ;OUTPUT IT
003B 3E 0A              0200          MVI   A,0AH       ;"LF"
003D CD 41 00           0205          CALL  WRITE       ;
0040 C9                 0210          RET
0041                    0215 *
0041 F5                 0220 WRITE    PUSH  PSW         ;SAVE CHARACTER
0042 DB 00              0225 STAT     IN    00H         ;STATUS PORT
0044 E6 02              0230          ANI   02H         ;CHECK TBE
0046 CA 42 00           0235          JZ    STAT        ;
0049 F1                 0240          POP   PSW         ;RESTORE CHARACTER
004A D3 01              0245          OUT   01H         ;OUTPUT IT
004C C9                 0250          RET               ;
004D                    0255 *
004D                    0260 BOTL     EQU   09B5H       ;BEGINNING OF TEXT LINE (FROM
004D                    0265 *                          ;PT'S VDM-1 DRIVER GLOBAL RAM)
```

HERE, NOW: A MICRO APL FOR 8080's

Dear Mr. Warren: *May 18, 1977*

I have just finished a micro APL for the 8080. It is completely debugged and operational. I am enclosing the documentation for publication which covers all of the operators and gives examples that thoroughly explain how to use it.

Since I hand assembled the language, I don't have a source listing available for publication (Manual typing of a 5.25K program is impossible!)

I am selling Tarbell Cassette tapes of EMPL 1.0 for $10 each. The tape comes with a user's manual, including complete information on implementing EMPL on any 8080 system, details of EMPL's operation (such as workspace organization), tape loading software, etc. Any improvements I make in the meantime will be included and completely documented. I hope this will be of interest to your readers.

Sincerely, *Britton House*
Erik T. Mueller *Roosevelt, NJ 08555*

E M P L , an 8080 APL

Britton House
Roosevelt, NJ 08555
© Copyright 1977 by Erik Mueller

8K EMPL is a micro APL for the Intel 8080. The current version fits in the first 5,376 bytes of memory. All the special symbols and operators of APL have been adapted to the ASCII character set.

The following description explains EMPL. Examples are used to clarify all operations.

EMPL has two modes: The normal Execution Mode in which all instructions are executed immediately, and the Definition Mode which permits the user to enter functions.

When in Execution Mode, the computer indents five spaces and waits for input.

Type an arithmetic expression and EMPL will respond with the result.

```
    2-6-3
-1
```

(Computer responses are underlined.)
All expressions are evaluated by the right-to-left rule. That is, each operator takes as its right argument everything to its right. For example, in the above problem, EMPL first evaluates 6-3 which equals 3, then it evaluates 2-3 which equals -1.

The range is ±32767-- double-byte integer arithmetic is used.

Variable names may be any length of alphabetic characters. They are assigned values with the ":=" (typed as Control I.)

```
  SPEED:=2059
  SPEED - 59
2000
```

One dimensional arrays are called vectors in EMPL. A variable may be specified to be a vector of any length provided sufficient memory is available.

```
    A:= 5 -4 2 33
    A
5  -4  2  33
     A+1
6  -3  3  34
     A*4 3 2 0
20 -12 4 0
```

A Dyadic Operator (such as "+","*","?", etc.) takes two arguments, one on either side of the operator; as compared with a Monadic Operator such as "!" (absolute value), or "?" (random), which takes only one argument to its right.

Constants may be numeric or string vectors. Since the internal representation of both is the same, operations may be performed between the two types. To print a vector as a string, type "$" before it.

```
    X:='STRING'
    X
83  84  82  73  78  71
    $X
STRING
    $X=1
TUSJOH
```

A description of some of EMPL's unique operators follows: "\" is used to create a vector of consecutive integers between 1 and the right argument.

```
    \5
1   2   3   4   5
```

"^" gives the length of the right argument.

```
    ^'HELLO'
5
    ^\23
23
```

"'" returns the smaller of two numbers; """ returns the greater.

```
    -1'2
-1
    4  1  9"3  2  56
4  2  56
```

"." gives the remainder when the right argument is divided by the left.

```
    2.5
1
```

"<","<="(Control Y), ">",">="(Control N), "=", and "#" return a 1 if the condition

```
is true, otherwise they return a 0.
    5<4
0
    7 3 4# 3
1  0 1
```
"," chains together two vectors.
```
     HI:= 1 2 3 4
     WN:=419 512
     XX:=HI,WN
     XX
1 2  3 4 419 512
     A:='CATE'
     B:='NATION'
     $A,B
CATENATION
     18,2,10*4
18 2 40
     18 2 10*4
72 8 40
```
"[" selects elements of the left argument at the locations given by the right argument.
```
     A:=2 17 -3 4
     A[1 1 2
2  2  17
     $'ABCRST'[3 1 6
CAT
```
"f%" performs the Dyadic Scalar Operator f from right to left on all the elements of the right argument. Thus, it treats +%\5 as if it were 1+2+3+4+5.
To find the largest element in a vector, type "%VECTOR.

To write a program (or function), the user must enter EMPL's Definition Mode. Type "&" followed by the new name of the function.
```
     &PAS
>
```
The computer now prompts with a ">" indicating that it is in Definition Mode. Editing is now like a BASIC editor: Type a line to insert it in the function; type the line number alone to delete the line; type an old line over to replace the old one. Type "@" to list the function; type "$" to renumber by a specified amount.
For example, the user types in a program to print the first N rows of Pascal's triangle.
```
>10 P:=1
>20 =:PRINT*N>=^P:=(0,P)+P,0
>15 PRINT: P
>$ 10
>@
     &PAS
[10] P:=1
[20] PRINT: P
[30] =: PRINT*N>=^P:=(0,P)+P,0
>
```
Note that ":=" can be used inside any expression. "PRINT:" is used as a label in line 20. Upon exiting Definition Mode, PRINT becomes a variable whose value is 20.
"=:" (Control H) is a branch to the line given by the expression which follows it. Branch to a null vector is no branch (computer proceeds to next line), and branch to an undefined line is the end of program. The user now types a "&" to leave Definition Mode.
To execute a program, type its name.
```
>
     N:=4
     PAS
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```
Conditional branches are accomplished by the following:
```
     =:LINE*\TEST
```
If the condition TEST is true, the computer will branch to LINE. Otherwise, the computer will proceed to the next line.
Strings inside double quotes may be used to print comments.
```
     "N=";N
N=4
```
Note that semicolons are used to separate items, and a semicolon at the end will suppress CRLF.
"%:" is used to execute a string as if it had been typed in execution mode.
```
     STR:='A:=5*\5'
     %:STR
     A
5  10  15  20  25
```
Specific elements of a vector may be assigned values.
```
     A:=\6
     A[3 4]:= 17 18
     A
1  2  17  18  5  6
```
When "@" is assigned a value, the value is printed on the terminal.
```
     A:=5*@:=5
5
```

When "$" is assigned a value, the value is printed as a string on the terminal.

```
     A:=$:='TESTING'
TESTING
```

If "@" is used as an operand in an expression, the computer first requests input from the terminal. "$" enables string input to be used. (The input is taken as a literal.)

```
     &TEST
[10] "WHAT IS YOUR NAME?";
[20] NAME:=$
[30] "HI,";$NAME
[40] "INPUT AN EXPRESSION"
[50] EXPR:=@
[60] "THE ANSWER IS";EXPR
     &
     TEST
WHAT IS YOUR NAME?RACHEL
HI, RACHEL
INPUT AN EXPRESSION
@:5*5
THE ANSWER IS 25
```

User defined monadic or dyadic functions are possible. Here is an example of a user defined monadic function to take the factorial of the right argument.

```
     &F:=FACT VL
 10  F:=*%\VL
     &
```

"FACT" may now be used as any EMPL monadic operator can be used.

```
     FACT 3
6
```

When FACT 3 is typed, a function called FACT is searched for. When it is found, VL is set equal to 3 and the function is executed. Upon completion of the execution, the computer uses the variable specified to the left of the assignment operator ":=" as the result.

Here is a user defined dyadic function.

```
     &RS:=SZ RHO VTR
[10] RS:=\0
[20] =:20*\(⍴RS:=RS,VTR)⍳SZ
[30] RS:=RS[\SZ
     &
```

This function is used to restructure the vector on the right such that it has the length indicated on the left.

```
     4 RHO 5 2 3
5  2  3  5
     $3 RHO 'ABCDEFG'
ABC
```

To reenter Definition mode of an old function, just type its name.

```
     &RHO
⍳
```

Parentheses are used to override the normal order of evaluation.

```
     (3-5)+7
5
```

The following commands may be used inside a program, or in Execution Mode.

```
)CLEAR - Clear the workspace.
)FNS   - List the names of all functions
         in the workspace.
)VARS  - List the names of all variables
         in the workspace.
)STOP  - Return to execution mode.
)ERASE object - Erase the specified varia-
         ble or function.
)SI    - Display the State Indicator.
)PUR   - Clear the State Indicator.
```

The State Indicator is the stack of the return addresses of all programs whose execution is pending.

Suppose a program is written, PROG, which calls for the execution of another one, ASK:

```
     &PROG
[10] ASK
[20] "OK"
      .
      .
      .
     &ASK
[10] HI THERE
     &
     PROG

ASK[10]HI THERE
ERROR  2810
```

The computer prints the program (or function) and line in which the error occurred.

```
     )SI
ASK[-1] *
PROG [20]⍝
2210 FREE
```

An error has occurred in line 10 of ASK. Since the execution of ASK hasn't been completed, the computer still has the return address saved. (Line 20 of PROG) Now whenever a =:0 instruction be typed, execution will continue at line 20 of PROG.

The program with the "*" to the right of it is the current suspended program. All

"=:" (branch) statements refer to this program. Line -1 indicates that it is currently in execution mode. All programs with "^" to the right of them are pending execution. The program to return to next is always on the top of the list. It is a good idea to clear the State indicator often to prevent excess garbage from taking up memory. The State Indicator is automatically cleared when entering Definition Mode, and after ERROR 5097 (workspace full.)

Here are some sample functions to illustrate EMPL's use.

(1) Generate n prime numbers:

```
     &P:=PR END
[10] P:=1+T:=1
[20] TEST: =:0*\END<=^P
[30] ADD: =:ADD*\+%0=P.T:=T+2
[40] P:=P,T
[50] =:TEST
     &
     PR 10
2  3  5  7  11  13  17  19  23  29
     1 + PR 3
3  4  6
```

(2) Compress a vector:

```
     &ANS:=SLCT COMP VCTR
[10] IDX:=1, ANS:=\0
[20] BK: ANS:=ANS,(VCTR[IDX)*\SLCT[IDX
[30] =:BK*(IDX:=IDX+1)<=^VCTR
     &
     1 0 1 1 0 COMP\5
1  3  4
```

(3) Sort a group of numbers:
(This function uses COMP)

```
     &ORD:=SORT UNS
[10] ORD:=\0
[20] LB: =:0*\&^UNS
[30] WHICH:=UNS='%UNS
[40] ORD:=ORD,WHICH COMP UNS
[50] UNS:=(&WHICH) COMP UNS
[60] =:LB
     &
     SORT -2 1 -1 0 2
-2  -1  0  1  2
```

(4) Reverse a vector:

```
     &RV:=REV VC
[10] VC[(1+^VC)-\^VC
     &
     $REV'HI THERE'
 EREHT IH
```

(5) Raise the left argument to the right argument. (They must both be single numbers.)

```
     &E:=BSE RAIS NUM
[10] EX:=*%BSE*&&\NUM
     &
     2 RAIS 5
32
```

E M P L Language Summary

Monadic Scalar Operators

-Y	Minus Y
!Y	The absolute value of Y
&Y	Not Y
?Y	Random from 1 to Y

Monadic Mixed Operators

\Y	Vector of consecutive integers from 1 to Y
^Y	The length of Y

Dyadic Scalar Operators

X+Y	X plus Y; X or Y
X-Y	X minus Y
X*Y	X times Y;X and Y
X/Y	X divided by Y
X'Y	Minimum of X and Y
X"Y	Maximum of X and Y
X.Y	Remainder of Y/X
X<Y	X less than Y
X<=Y	X less than or equal to Y
X>Y	X greater than Y
X>=Y	X greater than or equal to Y
X=Y	X equal to Y
X#Y	X not equal to Y

Dyadic Mixed Operators

X,Y	Y catenated to X
X[Y	Elements of X at locations Y

Composite Operators

f%Y	The Operation f performed from right to left on all the elements of Y

Special Operators and characters

X:=Y	Assign Y to X
$X	Print X as a string
"TEXT"	Print literal text
%:X	Execute a string
=:X	Branch to X
VAR X	Indexed variable assignment
@	Numeric Input/Output
$	String Input/Output

```
'TEXT' String vector
(X)    Expression in parentheses

Function Definitions
&⟨function⟩    Niladic function; no result
&⟨result⟩:=⟨function⟩⟨rightvar⟩  Monadic
            function with result
&⟨result⟩:=⟨leftvar⟩⟨function⟩⟨rightvar⟩
            Dyadic function with result


Definition Mode Commands
@     List function
$     Renumber
&     Return to execution mode

EMPL Range = ±32,767

Commands
)CLEAR      Clear Workspace
)FNS        Display function names
)VARS       Display variable names
)PUR        Clear State Indicator
)ERASE o b  Erase function of variable
)SCN        Clear screen
)SI         Display State indicator
)STOP       Return to execution Mode

Special Control characters
Control B   Backspace
Control C   Stop
Control F   Forward space
Control H   =:
Control I   :=
Control N   ⟩=
Control Y   ⟨=

Error messages
412  String is too large to execute
885  Object already erased
1082 Unknown operand type in expression
1469 Length of vectors doesn't match
2268 Illegal Operation in reduction
2495 Variable not present in indexing
2529 Length of index doesn't match length
     of value to be assigned
2559 Index greater than length of variable
2628 Unknown assignment operand type
2690 Function not found
2810 Variable or function not found
3503 Backspace less than beginning of line
     buffer
3521 Forward spacing greater than end of
     line buffer
3542 Input over 72 characters
3737 Bad syntax in function definition
3827 Illegal command in definition mode
4058 Illegal redefinition of monadic func-
     tion header
4083 Illegal redefinition of dyadic func-
     tion header
5097 Workspace full
5302 Division by zero
```

MANUAL AVAILABLE FROM THE DISE REPOSITORY

A COMPUTER SCIENCE HARDWARE LABORATORY MANUAL

Professors B. Carey and M. Barber

Electrical Engineering and Computer Science
University of Connecticut
Storrs, Connecticut 06268
(203) 429-4816

This manual presents a sequence of nine experiments involving the design, construction, and testing of digital hardware to be completed in a fourteen week semester. Each experiment requires four hours of lab work per week. The laboratory is physically organized around a number of general purpose, digital, bread-boarding modules to permit a convenient structuring of experiments and the implementation of problem solutions. The experiments consist of a logical sequence of problems which start with the design of combinational modules, progress to standard sequential functions such as shirt registers and binary counters, move on to multifunction modules, explore the transmission of information between and synchronization of two digital networks, look at the design of an arithmetic function in a digital computer, and ends with the design and programming of a complete general purpose computer system. The experiments involve the reasonable range of SSI, MSI and LSI devices seen in the 7400 logic family and may directly concern themselves with other logic families if appropriate.

HOW TO OBTAIN: A single copy will be provided without charge to any individual involved in teaching or in the development of digital systems education. Requests should be made on the letterhead stationery of the individual's organization and include the individual's title. Please send requests to: DISE REPOSITORY, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

MUG OFFERS MEDICINE FOR MUMPS

1976 MUMPS Users' Group Meeting is a Huge Success

The fifth MUMPS Users' Group Meeting was held 9/29-10/1 in Madison, WI. [MUMPS stands for Massachusetts (General Hospital) Utility Multi-Programming Systems. It is a FOCAL-like language with extensive file-handling facilities for tree-structured files as a major feature. Notable aspects from the hobbyist viewpoint are (1) it's in the public domain, and (2) it runs on a PDP-11. It was originally developed for use in hospital and biomedical applications.—Editor] The meeting was packed with exciting presentations and live demonstrations of interactive applications developed using the MUMPS computer language. Among the topics of special interest were voice-computer interfaces, medical information systems, radiology systems, patient counseling by computer, patient appointment systems, computer-aided instruction, on-line cataloging for the National Library of Medicine, and the formation of the MUMPS Users' Group Application Library to facilitate the transfer of MUMPS applications.

An introductory tutorial on programming in MUMPS was given, and was attended by administrators and physicians as well as programmers. A tutorial and a workship on advanced MUMPS techniques were held for experienced MUMPS programmers. There were other workshops on such topics as computer-aided instruction and MUMPS application transfer.

Seven vendors of MUMPS applications and implementations participated in a vendors' forum. Two of the vendors that exhibited at the Meeting (namely, Artronix, Inc. of St. Louis, and Digital Equipment Corporation of Maynard) announced their completion of implementations of the NBS-defined Standard MUMPS which is now seeking approval as an ANSI Standard.

There are now about 400 institutions around the world that use MUMPS, and this total is growing by about 80% per year. A variety of computers are used, including DEC's PDP-11, PDP-15 and PDP-10, Data General's Eclipse and Nova, Artronix' PC-12 and PC-16, Burroughs' B6700, and IBM's System/360 or System/370. For information on MUMPS, future MUG meetings, and MUMPS applications, contact the MUG Executive Secretary, Dr. Joan Zimmerman, Biomedical Computer Laboratory, 700 S. Euclid Ave., St. Louis, MO 63110, (314) 454-3364.

GUIDE TO 8080 PILOT, VERSION 1.1

Copyright © 1977 by John A. Starkweather, Ph.D.
February, 1977

INTRODUCTION

PILOT is a programming system for controlling interactive conversation. PILOT stands for Programmed Inquiry, Learning or Teaching. It has most commonly been used as an author language for Computer-Assisted Instruction. It was first developed at the University of California in San Francisco and has been implemented on a variety of large and small computers. This guide provides a description and operating procedures for a version of PILOT that will run on computers using the Intel 8080 microprocessor. It is an interpreter written in assembly language that requires less than 4k bytes of memory for the interpreter code. Total memory requirements depend on the space required for PILOT program text and will often be no more than 8k bytes.

PILOT is designed to be simple in its syntax so that those without prior computer experience can easily learn to control its features. Dialogue programs can be rapidly constructed and tested.

The program mentioned in this article was developed via a contract from the National Library of Medicine, Lister Hill Center, Bethesda, Maryland.

The source code for 8080 PILOT, Version 1.1 will be published in the May issue of *DDJ*. –Ed.

Requests for PILOT-8080 in machine-readable form should be directed to:

PILOT-8080
Computer Technology Branch
Lister Hill Center for Biomedical Communication
National Library of Medicine
8600 Rockville Pike
Bethesda MD 20014
(301) 496-1647

Now that you know about PILOT, it is a reasonable idea that you may want some programs in PILOT, who else is using it, etcetera. In fact, the sort of services provided by the

PILOT INFORMATION EXCHANGE

which maintains a library of materials in PILOT, source code in several languages, and a membership list with a brief note as to who is doing what . . .

So, get in touch with:

Gregory Yob
PO Box 354
Palo Alto CA 94301

Paper tapes of the 8080 PILOT will be available shortly . . ., the details are being worked out now.

If you have Tarbell or Cutts, sufficiently loud and repeated requests will motivate availability in this medium as well.

```
WHY NOT BASIC?

   It is possible to imitate the operation of PILOT in a
general purpose programming language such as BASIC,
APL, FORTRAN, or PL/I. The reverse is certainly not
true, for PILOT is a specialized language oriented
toward dialogs, drills, tests, etc.  If a BASIC program
is to be made interactive for the handling of free-
response dialog, the programmer must expend a large
effort in processing input and arranging for
comparisons with possible words or portions of words
that might be important to recognize.

   PILOT makes this kind of processing easy and keeps the
program sufficiently readable that it can be reviewed
by someone with a primary interest in the logic of how
language content is to be presented.  PILOT is
relatively poor at kinds of computation handled well by
general purpose languages.  For many instructional
uses, that doesn't matter, but some versions of PILOT
allow a mix of programming with a concurrently
available general language.

TRY IT YOURSELF

This manual attempts to give complete written
information about PILOT programming and operation of
PILOT-8080, Version 1.1.  The user is encouraged,
however, to explore actual operation of PILOT
instructions by using the test program listed in
Appendix C and distributed with the system.  Most PILOT
functions can be exercised in a variety of ways with
this program.

                PROGRAMMING PILOT

A SAMPLE PILOT PROGRAM

A PILOT program consists of a series of statements
which give a step-by-step description of what the
computer must do in order to interact with a person
sitting at a keyboard.  An instruction code, consisting
of one or more letters followed by a colon, defines the
statement type and determines its function.

Single letter codes define "core" instructions that
occur in all versions of PILOT.  Of these, three are
most important, and can be used to demonstrate the
writing of a simple PILOT program:

        T: DO YOU ALREADY KNOW ABOUT PILOT?
        A:
        M: YES, SURE, YEP, RIGHT, TRUE
        TY: THEN YOU CAN SKIP THIS INTRODUCTION.
        TN: THIS INTRODUCTION IS FOR YOU.

The three codes are:
        T: for "type text"
        A: for "accept an answer"
        M: for "match"
```

The letters Y and N are "conditioners" that when appended to another code cause it to be effective or not depending upon the success of the last attempted match. Thus, the text "THEN YOU CAN SKIP THIS INTRODUCTION." will be typed only if the answer given to the question about PILOT contains the words "YES" or "SURE". The text "THIS INTRODUCTION IS FOR YOU." will be typed only if the match was not successful. A sequence of operation might be as follows:

```
      DO YOU ALREADY KNOW ABOUT PILOT?
  I SURE DO.
      THEN YOU CAN SKIP THIS INTRODUCTION.
```

ANALYSIS OF RESPONSES

PILOT is designed to allow the development of conversational programs that allow relatively free response by the user. Recognition of user responses is accomplished, as indicated in the example above, by searching the responses for specific words or word stems. In ordinary conversation, one or more words in a sentence often carry most of the sentence's meaning. The M: statement is used to define those words, portions of words, or word groups that we wish to be recognized in an answer. Program authors can allow conversational replies to their presented questions and develop a list of word elements which are keys to appropriate recognition and handling of the replies.

PILOT STATEMENTS

There are five more types of "core" statements, using the letters J, U, E, C, and R. J stands for "jump" and causes the PILOT program to continue from another part of the sequence of statements marked by a label in the J-statement. U stands for "use" and causes a marked group of statements to be "used" at that point in the program. The group of statements (called a subroutine) begins with a label which is named in the U-statement and ends with an E-statement. The E-statement, meaning "end", indicates the end of a subroutine or the end of the entire program.

C stands for "compute" and a C-statement contains notation calling for computation of numeric constants or variables. The portion of the C-statement to the right of the colon contains a complete statement in the syntax of a general programming language such as BASIC. The present system is limited to assignment statements.

R stands for "remark". An R-statement contains text which will not affect program operation but which can be helpful as information to the program author.

CONDITIONAL EXECUTION

The letters Y and N are conditioners that affect the operation of any statement type when appended to the code. Operation will then depend on the last attempt to match M-statement patterns with the latest A-statement response. Y: by itself is allowed as a shorthand form of TY: In like manner, N: by itself is allowed as a shorthand form of TN:

A PILOT statement can also be made conditional upon the value of an arithmetic or logical expression placed in parentheses after the instruction and before the colon. In the present system this method is limited to a numeric variable name. If the current value of the referenced numeric variable is > 0, then the statement will be executed. Otherwise it will be ignored. C-statements can be used to set the value of the numeric variable. In the present system, this method of parenthetic condition will take precedence over the Y-N condition.

CONTINUATION

A colon (:) at the beginning of a line indicates continued use of a text statement (T, Y, or N). Thus:

```
T:THIS TEXT WILL BE DISPLAYED
 :AND THIS WILL ALSO BE DISPLAYED.
```

PILOT LABELS

Any PILOT statement may include a label prior to the operation code which allows the statement to be referenced by JUMP (J:) or USE (U:) statements. Labels begin with an asterisk (*) and end with a blank. They may have a maximum of ten characters. A label may also be used as the only element in a line, terminating with the end of line carriage return.

In this system, no complaint is made if there are duplicate labels, but only the first in sequence will be found when the label is referenced.

EXAMPLE:

```
*LABEL
T:DISPLAY THIS
.....
.....
J:*LABEL          (causes jump to the above)
```

STRING VARIABLES

Pilot allows the response to an ACCEPT (A:) operation to be saved as a character string and retrieved later in the program. PILOT will save the response if a string variable name is written after the colon in an ACCEPT (A:) statement. A string variabale name begins with a blank or with the end of line carriage return.

Later on, if the string variable name appears in any portion of a TYPE (T:), YES (Y:), or NO (N:) statement,

then it will be replaced with the latest response given to the A-statement which contained it.

If there has been no prior response saved for a string variable name in a T, Y, or N-statement, then the string variable name itself will be displayed.

A string variable name may have a maximum of ten characters in addition to the dollar sign. The character string stored as a string variable is limited by the length of response to ACCEPT (A:) allowed at that point in the program.

EXAMPLE:

```
-----------------------------------------------------
        (program)                    (execution)
    A:$NAME                        "ROBERT" entered
    T:HELLO, $NAME                  HELLO, ROBERT
    .....
    .....
    T:THIS IS $UNKNOWN              THIS IS $UNKNOWN
-----------------------------------------------------
```

NUMERIC VARIABLES

In the present system, numeric variables are limited to integer values from -99 to +99. Numeric variable names begin with the poind sign character (#) and consist of a single alpha letter (A-Z). Numeric variables may be used in an ACCEPT (A:) statement to allow a numeric value to be entered and in a TYPE (T:), YES (Y:), or NO (N:) statement for retrieval and display. They may be altered either by an A-statement entering a new value or by a C-statement causing assignment of a new value. They are set to zero at the time that PILOT is initialized.

When a numeric variable name appears in a conditional statement (such as T(X):HELLO) or in a C-statement (such as C:X=A+B), the # character is not used.

EXAMPLE:

```
-----------------------------------------------------
        (program)                    (execution)
    A:#X                           "23" entered
    T:THE VALUE IS #X               THE VALUE IS 23
-----------------------------------------------------
```

IMMEDIATE OPERATION

Preceding a PILOT statement with ctl-Z causes immediate execution of the statement when it is terminated with a carriage return. Thus, whenever PILOT is awaiting response to an ACCEPT (A:) statement, an immediate statement may be used. After the operation is complete, PILOT will return to expect a normal response to the same A-statement. For example, "(ctl-Z) DP:" will display the current program in memory and "(ctl-Z) U:*LABEL" will cause execution of the referenced subroutine.

Ctl-Z followed immediately by a carriage return will abort the current program operation and return PILOT to its starting point.

PILOT STATEMENT DESCRIPTIONS

SUMMARY

The following section describes each of the PILOT instructions provided in this version of PILOT for the 8080 microprocessor (PILOT-8080, Version 1.1). The descriptions include syntax, function, and examples.

Where appropriate, a description is given of any error messages that might be generated by invalid statements or invalid user response to their operation.

Appendix A presents a syntax description in formal notation.

Instructions are described in the following order:

PILOT core instructions

T:	TYPE (includes Y: and N:)
A:	ACCEPT
M:	MATCH (includes description of MC:, a keyword)
J:	JUMP
U:	USE
E:	END
C:	COMPUTE
R:	REMARK

Note that the core instructions are limited to single letter codes. These instructions are standard in syntax and operation for many different implementations of PILOT. It is therefore advantageous to write as much program material as possible with the use of core instructions if you wish others to benefit from your work. When extensions to the language are made to meet special needs, multi-letter codes are used and termed keywords to distinguish the difference.

PILOT-8080 keyword extensions

INMAX:	INPUT LINE LENGTH MAXIMUM
NEW$:	ERASE STRING VARIABLES
DP:	DISPLAY CURRENT PROGRAM
LOAD:	LOAD NEW PROGRAM
SAVE:	SAVE CURRENT PROGRAM
PRINT:	PRINT CURRENT PROGRAM
EDIT:	EDIT CURRENT PROGRAM
BYE:	EXIT FROM PILOT

TYPE (T:), YES (Y:), and NO (N:)

SYNTAX:

```
[<label>] T [<cond>] : <message>
[<label>] Y [(<numeric variable>)] : <message>
[<label>] N [(<numeric variable>)] : <message>
```

DESCRIPTION: Displays a message to the PILOT user. A message can consist of a literal character string, string variables, and numeric variables. All character positions to the right of the colon are reproduced in a literal fashion with the exception that the values of variables are inserted as replacements for their names. String variable names are identified by beginning with "$" and numeric variable names by beginning with "#".

YES (Y:) and NO (N:) statements are abbreviated forms of TY: and TN: and are entirely equivalent in operation. Since these statements are already conditional upon the YES/NO condition, the only valid further condition in this version of PILOT is by the value of a numeric variable.

ERROR MESSAGES: Reference to a variable that has not been assigned a value will cause display of the string variable name. This may of course be done deliberately, but it will often indicate misspelling of the string variable name.

EXAMPLE:

```
        T:THE NEXT LINE WILL DISPLAY "HELLO, $NAME."
        T:HELLO, $NAME.
        A:$NAME
        T:DISPLAY WILL BE: "HELLO, (entered name)."
        T:HELLO, $NAME.
        C:S=S+1
        T:YOUR NEW SCORE IS #S.
```

ACCEPT (A:)

SYNTAX:

```
[<label>] A [<cond>] :
[<label>] A [<cond>] : $<string variable>
[<label>] A [<cond>] : #<numeric variable>
```

DESCRIPTION: Opens the console device for the PILOT user to enter a response. The user may cancel the last character with delete (shift O) or rubout and cancel the current line with ctl-U. The line terminates when carriage return is entered of if the INMAX: value is reached.

If there is no operand then the response will be treated as a character string and stored in a temporary entry buffer.

If a string variable name is included then the response will be stored in the string variable.

If a numeric variable name is included and a non-numeric response is entered, an error message will be displayed and another response will be accepted.

LIMITATIONS: The length of a response is limited to 80 characters or to a lower limit set by the keyword CHMAX:n. CHMAX has an initial value of 72 in this version of PILOT. A numeric variable response must be a one or two positive digit integer between 0 and 99.

ERROR MESSAGES: "*NUMERIC ENTRY REQUIRED" occurs if a non-numeric entry is attempted in response to an accept with numeric variable name.

"*NO ROOM" indicates insufficient space for string variable storage.

EXAMPLE:

```
        T:WHO ARE YOU?
        A:
        .....
        .....
        T:WHAT IS YOUR NAME?
        A:$NAME
        .....
        .....
        T:WHAT IS YOUR AGE?
        A:#A
        .....
        .....
```

MATCH (M:) and (MC:)

SYNTAX:

```
[<label>] M  [<cond>] : <pattern>[,<pattern>...,<pattern>]
[<label>] MC [<cond>] : <pattern>[^<pattern>...^<pattern>]
```

DESCRIPTION: Causes a scan of the last user response, the one received by the last ACCEPT (A:) statement. Patterns in the expression field of the MATCH statement are compared with the last response. Any item which matches causes the YN-switch to be set true. Patterns in the list are separated by commas, with leading or trailing blanks considered part of the pattern. A comma which terminates the last item is ignored but can serve to indicate the presence of a trailing blank.

MC: is a keyword extension which allows a search for text containing commas. It is identical except for the substitution of the caret (^) in place of the comma as a separator between patterns. On a teletype this character (5EH) is produced by shift N and prints as an up arrow.

OPERATION:

1. M-patterns have multiple blanks reduced to one.
2. The last response has a blank added to each end.
3. The last response has multiple blanks reduced to one.
4. A moving window scan of the response with each pattern occurs until either a match is found or the input is exhausted.

EXAMPLES:

```
M:A,B,C
        Matches A or B or HAT or ALICE or JOB
        Does not match TENT or X

M: A, B, C
        Matches A or B or ALICE
        Does not match JOB or HAT

M: A , B , C ,
        Matches only A or B or C
```

JUMP (J:)

SYNTAX:

```
[<label>] J [<cond>] : [*] <destination>
```

DESCRIPTION: Jumps to the specified destination in the current program. The destination must reference an existing label, but the leading asterisk is optional.

ERROR MESSAGES: A label that cannot be found will be displayed, followed by:

```
"-LABEL NOT FOUND"
```

EXAMPLE:

```
        J:*START
        .....
        .....
*START  T:ARE YOU READY TO BEGIN?
        A:
        M: YES
        JY:*QUEST
        .....
        .....
*QUEST  T:I HAVE A QUESTION FOR YOU.
        .....
```

USE SUBROUTINE (U:)

SYNTAX:

```
[<label>] U [<cond>] : [*]<destination>
```

DESCRIPTION: Jumps to the specified destination (a subroutine) in the current program and saves the current location so that control can return to the statement following USE when an END (E:) statement is encountered. The destination must reference an existing label, but the leading asterisk is optional.

LIMITATIONS: Subroutines may be nested to a maximum depth of seven.

ERROR MESSAGES: A label that cannot be found will be displayed, followed by:

```
"-LABEL NOT FOUND"
```

Too great a nesting of USE statements will cause:

```
"*USE DEPTH EXCEEDED"
```

EXAMPLE:

```
*START  T:THIS IS WHERE WE START.
        T:DO YOU NEED INSTRUCTIONS?
        A:
        M: YES
        UY:*INSTR
        .....
        .....
*INSTR  T:THIS IS WHAT YOU NEED TO KNOW.
        .....
        .....
        E:
```

END OF SUBROUTINE OR PROGRAM (E:)

SYNTAX:

```
[<label>] E [<cond>] :
```

DESCRIPTION: Indicates the end of a subroutine or the end of the current program. If encountered following a USE (U:) statement, control will be passed to the statement followng the USE statement. If there are no USE statements pending, the program will terminate and PILOT will display its initial message:

```
"PILOT-8080, 1.1
 LOAD A NEW PROGRAM?"
```

LIMITATIONS: Subroutines may be nested to a maximum depth of seven. See USE (U:) for potential error message.

EXAMPLE:

```
        *START  U:*FIRST
                T: TIME
                E:
        *FIRST  U:*SECOND
                T: THE
                E:
        *SECOND U:*THIRD
                T: IS
                E:
        *THIRD  T: NOW
                E:
```

COMPUTE (C:)

SYNTAX:

```
[<label>] C [<cond>] : <num variable> = <num expression>
```

DESCRIPTION: Computes a value based on evaluation of the numeric expression and assigns the result to the numeric variable on the left of the equal sign.

LIMITATIONS: Only the following forms of numeric expressions are allowed:

```
integer
numeric variable + integer
numeric variable - integer
numeric variable + numeric variable
numeric variable - numeric variable
```

In the present system integers are limited to values from -99 to +99. Numeric variable names are the capital letters A through Z. A longer name may be written, but only the first letter will be used.

ERROR MESSAGES: An offending expression is displayed, followed by one of the following:

```
"*ILLEGAL EXPRESSION"
"*VALUE >  99"
"*VALUE < -99"
```

The message "*ILLEGAL EXPRESSION" can occur as a result of bad syntax or in response to an illegal numeric variable name. After display of the message, a branch to keyboard entry will occur, and if "RETURN" is pressed, execution will proceed with the next PILOT line. In the case of value > 99, the value will be set to 99, and for value < -99, the value will be set to -99.

EXAMPLE:

```
        C:X = A+5
        C: X = A-B
```

REMARK (R:)

SYNTAX:

```
[<label>] R : <text>
```

DESCRIPTION: PILOT ignores the REMARK statement. It is a means of storing potentially useful information in the program listing. The REMARK statement may occur at any point in a PILOT program sequence.

EXAMPLE:

```
                R:.....................................
                R:LESSON TWO: WRITTEN BY JRG 1/2/73
                R:.....................................
        *LESSON2
                .....
                .....
```

INPUT MAXIMUM NUMBER OF CHARACTERS (INMAX:)

SYNTAX:

```
[<label>] INMAX [<cond>] : <integer>
[<label>] INMAX [<cond>] : <numeric variable>
```

DESCRIPTION: Limits the number of characters accepted by following ACCEPT statements (A:). The limit is set by providing either an integer (1-80) or a numeric variable name to the right of the colon.

Input is terminated when the limit is reached. For a useful example, if INMAX is set to 1, input will be immediate with a single character and without pressing "return".

LIMITATIONS: INMAX values should be set between 1 and 80. PILOT-8080, Version 1.1 starts with a default of 72.

EXAMPLE:

```
        T:PLEASE ENTER A FIVE DIGIT NUMBER.
          INMAX:5
        A:
        T:THANK YOU.
        T:HOW BIG A NUMBER DO YOU WISH TO ENTER?
        A:#N
        T:NOW WE WILL DO IT.
          INMAX:N
        A:
```

NEW STRING VARIABLES (NEW$:)

SYNTAX:

```
[<label>] NEW$ [<cond>] :
```

DESCRIPTION: Erases storage of any string variables that have been defined up to this point in the program. This allows recovery of the memory space they consumed. Use of this instruction may be necessary when the program sequence allows multiple use of the same string variable name.

EXAMPLE:

```
                T:GIVE YOUR ANSWER NOW.
        *ANSWER A:$TEXT
                M:PATTERN
                    N:SOMETHING'S WRONG.
                    N:PLEASE TRY THAT AGAIN.
                    NEW$N:
                    JN:*ANSWER
                T:FINE, LET'S GO ON.
                .....
```

DISPLAY PROGRAM (DP:)

SYNTAX:

```
[<label>] DP [<cond>] :
```

DESCRIPTION: Displays the PILOT program currently in memory on the console device.

This instruction is normally used in the immediate mode.

ESCAPE MECHANISM: The output code supplied in the distributed system checks the console for input between every output character. While the program is being displayed, operation may be interrupted with ctl-Z from the console. The message "*INTERRUPTED" will appear and display will cease. A choice is then available:

1. A carriage return from the console will resume display of the remaining program text.

2. An additional ctl-Z from the console will return PILOT to its initial message:

```
"PILOT-8080, 1.1
 LOAD A NEW PROGRAM?"
```

EXAMPLE:

```
        (ctl-z) DP:
```

LOAD NEW PROGRAM OR PROGRAM SEGMENT (LOAD:)

SYNTAX:

```
[<label>] LOAD [<cond>] :
```

DESCRIPTION: Loads a PILOT program into the program buffer from a storage device and initiates execution of the program.

This command is used in the immediate mode (preceded by ctl-z) in order to load a new program and also in a program sequence to replace the current portion of a long program with the next section. Ctl-Z (1AH) in the input will terminate the load operation.

LIMITATIONS: The PILOT interpreter will respond to this command by accepting input from the reader input port. If the reader is a teletype, it will accept input from the paper tape reader or from the keyboard. PILOT-8080, Version 1.1 provides a jump table for input-output vectors, however, so a user of the system can supply an address for reader input (RI) that will reference an input routine to use other equipment. The provided routine need only return with the next character in the A-register.

ERROR MESSAGES: "/OVERFLOW" indicates that there is insufficient memory space for the attempted load. Since program text and string variable storage use the same memory area, string variables from a previous program segment can cause this problem in a segmented program (see NEW$).

EXAMPLE:

```
        LOAD:
```

SAVE PROGRAM (SAVE:)

SYNTAX:

```
[<label>] SAVE [<cond>] :
```

DESCRIPTION: Saves the PILOT program currently in memory on the storage device. The default code within the PILOT interpreter for PILOT-8080, Version 1.1 sends this output to the console output port. A user of the system can supply an address for punch output (PO) within a jump table that will reference a user supplied routine to use other equipment. The next character to be stored is found in the C-register.

This instruction is normally used in the immediate mode.

ESCAPE MECHANISM: The output code supplied in the distributed system checks the console for input between every output character. While the program is being saved, operation may be interrupted with ctl-Z from the console. The message "*INTERRUPTED" will appear and output will cease. A choice is then available:

1. A carriage return from the console will resume output of the remaining program text.

2. An additional ctl-Z from the console will return PILOT to its initial message:

```
"PILOT-8080, 1.1
 LOAD A NEW PROGRAM?"
```

EXAMPLE:

```
        (ctl-z) SAVE:
```

PRINT PROGRAM (PRINT:)

SYNTAX:

```
[<label>] PRINT [<cond>] :
```

DESCRIPTION: Prints the PILOT program currently in memory on the print device. The default code within the PILOT interpreter for PILOT-8080, Version 1.1 sends this output to the console output port. A user of the system can supply an address for listing output (LO) within a jump table that will reference a user supplied routine to use other equipment. The next character to be printed is found in the C-register.

This instruction is normally used in the immediate mode.

ESCAPE MECHANISM: The output code supplied in the distributed system checks the console for input between every output character. While the program is being printed, operation may be interrupted with ctl-Z from the console. The message "*INTERRUPTED" will appear and printing will cease. A choice is then available:

1. A carriage return from the console will resume printing of the remaining program text.

2. An additional ctl-Z from the console will return PILOT to its initial message:

```
"PILOT-8080, 1.1
 LOAD A NEW PROGRAM?"
```

EXAMPLE:

```
        (ctl-z) PRINT:
```

EDIT CURRENT PROGRAM (EDIT)

SYNTAX:

```
[<label>] EDIT [<cond>] :
```

DESCRIPTION: Transfers control to an editor, external to the PILOT interpreter, in order to modify the current PILOT program.

PILOT-8080, Version 1.1 does not contain an internal editor. A user of the system can supply the entry address of an editor in PILOT'S jump table (EDIT) that will be referenced by this instruction.

The PILOT interpreter can successfully deal with program text in a variety of formats. This allows many existing editors to be used without modification.

[<char-count>][<line-no.>] <PILOT-line> CR [LF]

Optional elements are:
1. Binary count of characters in the line (one byte).
2. Line numbers in the form of four ASCII digits.
3. Line feed characters following carriage return.

Blanks following the carriage return character are ignored during the LOAD operation.

This instruction is normally used in the immediate mode.

EXAMPLE:

```
        (ctl-z) EDIT:
```

EXIT FROM PILOT (BYE:)

SYNTAX:

```
[<label>] BYE [<cond>] :
```

DESCRIPTION: Causes PILOT to terminate execution and exit to an external operation. A user of the system can supply the address of an external monitor in PILOT'S jump table (EXIT) that will be referenced by this instruction.

This instruction is normally used in the immediate mode.

EXAMPLE:

```
        (ctl-z) BYE:
```

INSTALLATION

EQUIPMENT CONFIGURATION

PILOT-8080, Version 1.1, as distributed, can be installed without modification in a minimum system consisting of an 8080 processor, 8K (8192) bytes of random-access memory addressed from 0000 to 1FFF (hex), and a teletype used as a console, printer, and program storage device. Appendix D gives instructions for loading the PILOT system from paper tape.

When PILOT starts operation, references to the provided teletype input/output code (I/O vectors) are copied from location IOJMP (2F7H) within the interpreter. This is an arrangement that allows the system to be incorporated in read-only memory and still permits a user to substitute different I/O operations for alternative equipment.

The provided I/O routines use port 0 for status and port 1 for data. Status indicators at port 0 are bit 6 (40H) for read data available and bit 7 (80H) for transmit buffer empty.

In order to provide an escape mechanism during the operation of DISPLAY PROGRAM (DP:), PRINT PROGRAM (PRINT:), and SAVE PROGRAM (SAVE:), the provided output routine checks for input of ctl-Z before each output character. If ctl-Z is received, then the output operation ceases. It is recommended that if substitute I/O programming is developed for alternative equipment, that it provide for this feature.

SYSTEM MODIFICATION

Input/output capability can be changed to match alternative peripheral equipment by writing appropriate I/O routines, each terminated with a return instruction, and placing their entry point addresses in appropriate portions of a jump table starting at location CI (100H). PILOT does not require use of memory below 80H, so that user routines may use interrupt restart locations if desired.

The following section gives details of the jump table, starting in location 100H, that allows the PILOT system to be adapted to a more extensive configuration of input/output equipment. It is possible to arrange for separate devices for loading and saving programs, for displaying and printing programs, or for a variety of console devices. In addition, the jump table allows a user to reference an external monitor, an external editor, and to provide alternative actions in the event of program text that does not match that of legal PILOT syntax. PILOT's normal action is to display such code.

PILOT RAM Loc.	User Supplied Contents	Name	Description
0100	0C3H	JMP	Subroutine to return one (1)
0101	CI Address (Lo Byte)	CI	character from the console
0102	CI Address (Hi Byte)		keyboard in the A-register.
0103	0C3H	JMP	Subroutine which accepts a
0104	CO Address (Lo Byte)	CO	character from the C-register
0105	CO Address (Hi Byte)		and outputs it to the console
0106	0C3H	JMP	Subroutine to return one (1)
0107	RI Address (Lo Byte)	RI	byte from the reader device
0108	RI Address (Hi Byte)		in the A-register.
0109	0C3H	JMP	Subroutine which accepts a
010A	LO Address (Lo Byte)	LO	character from the C-register
010B	LO Address (Hi Byte)		and outputs it to the printer
010C	0C3H	JMP	Subroutine which accepts a
010D	PO Address (Lo Byte)	PO	character from the C-register
010E	PO Address (Hi Byte)		and outputs it to the storage device.
010F	0C3H	JMP	Entry point to user's monitor
0110	EXIT Address (Lo)	EXIT	program. Used by the PILOT
0111	EXIT Address (Hi)		statement BYE:
0112	0C3H	JMP	Entry point to user's editor.
0113	EDIT Address (Hi)	EDIT	Used by the PILOT statement
0114	EDIT Address (Hi)		EDIT:
0115	0C3H	JMP	Subroutine to provide altern-
0116	ASCAN Address (Lo)	ASCAN	ative action for illegal
0117	ASCAN Address (Hi)		PILOT statements.

If changes have been made to the jump table so that it is not desirable to have the vectors reset, the PILOT interpreter should be entered at the label RSTRT instead of PSTRT. As distributed the address of RSTRT is 30FH. If the system is being reassembled, an alternate jump to RSTRT might be placed at location 43H, and code at this location can also make desired changes to the jump table. As an example in the distributed system, a few lines of code at location 43H can be seen to route all output to an external video driver that expects to find each output character in the B register.

Appendix B describes memory allocation for the system and the options available that allow change in the size of the program buffer or change in the address location of either the program buffer, the PILOT interpreter, or the system data area. Normal entry to start the PILOT system is at location 40H. This location contains a jump to the beginning of the PILOT interpreter at PSTRT (START), which as distributed is at 2E0H. One of the first things PILOT does at this starting point is to set up a jump table of input-output vectors beginning at location 100H.

The following values can be changed at the time of assembly by a change to equate statements at the beginning of the PILOT system:

	NAME	EFFECT	VALUE PROVIDED (HEX)
	ENTRY	Normal entry address	0040
	STKPR	Stack pointer, jump table address	0100
	PSTRT	Start of PILOT interpreter	02E0
	PBUFB	Beginning of program buffer	1000
	PBUFE	End of program buffer	1FFF
	VIDEO	External video display driver	FE77
	MNTR	External monitor address	E060
also:			
	LINE	Maximum starting line length	72(dec)

APPENDIX A.

PILOT LANGUAGE DEFINITION

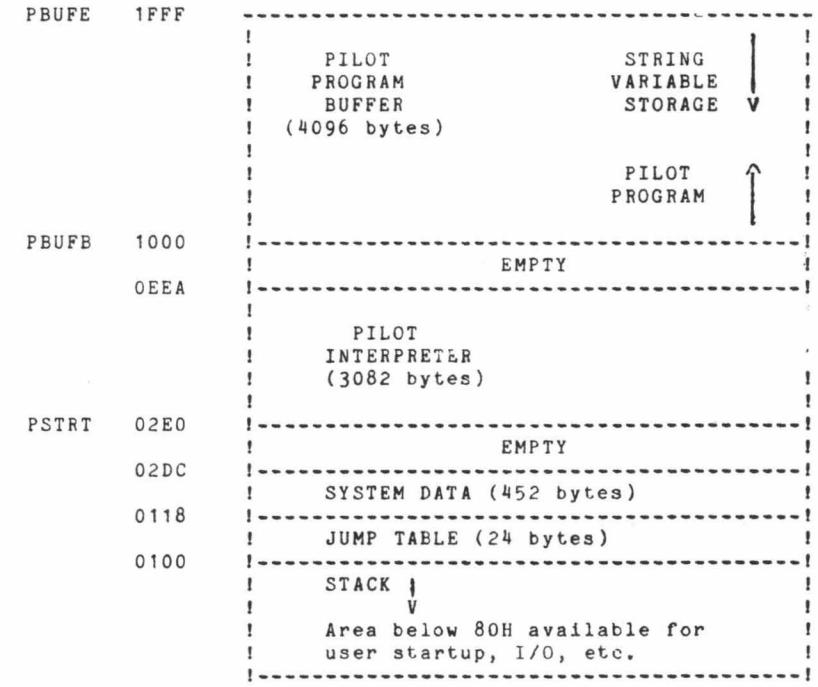
The following is a formal definition of PILOT-8080, Version 1.1 Items contained within "<" and ">" are elements of PILOT. Items contained within "[" and "]" are optional. The symbol (!) is used to indicate an "OR" relationship.

```
<PILOT line>
  ::= *<label>
    ! [*<label>]<PILOT statement>
        <label>s begin immediately after the asterisk and
        are terminated by the line's end (carriage return)
        or by the first blank.

<label>
  ::= <identifier>
<PILOT statement>
  ::= <opcode>[<cond>]:<expression>
<opcode>
  ::= <core opcode>
    ! <keyword>
    ! <continuation>
<core opcode> is a single letter.
        In the present system we have -
  ::= T ! A ! M ! J ! U ! E ! C ! R ! Y ! N
<keyword> is a multi-letter code not ending in Y or N.
        In the present system we have -
  ::= MC ! INMAX ! NEW$ ! DP ! LOAD ! SAVE ! PRINT
        ! EDIT ! BYE
<continuation>
  ::= :
<cond>
  ::= Y ! N ! (<numeric variable>) ! (<integer>)
<expression>
  ::= <message> ! <variable> ! <pattern>
        ! <destination> ! <assign statement>

        Different <opcode>s and <expression>s are shown
        in the context of <PILOT statement>s below.

<PILOT statement>
  ::= T [<cond>]:<message>
        TYPE- The specified message is displayed at the console.
    ! A [<cond>]:[<variable>]
        ACCEPT- A response is read and saved in a temporary
        response buffer. If a variable name is specified,
        it is stored under that name.
    ! M [<cond>]:<pattern>[,<pattern>... ,<pattern>]
        MATCH- An internal Y-switch is set to true if the
        response buffer matches any pattern provided in the
        M-statement. Comma (,) is used to separate
        multiple patterns.
    ! J [<cond>]:[*]<destination>
        JUMP- Passes control to the line referenced by
        <destination>.
    ! U [<cond>]:[*]<destination>
        USE SUBROUTINE- Passes control to the line referenced
        by <destination>, pushing the location of the following
        line on the subroutine stack.
    ! E [<cond>]:
        END- Passes control to the location popped off the
        subroutine stack, or if the stack is empty, terminates
        the PILOT program.
    ! C [<cond>]: <assign statement>
        COMPUTE- Executes an assignment statement.
    ! R [<cond>]: <text>
        REMARK- A comment included by the author is ignored.
    ! :
        CONTINUATION- The <opcode> and <cond> from the last used
        <PILOT statement> are reused.
    ! Y:
        YES- Abbreviated form of TY:
    ! N:
        NO- Abbreviated form of TN:
    ! MC [<cond>]: <pattern>[^<pattern>...^<pattern>]
```

```
        MATCH COMMAS- Operates in the same manner as "M"
        except that <pattern> can contain commas (,).
        The caret (^) is used to separate multiple patterns.
    ! INMAX [<cond>]: numeric variable ! integer
        INPUT MAXIMUM- Sets the maximum number of characters
        accepted as input (opcode A) from the console.
        In the present system, a response is automatically
        terminated when the limit is reached.
    ! NEW$ [<cond>]:
        NEW STRING VARIABLES- Deletes existing storage of
        string variable names and data.
    ! DP [<cond>]:
        DISPLAY PROGRAM- Displays the PILOT program currently
        in memory.
    ! LOAD [<cond>]:
        LOAD PILOT PROGRAM- Overlays any existing PILOT program
        in memory with a new program segment. Stored information
        in the form of <variable> is not disturbed.
    ! SAVE [<cond>]:
        SAVE PROGRAM- Saves the PILOT program currently in
        memory on the storage device.
    ! PRINT [<cond>]:
        PRINT PROGRAM- Prints the PILOT program currently in
        memory on the print device.
    ! EDIT [<cond>]:
        EDIT PROGRAM- Transfers control to an external editor.
    ! BYE [<cond>]:
        BYE- Leaves the PILOT system and transfers control
        to an external monitor.

<message>
  ::= [<text>] [<variable>] [<text>]
<variable>
  ::= $<string variable>
    ! #<num variable>
<pattern>
  ::= <text>
<destination>
  ::= <label>
<assign statement>
  ::= num variable = num variable + integer
    ! num variable = num variable - integer
    ! num variable = num variable + num variable
    ! num variable = num variable - num variable
<string variable>
  ::= <identifier>
        an identifier that begins immediately after a
        dollar sign ($) and terminates with the first blank
        or with the end of line.
<num variable>
  ::= A ! B ! C ! .... Z
        a single letter A-Z that begins immediately after a
        pound sign (#).
<identifier>
        an alphanumeric name limited to 10 characters.
<integer>
        integers are limited to values from -99 to +99
<text>
        a character string limited to 80 characters.
```

APPENDIX B.

PILOT SYSTEM MEMORY ALLOCATION IN THE DISTRIBUTED SYSTEM



MODIFICATION OF ADDRESSES AND BUFFER SIZE

PSTRT - PILOT INTERPRETER STARTING ADDRESS
This address can be set at the time of assembly by a change in the equate statement at the beginning of the program.

PBUFB - PROGRAM BUFFER BEGINNING
This address can be set at the time of assembly by a change in the equate statement at the beginning of the program.

PBUFE - PROGRAM BUFFER END
This address can be set at the time of assembly by a change in the equate statement at the beginning of the program. In addition, this address can be changed at any time by modifying the object code at location INIT+10 (location 360H in the distributed version).

The PILOT interpreter portion (2E0H-EE9H as distributed) contains code that is not modified at any time. The interpreter can therefore be placed in read-only memory if desired and, as noted above, can be placed at any address location.

APPENDIX C.

```
T:TEST PROGRAM OF PILOT-8080 FUNCTIONS
T:
T:YOU CAN TRY PILOT CORE INSTRUCTIONS:
:  T:  TO PRESENT TEXT.
:  A:  TO ACCEPT AN ANSWER.
:  M:  TO MATCH ELEMENTS OF AN ANSWER.
:  J:  TO JUMP TO A LABELED PLACE.
:  U:  TO USE A SUBROUTINE.
:  E:  TO END A SUBROUTINE OR THE ENTIRE PROGRAM.
:  C:  FOR LIMITED COMPUTATION.
:  R:  TO INSERT A REMARK WITH NO OPERATIONAL EFFECT.
:
:BEGIN BY TYPING ANYTHING, THEN PRESS 'RETURN'
:  USE DELETE (SHIFT O) OR RUBOUT TO ERASE ONE LETTER, AND
:  CTL/U TO KILL A LINE BEFORE YOU HAVE PRESSED RETURN.
A:$WHAT
T:YOU TYPED $WHAT.
T:$WHAT IS WHAT YOU TYPED.
T:    PRESS 'RETURN' TO CONTINUE.
A:
T:
:Y OR N APPENDED TO ANY INSTRUCTION MAKES ITS USE CONDITIONAL
 :  UPON THE LAST ATTEMPTED MATCH.
:Y: OR N: ALONE ARE SHORTHAND VERSIONS OF TY: OR TN:
T:
*TESTM
T:I WILL LOOK FOR 'ABC', ' DEF', 'GHI ', OR ' JKL '
T:NOTE THE SPACES.  THEY ARE IMPORTANT IF THE LETTERS WE ARE
 T:TRYING TO MATCH ARE NOT AT EITHER END OF A LINE.
A:
M:ABC, DEF,GHI , JKL ,
TY:MATCH
TN:NO MATCH
   U:*ASK
   JY:*TESTM
   J:*TESTC

*ASK
T:AGAIN? (Y OR N)
A:
M: Y
E:
*TESTC
T:TYPE A NUMBER FROM 0 TO 99
T:I USE THE FIRST TWO DIGITS ENTERED.
   A:#A
T:ANOTHER NUMBER PLEASE
   A:#B
T:YOUR NUMBERS ARE #A AND #B.
T:I WILL SUBTRACT THEM.
   C:C=A-B
T:#A - #B = #C
T:
T:     TYPE 'NEXT' TO GO ON
A:
M:NEXT
   JN:*TESTC
T:
```

```
*TESTPLUS
   C:A=0
   C:B=0
T:TESTS CAN BE CONDITIONAL ON A NUMERIC VALUE > 0
T:A NUMERIC VARIABLE IS APPENDED AS IN 'J(X):*LABEL'
 T:TYPE 'ABC' OR 'DEF'
A:
M:ABC
   CY:A=1
M:DEF
   CY:B=1
T(A):ABC FOUND
T(B):DEF FOUND
   C:C=A+B
   J(C):*END
T:NO MATCH OF EITHER ABC OR DEF

*END
T:END OF TEST
   U:*ASK
   JY:*TESTPLUS
T:
:NINE OTHER INSTRUCTIONS ARE AVAILABLE IN THIS SYSTEM:
:   MC:      TO LOOK FOR TEXT WITH COMMAS IN IT.
:   INMAX:   TO LIMIT THE NUMBER OF CHARACTERS ACCEPTED.
:   NEW$     TO ERASE STORED $TEXT (STRING VARIABLES).
:   DP:      TO DISPLAY THE PROGRAM AT THE CONSOLE.
:   LOAD:    TO LOAD A NEW PROGRAM. ($TEXT REMAINS).
:   SAVE     TO SEND THE PROGRAM TO A STORAGE DEVICE.
:   PRINT:   TO SEND THE PROGRAM TO A LISTING DEVICE.
:   EDIT:    TO LEAVE PILOT AND USE AN EDITOR.
:   BYE:     TO LEAVE PILOT AND GO TO A MONITOR.
   T:
   T:YOU CAN LOOK FOR TEXT WITH COMMAS IN IT.
   T:A CARET '↑' (SHIFT N) IS USED AS A SEPARATOR FOR MC:
 *TESTMC
   T:I WILL LOOK FOR 'ABC,D' OR 'A,BC '
A:$ENTRY

MC:ABC,D↑A,BC
Y:YES, $ENTRY IS OK.
N:NO, YOU TYPED $ENTRY INSTEAD OF THE ABOVE.
   U:*ASK
   JY:*TESTMC
T:
*TESTIN
T:INMAX:1 WILL BE USED TO LOOK FOR 'A'.
T:PRESS ONLY A SINGLE KEY.  DO NOT PRESS 'RETURN'
T:      ----                    ---
   INMAX:1
    A:
M:A
Y:YOU TYPED 'A'
N:THAT WASN'T 'A'
T:AGAIN? (Y OR N)
   A:
   INMAX:72
   M: Y
   JY:*TESTIN
T:
*ERRORS
```

```
T:I WILL SHOW SOME ERROR MESSAGES.  AFTER EACH, PRESS 'RETURN'
T:I WILL TRY TO JUMP TO AN UNKNOWN LABEL '*HERE'
T:PRESS 'RETURN'
A:
   J:*HERE
T:
T:A C: INSTRUCTION HAS BAD SYNTAX.
T:    PRESS 'RETURN'
A:
     C:X NOT Y
T:THE EXPRESSION FIELD IS DISPLAYED + ERROR MESSAGE.
T:   PRESS 'RETURN'
A:
T:
T:THE VALUE OF X WILL BECOME GREATER THAN 99.
T:PRESS 'RETURN'
A:
   C:X=99
   C:X=X+10
T:THE STATEMENT CAUSING OVERFLOW IS DISPLAYED.
T:THE VALUE OF X WAS LEFT AT #X
T:A SIMILAR MESSAGE APPEARS WHEN X BECOMES SMALLER THAN -99.
T:PRESS 'RETURN'
A:
T:TYPE 'R' TO REPEAT ERROR MESSAGES.
   INMAX:1
A:
   INMAX:72
M:R
   JY:*ERRORS
T:
T:AN ILLEGAL STATEMENT IS SIMPLY DISPLAYED.
   MW:*
T:MW: IS NOT LEGAL IN THIS SYSTEM.
T:
T:YOU CAN OBTAIN IMMEDIATE OPERATION OF ANY INSTRUCTION BY
 T:PRECEDING IT WITH CTL Z.   FOR EXAMPLE,'CTL Z'DP:
T:    'CTL Z'DP: WILL DISPLAY THE CURRENT PROGRAM IN MEMORY.
T:    'CTL Z'J:*LABEL  WILL JUMP IMMEDIATELY TO *LABEL.
T:CTL Z BY ITSELF WILL RETURN PILOT TO ITS STARTUP POINT.
T:
T:TESTS ARE NOW COMPLETE.
A:
```

APPENDIX D.

LOADING THE PILOT SYSTEM

The paper tape form of PILOT object code is in check summed Intel hex format. The next page provides a listing of a loader capable of loading PILOT. This loader should be entered into locations 1FAD (hex) thru 1FFF(hex).

The loader reads input status (receiver data available) from port zero (0) and waits for data bit six (40H) to come true. If the status bit assignment for receiver data available on your equipment is not data bit six, you will need to change the bit mask at location 1FF5 (hex) to match your needs.

Use the loader to enter PILOT into memory and run PILOT starting at location 40 (hex).

```
1FAD                    0001 *** INTEL HEX FORMAT LOADER ***
1FAD                    0002 *
1FAD                    0003        ORG     1FADH
1FAD                    0004 SP     EQU     6H
1FAD                    0005 *
1FAD 31 AD 1F           0006        LXI     SP,1FADH
1FB0 CD B3 1F           0007        CALL    READ
1FB3 CD F2 1F           0008 READ   CALL    TTYIN
1FB6 FE 3A              0009        CPI     ':'
1FB8 C2 B3 1F           0010        JNZ     READ
1FBB CD D7 1F           0011        CALL    CHAR
1FBE 57                 0012        MOV     D,A
1FBF C8                 0013        RZ
1FC0 CD D7 1F           0014        CALL    CHAR
1FC3 67                 0015        MOV     H,A
1FC4 CD D7 1F           0016        CALL    CHAR
1FC7 6F                 0017        MOV     L,A
1FC8 CD D7 1F           0018        CALL    CHAR
1FCB CD D7 1F           0019 LOOP   CALL    CHAR
1FCE 77                 0020        MOV     M,A
1FCF 23                 0021        INX     H
1FD0 15                 0022        DCR     D
1FD1 C2 CB 1F           0023        JNZ     LOOP
1FD4 C3 B3 1F           0024        JMP     READ
1FD7                    0025 *
1FD7 CD F2 1F           0026 CHAR   CALL    TTYIN
1FDA CD EA 1F           0027        CALL    HEX
1FDD 07                 0028        RLC
1FDE 17                 0029        RAL
1FDF 17                 0030        RAL
1FE0 17                 0031        RAL
1FE1 5F                 0032        MOV     E,A
1FE2 CD F2 1F           0033        CALL    TTYIN
1FE5 CD EA 1F           0034        CALL    HEX
1FE8 83                 0035        ADD     E
1FE9 C9                 0036        RET
1FEA                    0037 *
1FEA D6 30              0038 HEX    SUI     48
1FEC FE 0A              0039        CPI     10
1FEE D8                 0040        RC
1FEF D6 07              0041        SUI     7
1FF1 C9                 0042        RET
1FF2                    0043 *
1FF2 DB 00              0044 TTYIN  IN      0
1FF4 E6 40              0045        ANI     64
1FF6 CA F2 1F           0046        JZ      TTYIN
1FF9 DB 01              0047        IN      1
1FFB 00                 0048        NOP     .       ;REPLACE WITH OUT 1
1FFC 00                 0049        NOP     .       ;TO ECHO PRINT
1FFD E6 7F              0050        ANI     127
1FFF C9                 0051        RET
```

A 650X PROGRAM RELOCATER

Gentlemen: Dec. 31, 1976

I recently sent you a program relocater for the MOS Technology 650X Microprocessor. Unfortunately, I have discovered two typo errors in the listing I sent you. The errors occur in Line 210 and Line 360. These lines should read:

```
210 OEO2 B5 05        S.10: LDA TEMP1+1;
                            Set Pointers
360 OEID 29 9F            And #$ 9F
```

These corrections make the earlier version I sent you work correctly.

Recently, this program was modified to enable easier data entry into the relocater. It also was changed to permit easier relocation of programs with imbedded data or text provided the user has an easy means to zero portions of memory. As it is written, the relocater will not relocate a break instruction (00_H) but merely skips over that instruction and continues.

To relocated programs with imbedded data or text,

1. Move the entire program as if the program were all data.
2. Go back to the original program and replace all data & text with the break (00_H) instruction.
3. Relocate the entire program. When the relocater finds the 00_H in the old program, both pointers are incremented and the relocation continues. Since the program had been previously moved as data, all data and text in the program had already been moved and are unchanged.

The program may also be used in the same manner as the earlier version where data & text are moved separately from the instructions but the user is cautioned that all break instructions are ignored. These can be manually inserted.

The user can also easily modify the program to relocate break instructions by changing line OEI9 to read:

```
OEI9 4C IF OE      JMP MOV 1.
```

DATA ENTRY:

0000 H — Relocate/Move Flag

00 in this byte for relocate
01 in this byte for move

0001 LSB — Low Bound for addresses
0002 MSB

0003 LSB — High Bound for addresses
0004 MSB

Note: Any addresses within these bounds are recomputed during a relocation.

0005 LSB — Low address of program to be moved
0006 MSB

0007 LSB — High address of program to be moved
0008 MSB

0009 LSB — Destination address
000A MSB

I am sorry that errors were in the earlier version that I sent you. Contact me if this article is of interest and if I can supply any additional information.

Sincerely,
Ralph Sherman

15 Hydaway Drive
Forest, Virginia 24551

PROGRAM EXAMPLE USING TIM TO RELOCATE THE RELOCATER FROM 0E00 to 0200

```
. M  0000   XX XX XX XX XX XX XX XX
. :  0000   01 00 0

. M  0000   XX  XX  XX  XX  XX  XX  XX  XX
. :  0000    01  00  0E  F2  0E  00  0E  F2
. :  0008   OE  oo  02  XX  XX  XX  XX  XX

. R  7052   XX  XX  XX  XX  XX.
. :  0E00   04
. G
  *  0E00   XX  XX  XX  XX  XX
```

This has moved the entire relocater down to 0200 as data – (the break instruction is now moved).

Then change the byte in 000H to 00.

```
. H  0000   01   00   0E   F2   0E   00   0E   F2
. :  0000   00 (carret)

. R  XXXX  XX  XX  XX  XX  XX
. :  0E00   04
. G
. *  0E78   XX  XX  XX  XX  XX
```

The relocation is now complete.

Hint: Testing $000F_H$ shows the next available location after the relocated program.

```
M  000F   F3   02   XX  XX  XX  XX  XX  XX
```

```
1000          ;6502 PROGRAM RELOCATOR
1005          ;AUTHOR R. HERMAN
1010          ;REV 0, DEC 31,1976
1012          ;
1014          ;REMOVE LINE '.130-1140 TO 'OL' JW SRL' MOVE
1015 0000           RFI AD: .=0   ;=1 TO FORCE 1 FOR PATH MOVE
1020 0001           TEST1:  .ORG  **+2  ;LOW BOUND FOR ADDRESSES
1025 0003           TEST2:  .ORG  **+2  ;HI  BOUND
1030 0005           START:  .ORG  **+2  ;LOW ADDRESS FOR MOVE
1035 0007           STOP:   .ORG  **+2  ;HI  ADDRESS
1040 0009           DEC:    .ORG  **+2  ;DESTINATION ADDRESS
1045 000B           OPTR:   .ORG  **+2
1050 000D           TEMP2:  .ORG  **+2
1055 000F           NPTR:   .ORG  **+2
1060 0011           TEMP1:  .ORG  **+2
1065 0013                   .ORG  $E00
1070 0E00 A2 05     BEGIN:  LDX   #5
1075 0E02 B5 05     S.10:   LDA   START+X ;SET POINTERS
1080 0E04 95 0B             STA   OPTR,X
1085 0E06 CA                DEX
1090 0E07 10 F9             BPL   S.10
1095 0E09 E8                INX
1100 0E0A A5 00     MOVE:   LDA   RFLAG ;OK FOR DATA MOVE
1105 0E0C F0 06             BEQ   NO.1 ;NO
1110 0E0E 20 4F 0E          JSR   MOV1 ;YES
1115 0E11 4C 60 0E          JMP   DONE?
1120 0E14 A1 0B     NO.1:   LDA   (OPTR,X)
1125 0E16 A8                TAY   ;SAVE OP CODE
1130 0E17 D0 06             BNE   NO.2 ;DO NOT MOVE BP
1135 0E19 20 53 0E          JSR   SKIP
1140 0E1C 4C 60 0E          JMP   DONE?
1145 0E1F 20 4F 0E  NO.2:   JSR   MOV1
1150 0E22 98        TSTJSR: TYA   ;RESTORE A/TL
1155 0E23 C9 20             CMP   #$20 ;JSR?
1160 0E25 D0 03             BNE   *+5  ;BYTE
1165 0E27 4C 7A 0E          JMP   .3BYTE
1170          ;TESTS FOR 1 BYT, 2 INSTRUCTIONS
1172 0E2A 98                TYA   ;RESTORE BYTE
```

```
1180 0E2B 29 9F           AND    #$9F
1185 0E2D F0 31           BEQ    DONE?
1190 0E2F 98              TYA    ; RESTORE BYTE
1195 0E30 29 1D           AND    #$1D
1200 0E32 C9 08           CMP    #$8
1205 0E34 F0 2A           BEQ    DONE?
1210 0E36 C9 18           CMP    #$18
1215 0E38 F0 26           BEQ    DONE?
1220      ; TEST FOR 3 BYTE INSTRUCTION
1225 0E3A 98              TYA    ; RESTORE BYTE
1230 0E3B 29 1C           AND    #$1C
1235 0E3D C9 1C           CMP    #$1C
1240 0E3F F0 39           BEQ    .3BYTE
1245 0E41 C9 18           CMP    #$18
1250 0E43 F0 35           BEQ    .3BYTE
1255 0E45 C9 0C           CMP    #$0C
1260 0E47 F0 31           BEQ    .3BYTE
1265      ;ALL REMAINING ARE 2 BYTE INSTRUCTIONS
1270 0E49 20 4F 0E        JSR    MOV1
1275 0E4C 4C 60 0E        JMP    DONE?
1280      ;MOVE 1 BYTE
1285 0E4F A1 0B    MOV1:  LDA    (OPTR,X)
1290 0E51 81 0F           STA    (NPTR,X)
1295 0E53 20 DA 0E SKIP:  JSR    IOPTR
1300 0E56 20 E1 0E        JSR    INPTR
1305 0E59 60              RTS
1310      ; MOVE 2 BYTES AND TEST DONE?
1315 0E5A 20 4F 0E MOV2:  JSR    MOV1
1320 0E5D 20 4F 0E        JSR    MOV1
1325 0E60 A5 0B    DONE?: LDA    OPTR ; CHECK OPTR VS S"OR ADDRESS
1330 0E62 85 11           STA    TEMP1
1335 0E64 A5 0C           LDA    OPTR+1
1340 0E66 85 12           STA    TEMP1+1
1345 0E68 A5 07           LDA    STOP
1350 0E6A 85 0D           STA    TEMP2
1355 0E6C A5 08           LDA    STOP+1
1360 0E6E 85 0E           STA    TEMP2+1
1365 0E70 20 CF 0E        JSR    TEST ; TEST ADDRESSES
1370 0E73 90 95           BCC    MOVE
1375 0E75 F0 93           BEQ    MOVE
1380 0E77 00              BRK
1385 0E78 EA              NOP
1390 0E79 EA              NOP
1395      ; 3 BYTE INSTRUCTIONS  TEST FOR ADDRESS RECOMPUTATION
1400 0E7A A1 0B    .3BYTE:LDA    (OPTR,X) ; TEST ADDRESS
1405 0E7C 85 11           STA    TEMP1
1410 0E7E 20 DA 0E        JSR    IOPTR
1415 0E81 A1 0B           LDA    (OPTR,X)
1420 0E83 85 12           STA    TEMP1+1
1425 0E85 20 E8 0E        JSR    DOPTR
1430 0E88 A5 01           LDA    TEST1
1435 0E8A 85 0D           STA    TEMP2
1440 0E8C A5 02           LDA    TEST1+1
1445 0E8E 85 0E           STA    TEMP2+1
1450 0E90 20 CF 0E        JSR    TEST
1455 0E93 F0 02           BEQ    B.10 ; MSB EQUAL
1460 0E95 90 C3           BCC    MOV2 ; ADDRESS < TEST1
1465 0E97 A5 03    B.10:  LDA    TEST2
1470 0E99 85 0D           STA    TEMP2
1475 0E9B A5 04           LDA    TEST2+1
1480 0E9D 85 0E           STA    TEMP2+1
1485 0E9F 20 CF 0E        JSR    TEST
1490 0EA2 F0 02           BEQ    B.20 ; LSB EQUAL
1495 0EA4 B0 B4           BCS    MOV2 ; ADDRESS > TEST2
1500      ; ELSE ADDRESS REQUIRES RECOMPUTATION
1505 0EA6 38       B.20:  SEC
1510 0EA7 A1 0B           LDA    (OPTR,X) ; COMPUTE ADDRESS INCREMENT FRO
1515 0EA9 E5 05           SBC    START
1520 0EAB 85 0D           STA    TEMP2
1525 0EAD 20 DA 0E        JSR    IOPTR
1530 0EB0 A1 0B           LDA    (OPTR,X)
1535 0EB2 E5 06           SBC    START+1
1540 0EB4 85 0E           STA    TEMP2+1
1545 0EB6 20 DA 0E        JSR    IOPTR
1550 0EB9 18              CLC    ; ADD INCREMENT TO NEW PROGRAM BEGIN ADDR
1555 0EBA A5 0D           LDA    TEMP2
1560 0EBC 65 09           ADC    BEG
1565 0EBE 81 0F           STA    (NPTR,X) ; STORE ADDRESS IN NEW PROGRAM
1570 0EC0 20 E1 0E        JSR    INPTR
1575 0EC3 A5 0E           LDA    TEMP2+1
1580 0EC5 65 0A           ADC    BEG+1
1585 0EC7 81 0F           STA    (NPTR,X)
1590 0EC9 20 E1 0E        JSR    INPTR
1595 0ECC 4C 60 0E        JMP    DONE?
1600      ; TEST COMPARES 2 ADDRESSES MSB THEN LSB
1605      ; C=0 TEMP1<=TEMP2
1610      ; C=1 TEMP1 >TEMP2
1615      ; Z=1 TEMP1 =TEMP2
1620 0ECF A5 12    TEST:  LDA    TEMP1+1 ; TEST MSB
1625 0ED1 C5 0E           CMP    TEMP2+1
1630 0ED3 D0 04           BNE    T.10
1635 0ED5 A5 11           LDA    TEMP1 ; TEST LSB
1640 0ED7 C5 0D           CMP    TEMP2
1645 0ED9 60       T.10:  RTS
1650                      .FILE  1
2000      ; INCREMENT OLD PROGRAM POINTER
2010 0EDA E6 0B    IOPTR: INC    OPTR
2020 0EDC D0 02           BNE    INC.10
2030 0EDE E6 0C           INC    OPTR+1
2040 0EE0 60       INC.10:RTS
2050      ; INCREMENT NEW PROGRAM POINTER
2060 0EE1 E6 0F    INPTR: INC    NPTR
2070 0EE3 D0 02           BNE    INC.20
2080 0EE5 E6 10           INC    NPTR+1
2090 0EE7 60       INC.20:RTS
2100      ;DECREMENT OLD PROGRAM POINTER
2110 0EE8 C6 0B    DOPTR: DEC    OPTR
2120 0EEA A5 0B           LDA    OPTR
2130 0EEC C9 FF           CMP    #$FF ; CROSS PAGE BOUNDARY?
2140 0EEE D0 02           BNE    D.10
2150 0EF0 C6 0C           DEC    OPTR+1
2160 0EF2 60       D.10:  RTS
2170                      .END
```

ERRORS IN ASSY

SYMBOL TABLE

.1BYTE	0E2A	INC.10	0EE0	NPTR	000F	TEMP2	000D
.3BYTE	0E7A	INC.20	0EE7	OPTR	000B	TEST	0ECF
D.10	0E97	INPTR	0EE1	RFLAG	0000	TEST1	0001
B.20	0EA6	IOPTR	0EDA	S.10	0E02	TEST2	0003
BEG	0009	MO.1	0E14	SKIP	0E53	TSTJSR	0E22
BEGIN	0E00	MO.2	0E1F	START	0005		
D.10	0EF2	MOV1	0E4F	STOP	0007		
DONE?	0E60	MOV2	0E5A	T.10	0ED9		
DOPTR	0EE8	MOVE	0E0A	TEMP1	0011		

32 SYMBOLS USED

HEX LOAD TAPE

```
C 31-1976
MOV0
C0E00A205B505950BCA10F9E8A500F0A6204F0E4C600EA10BA8D0062053E0E0A63
D0E1920530E4C600E204F0E98C920D0034C7A0E98299FF031982910C908F02A0A6E
D0E34F02AC918F02698291CC91CF039C918F035C90CF031204F0E4C600EA10B0C3A
D0E4FA10B810F20DA0E20E10E60204F0E204F0EA50B8511A50C8512A507850D08F3
C0ECA850DA508850E20CF0E9095F09300EAEAA10B851120DA0EA10B85122E80E0C84
D0E8520E80EA501850DA502850E20CF0EF00290C3A503850DA504850E20CF0E0AED
D0E9F20CF0EF002B0B438A10BE505850D20DA0EA10BE506850E20DA0E18A50D0B81
D0EBAA50D6509810F20E10EA50E650A810F20E10E4C600EA512C50ED004A5110A33
50ED5A511C50D6002D0
MOV1
90EDAE60BD002E60C60E60FD002E61060C60BA50BC9FFD002C60C600C80
```

Dear Jim, February 23, 1977

I have submitted this to the *Kim-1 User Notes* but thought that it might deserve more widespread distribution. People with other 6502 or 6800 systems might find it useful if they are going to interface to the SWTP GT6144. I therefore submit it for your consideration also.

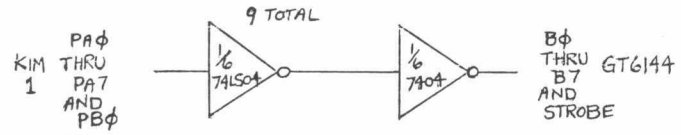
Yours truly,

Stan Ockers

RR 4, Box 209
Lockport, IL 60441

I RECENTLY ADDED THE SOUTHWEST TECHNICAL GRAPHICS BOARD TO MY KIM-1. THERE IS NOT MUCH RESOLUTION (60X96) BUT IT IS RELATIVELY CHEAP ($100) AND GIVES A NICE ROCK STEADY DISPLAY WITH NO FLICKERING OR WHITE FLASHES ON ENTRY OF NEW INFORMATION. YOU QUICKLY COME TO REALIZE THAT GOOD RESOLUTION TAKES LOTS OF MEMORY. I'M ATTEMPTING PRIMARILY TO USE ONLY THAT CONTAINED IN THE BASIC KIM-1.

THE KIT SHOULD NOT PROVIDE ANY PROBLEMS TO SOMEONE WITH SOME KIT BUILDING EXPERIENCE. SOCKETS ARE NOT PROVIDED AND IT TURNS OUT THAT WHILE I USED SOME, IT WOULDN'T HAVE BEEN NECESSARY. I INITIALLY HOOKED THE INPUT DIRECTLY TO THE KIM OUTPUT PORTS. IT WORKED, BUT I LATER REALIZED THAT SOME OF THESE INPUTS REPRESENTED 2 OR 3 TTL LOADS SO I ADDED BUFFERS AS SHOWN IN THE FIGURE BELOW.



I USED PORT A TO OUTPUT INFORMATION AND PB0 TO OUTPUT THE STROBE.

ONE PROBLEM I HAD WAS THAT I DIDN'T REALIZE FOR A WHILE THAT EVERY TIME I RESET THE KIM THE GRAPHICS BOARD WENT INTO A DISABLE GRAPHICS MODE. IT HAS TO BE SENT A CERTAIN CODE TO ENABLE IT AGAIN. I HAVE DONE THIS IN THE INITILIZATION ROUTINE BUT IT CAN BE DONE BY GROUNDING A CERTAIN PIN ON THE BOARD.

THERE IS NO R.F. GENERATOR ON THE BOARD. I WENT DIRECTLY INTO THE BASE OF THE VIDEO DRIVER ON A CHEAP T.V. SET THROUGH AN ELECTROLYTIC (150 MFD) CAPACITOR. THE BRIGHTNESS HAS TO BE ADJUSTED WHEN GOING FROM BLACK ON WHITE TO WHITE ON BLACK BUT OTHERWISE IT SEEMS TO WORK O.K.

THE T.V. SET DOES HAVE A TRANFORMER IN IT TO ISOLATE FROM THE A.C. LINE!

I'M WORKING ON A NUMBER OF BASIC (ELEMENTARY) ROUTINES TO BE USED IN GAMES ETC. WITH THE BOARD. SOME ARE INCLUDED IN THE T.V. SKETCH PROGRAM WHICH FOLLOWS. THIS PROGRAM ALLOWS YOU TO SKETCH THINGS ON THE T.V. USING THE SQUARE OF BUTTONS SURROUNDING THE "5" ON THE KIM KEYBOARD. EACH KEY MAKES THE LINE GO VERTICALLY, HORIZONTALLY OR DIAGONALLY IN THE DIRECTION INDICATED BY THE LOCATION OF THAT KEY. KEY "5" IS A FLIP-FLOP FOR SWITCHING FROM WRITE TO NO-WRITE MODE. A BLINKING CURSOR ALLOWS YOU TO KNOW WHERE YOU ARE WHEN NOT WRITING. KEY "E" IS FOR NORMAL WHITE ON BLACK WHILE KEY "F" WILL SWITCH YOU TO A BLACK ON WHITE MODE. KEY "B" WILL BLANK THE SCREEN WITH THE CURSOR REMAINING IN THE SAME SPOT.

```
                ***** T.V. SKETCH PROGRAM *****

0200            D8          CLD
  01        A9 60           LDA #$60        ; INITILIZE SPEED
  03        85 EA           STA 00EA
  05        A9 00           LDA #$00        ; INITILIZE BLINK FLAG
  07        85 E9           STA 00E9
  09     20 E5 02           JSR INIT        ; INITILIZE GRAPHICS
  0C     20 F8 02           JSR BLNK        ; BLANK SCREEN
  0F     20 AD 02  CURS     JSR SPON        ; START OF CURSOR
  12     20 26 03  RETN     JSR DELA        ; DELAY
  15     20 40 1F  KEYD     JSR KEYIN       ; IS A KEY DOWN?
  18        D0 08           BNE WHAT        ; YES, FIND OUT WHICH
  1A     20 A9 02           JSR SPOF        ; TURN OFF SPOT
  1D     20 26 03           JSR DELA
  20        D0 ED           BNE CURS        ; UNCOND. JUMP
  22     20 6A 1F  WHAT     JSR GETKEY
  25        C9 0B           CMP #$0B        ; IS IT A "B"?
  27        F0 1D           BEQ BLKS        ; YES, BLANK SCREEN
  29        C9 05           CMP #$05        ; IS IT A "5"?
  2B        F0 2A           BEQ MODE        ; YES, CHANGE MODE
  2D        C9 0E           CMP #$0E        ; IS IT AN "E"?
  2F        F0 33           BEQ NORM        ; YES, NORMAL SCREEN
  31        C9 0F           CMP #$0F        ; IS IT A "F"?
  33        F0 36           BEQ INVR        ; YES, INVERTED SCREEN
  35     20 72 02           JSR MOVE        ; DETERMINE MOVE
  38        A5 E9           LDA 00E9        ; BLINK OR DRAW?
  3A        D0 05           BNE DRAW        ; DRAW
  3C     20 0F 03           JSR BLIN        ; BLINK
  3F        50 D1           BVC RETN        ; UNCOND. JUMP
  41     20 12 03  DRAW     JSR CONT        ; CONTINUOUS LINE
  44        50 CC           BVC RETN        ; UNCOND. JUMP
  46        A5 EE  BLKS     LDA 00EE        ; BLANK SCREEN
  48           48           PHA             ; SAVE VERT. AND
  49        A5 ED           LDA 00ED          HORIZ. POSITION
  4B           48           PHA
  4C     20 F8 02           JSR BLNK        ; BLANK SCREEN
  4F           68           PLA             ; RESTORE VERT.
  50        85 ED           STA 00ED          HORIZ. POSITION
  52           68           PLA
  53        85 EE           STA 00EE
  55        50 BB           BVC RETN        ; UNCOND. JUMP
  57        A5 E9  MODE     LDA 00E9        ; CHANGE DRAW-NO DRAW
  59        49 FF           EOR #$FF          MODE
  5B        85 E9           STA 00E9
  5D     20 40 1F  WAIT     JSR KEY IN      ; WAIT FOR KEY RELEASE
  60        D0 FB           BNE WAIT
  62        F0 AE           BEQ RETN
  64        A9 F1  NORM     LDA #$F1        ; SET NORMAL SCREEN
  66     20 D8 02           JSR STRO
  69        D0 A7           BNE RETN        ; UNCOND. JUMP
  6B        A9 F0  INV      LDA #$F0        ; SET INVERTED
  6D     20 D8 02           JSR STRO          SCREEN
  70        D0 A0           BNE RETN        ; UNCOND. JUMP
                ***** MOVKEY SUBROUTINE *****
0272        C9 00           CMP #$00        ; DETERMINE WHICH KEY
  74        F0 1C           BEQ DNLT          IS PRESSED AND
  76        C9 01           CMP #$01          MAKE NECESSARY
  78        F0 2C           BEQ DOWN          CHANGES TO 00ED AND
  7A        C9 02           CMP #$02          00EE
  7C        F0 19           BEQ DNRT
  7E        C9 06           CMP #$06
  80        F0 17           BEQ RITE
  82        C9 0A           CMP #$0A
  84        F0 16           BEQ RTUP
  86        C9 09           CMP #$09
  88        F0 14           BEQ STUP
  8A        C9 08           CMP #$08
  8C        F0 13           BEQ LTUP
  8E        C9 04           CMP #$04
  90        F0 02           BEQ LEFT
  92        E6 EC  DNLT     INC 00EC        ; DOWN-LEFT ENTRY
  94        C6 EB  LEFT     DEC 00EB        ; LEFT ENTRY
  96           60           RTS
  97        E6 EC  DNRT     INC 00EC        ; DOWN-RIGHT ENTRY
  99        E6 EB  RITE     INC 00EB        ; RIGHT ENTRY
  9B           60           RTS
  9C        E6 EB  RTUP     INC 00EB        ; RIGHT-UP ENTRY
  9E        C6 EC  STUP     DEC 00EC        ; STRAIGHT UP ENTRY
  A0           60           RTS
  A1        C6 EB  LTUP     DEC 00EB        ; LEFT-UP ENTRY
  A3        C6 EC           DEC 00EC
  A5           60           RTS
  A6        E6 EC  DOWN     INC 00EC        ; DOWN ENTRY
  A8           60           RTS
                ***** SPOT ON - OFF SUBROUTINE *****
02A9        A9 00  SPOF     LDA #$00        ; SPOT OFF ENTRY
  AB        F0 02           BEQ TSTH        ; UNCOND. JUMP
  AD        A9 40  SPON     LDA #$40        ; SPOT ON ENTRY
  AF           48  TSTH     PHA             ; TEST HOR. FOR OFF SCREEN
  B0        A5 ED           LDA 00ED        ; GET HORIZ. LOCATION
  B2        30 08           BMI SETR        ; IF NEG. SET TO $3F
  B4        C9 40           CMP #$40        ; IF >=40 SET TO $00
  B6        90 08           BCC RETN        ; IF NOT RETURN
  B8        A9 00           LDA #$00        ; LEFT HAND SIDE
  BA        F0 02           BEQ NEWH        ; UNCOND. JUMP
  BC        A9 3F  SETR     LDA #$3F        ; RIGHT HAND SIDE
  BE        85 ED  NEWH     STA 00ED        ; UPDATE HORIZ. POS.
  C0           68  RETN     PLA             ; 00 OR 40
  C1        05 ED           ORA 00ED        ; COMBINE WITH 00ED
  C3     20 D8 02           JSR STRO        ; STROBE IN HORIZ.
  C6        A5 EE  VERT     LDA 00EE        ; VERTICAL ENTRY
  C8        30 08           BMI SETB        ; IF NEG. SET TO 5F
  CA        C9 60           CMP #$60        ; IF >=60 SET TO 00
  CC        90 08           BCC VMSK          IF NOT, DON'T CHANGE
  CE        A9 00           LDA #$00        ; TOP OF SCREEN
  D0        F0 02           BEQ NEWV        ; UNCOND. JUMP
  D2        A9 5F  SETB     LDA #$5F        ; BOTTOM OF SCREEN
  D4        85 EE  NEWV     STA 00EE        ; UPDATE VERTICAL
  D6        09 80  VMSK     ORA #$80        ; PUT IN VERT. BIT
  D8     8D 00 17  STRO     STA PAD         ; STROBE STARTS HERE
  DB     EE 02 17           INC PBD         ; PB0 TO 0 (NOTE* PB1 TO 1)
  DE     EA EA EA           NOP'S           ; FOR DELAY
  E1     CE 02 17           DEC PBD         ; PB0 BACK HIGH
  E4           60           RTS
                ***** INITILIZATION SUBROUTINE *****
02E5        A9 FF  INIT     LDA #$FF        ; A REGISTER TO OUTPUTS
  E7     8D 01 17           STA PADD
  EA        A9 01           LDA #$01        ; PB0 TO OUTPUT
  EC     8D 03 17           STA PBDD
  EF     8D 02 17           STA PBD         ; STROBE HIGH
  F2        A9 F1           LDA #$F1        ; GRAPHICS ENABLE
  F4     20 D8 02           JSR STR0
  F7           60           RTS
                ***** BLANK SCREEN SUBROUTINE *****
02F8     20 E5 02  BLNK     JSR INIT        ; INITILIZE
  FB        A9 3F           LDA #$3F
  FD        85 ED           STA 00ED        ; START AT BOTTOM RIGHT
  FF        A9 5F           LDA #$5F          OF SCREEN
0301        85 EE           STA 00EE
  03     20 A9 02  MORE     JSR SPOF        ; TURN OFF SPOTS
  06        C6 EE           DEC 00EE        ; MORE IN VERT LINE?
  08        10 F9           BPL MORE        ; YES
  0A        C6 ED           DEC 00ED        ; MORE HORIZ. LINES?
  0C        10 F5           BPL MORE        ; YES
  0E           60           RTS
                ***** BLINK A SPOT SUBROUTINE *****
030F     20 A9 02  BLIN     JSR SPOF        ; BLINK ENTRY
  12        A5 ED  CONT     LDA 00EB        ; CONTINUOUS LINE ENTRY
  14        85 ED           STA 00ED
  16        A5 EC           LDA 00EC        ; UPDATE LOCATION
  18        85 EE           STA 00EE          00EB TO 00ED
  1A     20 AD 02           JSR SPON          00EC TO 00EE
  1D        A5 ED           LDA 00ED
  1F        85 EB           STA 00EB        ; THIS IS NECESSARY TO
  21        A5 EE           LDA 00EE          CAUSE WRAP-AROUND
  23        85 EC           STA 00EC
  25           60           RTS
                ***** DELAY SUBROUTINE *****
0326        A5 EA  DELA     LDA 00EA        ; 00EA CONTROLS TIMING
  28     8D 07 17           STA 1707        ; START TIMER
  2B     AD 07 17  HOLD     LDA 1707        ; TIME UP?
  2E        F0 FB           BEQ HOLD        ; NO
  30           60           RTS
```

```
***** GRAPHICS SUBROUTINES *****

SPOF (SPOT OFF)  THIS SUBROUTINE WILL TURN OFF THE RECTANGLE
LOCATED HORIZONTALLY BY THE VALUE IN 00ED AND VERTICALLY BY THE
VALUE IN 00EE.  IF THE VALUES GIVEN BY THESE LOCATIONS ARE SUCH
THAT THE POINT WOULD BE LOCATED OFF THE SCREEN, THEY ARE REPLACED
BY VALUES WHICH WILL CAUSE THE SPOT TO "WRAP AROUND" VERTICALLY,
HORIZONTALLY OR BOTH.  THE SUBROUTINE DOES NOT AFFECT X OR Y
REGISTERS.  THE ACCUMULATOR WILL ALWAYS BE NON-ZERO ON EXIT,
HAVING AT LEAST THE MSB SET BECAUSE OF THE VERTICAL ENTRY.

SPON (SPOT ON) IDENTICAL TO SPOF EXCEPT THE SPOT IS TURNED OFF

VERT (VERTICAL ENTRY) IN SOME CASES TIME CAN BE SAVED BY ENTERING
THE HORIZONTAL ENTRY ONCE AND THEN CHANGING ONLY THE VERTICAL
ENTRY.  AN ENTRY AT VERT WILL DO THIS.  CONDITIONS ARE AS FOR SPOF.

INIT (INITILIZE)  SETS UP OUTPUT PORTS FOR TRANSFER OF INFORMATION
TO THE GRAPHICS BOARD.  ALSO SENDS CODE TO INITILIZE THE BOARD.
AFFECTS ONLY THE ACCUMULATOR.

BLNK (BLANK SCREEN)  BLANKS THE SCREEN.  AFFECTS ONLY THE ACCUM.
00ED AND 00EE WILL CONTAIN $FF ON EXIT.

BLIN (BLINK A SPOT)  ALLOWS YOU TO PRODUCE A MOVING SPOT BY
REPLACING THE COORDINATES IN 00ED AND 00EE WITH THOSE IN 00EB
AND 00EC.  THE SPOT AT THE ORIGINAL LOCATION IS TURNED OFF AND
ONE AT THE NEW LOCATION IS TURNED ON.  USES SPOF AND SPON.  AFFECTS
ONLY THE ACCUMULATOR.

CONT ( CONTINUOUS LINE )  IF ENTERED HERE THE ORIGINAL SPOT WILL
NOT BE ERASED AND IF ADJACENT COORDINATES ARE USED, A CONTINUOUS
LINE CAN BE DRAWN.  CONDITIONS AS FOR BLIN.
```

MIT EXPERT ON PERSONAL COMPUTING

People Magazine: *Will computers be widely used by the average person in coming years?*

Michael L. Dertouzos, Director MIT Laboratory for Computer Science: We don't see technical limitations in computer development until the mid-1980's. Until then, decreased cost will make computers smaller, cheaper and more accessible. In 10 or 15 years, one should cost about the same as a big color TV. This machine could become a playmate, testing your wits at chess or checkers. If a computer were hooked up to AP or UPI newswires, it could be programmed to know that I'm interested in Greece, computers and music. Whenever it caught news items about these subjects, it would print them out on my console – so I would see only the things I wanted to see.

– *People Magazine*
August 30, 1976

HOW 'BOUT COMPUTER CHESS?

Editor: March 1, 1977

Due to the apparent lack of a medium of communication among devotees of Computer Chess, I decided to start a Computer Chess Newsletter, patterned after Hal Singer's Micro-8 Newsletter.

I hope that some suitable publisher (such as PCC) or computer science department (such as Stanford) will pick this up and do what needs to be done. It seems a shame to re-invent the basics of computer chess programming so many times. An exchange of ideas might lead to a more powerful program than independent work can.

No decisions have been made so far as to price, size, or periodicity.

All contributions are welcome, such as letters, suggestions, news, programs, games, or whatnot.

Sincerely,
Douglas L. Penrod 1445 La Cima Road

6502 CHESS SOFTWARE

Dear Jim: December 10, 1976

I am pleased to announce the availability of a new and exciting software product designed specifically for the microcomputer hobbyist.

MICROCHESS is a chess playing program written for the KIM-1, 6502 microprocessor system. It requires no additional memory or peripherals. In fact, the total memory required for program operation is only 1100 bytes.

The program can be adjusted to one of three different levels of play requiring 3, 10, or 100 seconds for each computer move. Although the level of play is probably below that of the average serious chess player, it is a good match for the average computer programmer. Even the best players will be surprised by challenging moves.

MICROCHESS was designed with the hobbyist experimenter in mind. Experimenting with your own strategy ideas is as simple as modifying a single subroutine. Further expansion of the basic program will create a sophisticated chess playing machine. The documentation supplied includes a Player's Manual with instructions for using the program, a Programmer's Manual with details of the program operation and suggestions for expansion and modification, and a complete annotated source listing.

The cost of the entire package is only $10.00. I believe that this package is of special interest to the readers of *Dr. Dobb's Journal*, and would greatly appreciate your assistance in informing them of its availability. If you have any further questions please do not hesitate to write to me at the following address.

Sincerely,
Peter R. Jennings

MICROCHESS
1612-43 Thorncliffe Pk. Dr.
Toronto, Ont., M4H 1J4
Canada

MICROCHESS

NOW YOU CAN PLAY CHESS WITH YOUR KIM–1
6502 BASED MICROPROCESSOR SYSTEM.

MICROCHESS REQUIRES NO ADDITIONAL MEMORY.
The program and data occupy only 1100 of the 1152 bytes of available RAM.

MICROCHESS REQUIRES NO ADDITIONAL PERIPHERALS.
All moves are entered and displayed via the KIM keyboard and LED display.

MICROCHESS PLAYS CHESS.
Although a good chessplayer will probably beat the program, he will be surprised again and again by challenging moves.

MICROCHESS HAS SEVERAL LEVELS OF PLAY.
You may set the program up for 3, 10, or 100 seconds per move. Change the speed at any time during the game!

MICROCHESS IS EXPANDABLE AND FULLY DOCUMENTED.
You receive a Player's Manual, complete annotated source listing, and Program Documentation describing the strategic algorithms. Instructions are provided for modification, expansion or system conversion. Experiment with your own strategies by replacing one simple subroutine.

MICROCHESS COSTS ONLY $10.00.
Send your cheque or money order today to:

MICROCHESS, 1612-43 Thorncliffe Pk. Dr.,
Toronto, Ontario, M4H 1J4, Canada.

The characteristics and decodability of the Universal Product Code symbol

by D. Savir and G. J. Laurer

Described are the coding and symbol of the Universal Product Code. The symbol code structure, format, encodation technique, and characteristics with their technical tradeoffs are discussed.

The symbol is analyzed and evaluated. Decodability is shown to depend on the structure of the code and symbol, the size of the symbol, the precision with which the symbol is printed, the technique of scanning employed, the accuracy with which measurements are made, the decoding logic, and the physical operation of scanning. The relationship between the scan pattern of a fixed head scanner and symbol size is shown.

The scanning of information from the labels of supermarket merchandise is necessary for a practical supermarket system. This information is encoded in a standard form, the Universal Product Code (UPC). The standard comprises both the code—the representation of decimal characters in binary form—and the symbol—the printed form of the code which can be read by a scanner.

In this paper, we discuss the development of this standard; we define a class of codes suited to optical scanning and investigate some of its properties, and we describe the code belonging to this class that was selected for the UPC.

The code, symbol, appropriate decoding scheme, and scanning scheme are all dependent upon each other. They constitute the structure of the decoding process that is studied to evaluate the decodability, or decoding reliability, of the UPC. (The UPC is standard in the United States and Canada. At the time of writing this paper, proposals for merchandise-identifying symbols for other countries are being advanced.)

Development of the Universal Product Code

The justification for a supermarket system lies both in precise point-of-sale data capture and in the increase of checkout productivity compared to that of a conventional checkstand.[1] If a code were devised that could both identify each item sold in a store uniquely and enable the item to be checked rapidly, then its use could justify the development of such a supermarket system.

Item identification could be achieved using a number containing sufficient characters; the state of the art dictates that rapid checkout and identification number entry can be achieved only by using a fixed scanner[2] that reads a form of bar code rather than numeric information. Consequently, appropriate item identification should be by a combined code and symbol in which the symbol is the bar-coded representation of the numeric code. Clearly, the code and symbol (which we will henceforth call merely code or symbol, as appropriate) is strongly dependent on a specific supermarket system—the numeric code is filed in the system and constitutes a data base for store decision making and price look-up;[1] the bar representation is attuned to the scanner decoding methodology. It should, therefore, follow that a supermarket system should also include a code that would be placed on each item in the store, replacing the price mark. However, it became clear that the cost to a store of marking its items with scannable code would be much higher than the cost of price-marking, to the extent of negating much of the benefit of the system. The economically acceptable approach, recognizing the *permanence* of the code (the code number is as much part of item identification as its name; price is not part of it), was to have the code printed on the item label by the grocery manufacturer (source-marking). (Note: Subsequently, variants of the code containing price were proposed. These were intended for items of variable weight (meat, produce, cheese) and to be automatically printed by the weighing device.)

The obvious objection is that since the code depends on its supermarket system, provided by a vendor, then one of these events would be necessary for a source-marked code:

a. Grocery manufacturers would mark the symbol of every vendor on every item.
b. Grocery manufacturers would print different sets of labels for each item, each set containing the distinctive vendor symbol; a store using a particular vendor's equipment would receive appropriately labeled merchandise.
c. A standard code would be designed for the community of vendors, supermarkets, and manufacturers; this code would be printed on the labels.

Event a was unacceptable because (1) few labels would be large enough to contain all symbols, (2) as new vendors offered their products the printers would have to replace their plates, (3) checkout productivity would be impaired substantially, and (4) printing costs would be exorbitant.

Event b was unacceptable because (1) grocery manufacturers would be reluctant to print labels for systems that might not be installed, while supermarkets would be reluctant to install systems without assurance of adequate, appropriately marked labels and (2) printing costs and label inventories would be exorbitant.

Thus, c, the least unattractive choice, was taken—industry groups with conflicting objectives accepted a voluntary standard for the code. Most of the conflicts were about symbol size and print quality. Grocery manufacturers wanted symbols to occupy little label space since the symbol would detract from product identification, whereas larger symbols are more easily scanned. The sloppier the symbol can be printed, the easier it is to print and the harder it is to scan. The UPC code was selected by a committee, Uniform Grocery Product Code Council, Inc., which was composed of representatives of grocery manufacturers and supermarket chains. This committee delegated to its Symbol Standardization Subcommittee the task of soliciting, reviewing, and amending suggestions from vendors, and finally proposing the UPC code to the Council. During its task, the subcommittee engaged a consultant, McKinsey and Co., Inc., and prepared a set of guidelines to enable vendors to offer suggestions for a code that would satisfy the guidelines, offer something more than the guidelines required, and be compatible with scanning equipment that they could be expected to subsequently provide.

Some of the guidelines were:

- The code should contain 10 decimal characters (subsequently, 12 were required).
- The symbol should be scannable omnidirectionally, i.e., regardless of its orientation with respect to a scanning device.
- The symbol should be scannable when in motion at a veloc-

Reprinted by permission from *IBM Systems Journal*., © 1975 by International Business Machines Corporation.

ity not exceeding 100 inches per second.
- The scanning reject rate should not exceed 0.01 and the undetected error rate should not exceed 0.0001.
- The depth of field should be at least one inch.
- Normal environmental contamination (abrasion, dirt, etc.) should not affect the scanning process significantly.
- The symbol area should not exceed 1.5 square inches. (The symbol selected is of variable size.)

The symbol selected by the Uniform Grocery Product Code Council, in addition to meeting the guidelines, subject to the parenthetical comments, also contains parity and a modulus check; it can also be adapted to encode more (and fewer) characters.

Although the principal intent of the UPC code was to provide a data medium for fixed scanners, it was recognized that the coded information should be able to provide input to checkers using hand-held scanners (optical wands) and to checkers using keyboards for decimal code entry. The code was, therefore, designed to be readable by humans, by fixed scanners, and by wands.

Optical codes

An economically feasible means of printing a symbol is to print dark marks on a light background on a label. (The converse is also true; conceptually both processes yield similar results.) The code is of binary nature—the presence of a mark corresponds to impressed information, represented by a one bit, in a domain of the label (called a *module*), and the absence of a mark corresponds to lack of impressed information, represented by a zero bit in a module. Encoding of a symbol can be done by conventional binary means, i.e., by representing the characters by their binary code, partitioning the space on the label into modules and printing marks on the modules covered by one bits (Figure 6).

Decoding a symbol is more complex because the modules that define the domain of the elements of the code (one and zero bits) are not perceivable by the spot of light that scans the marks.[2] The elements of the code with respect to decoding are the marks (dark bars) and the spaces (light bars) between them, represented by runs of one and zero bits, respectively. Therefore, an integral part of the decoding process is the determination of the length of each run. This determination is confounded by imprecision that can lead to error.

Errors in binary decoding of line signals are due to noise, loss of information, or discrimination error, given that the decoding device is attuned to the signal. The standard error-detecting and error-correcting devices presume that any bit is as likely to be decoded erroneously as any other bit; the probability of erroneous message interpretation is evaluated on this presumption.

The nature of optical binary codes, however, is different. Firstly, the decoding device is not attuned to the signal—the speed at which signals from evenly spaced marks enter the decoding device varies due to variation in scanning-spot velocity (especially when the scanner is hand held), curvature of the surface upon which the marked label is placed, variation in depth of field over the label, etc. Secondly, the techniques of error detection must work both with the binary coded input and with the decimal input—the code should contain protection against miskeying on a terminal keyboard. Thirdly, bits are not equiprobably subject to misinterpretation. If the symbol is printed with adequate contrast (which is not difficult), random noise or loss of information will not be significant. The principal source of error is at the edges of marks.

reasons for error

In the scanning process, a spot interprets the location of each edge of each mark and decodes information accordingly. A decoding error occurs if and only if the location of one or more edges is interpreted incorrectly. One combination of code and symbol has a higher decodability than another if the former can tolerate a stochastically greater edge dislocation than the latter while still decoding the information correctly.

The edge of a mark is designed to be at a specific location. However, as the artwork is drawn, for example, an error is introduced. Further errors are introduced in the processes of reduction and platemaking. Additional errors occur in the process of printing. The symbol as seen by the eye contains at each edge of every mark the sum of all these errors. The scanner cannot perceive the edge to be exactly where it is printed; there are errors introduced due to optical effects and to the effects of digital timing and discrete sampling. In addition, there are errors caused by the environmental degradation of the symbol of which dirt, wrinkles, abrasion, and moisture are a few. The total dislocation of the edge of the mark as perceived by the scanner is the sum of all these errors. If this sum of errors on any edge exceeds some value, a decoding error will be made. An appropriate choice of code and decoding scheme will be shown to neutralize certain components of systematic error.

We partition the sources of edge dislocation into print error, $\bar{e}_p$, and system error, $\bar{e}_s$, such that the total dislocation on an edge is $\bar{e} = \bar{e}_p + \bar{e}_s$. Errors in artwork, platemaking, and printing are consolidated into $\bar{e}_p$; errors due to optical effects and the effects of digital timing and discrete sampling and those due to environmental degradation are consolidated into $\bar{e}_s$.

print errors

The errors contained in $\bar{e}_p$ that affect the location of the edge of the mark are due to (1) artwork, (2) platemaking, (3) edge roughness, (4) extraneous ink, (5) voids, (6) smear, (7) inkspread, and (8) expansion and contraction of substrate.

Errors in artwork consist of the *random* error in the line drawing and a *systematic* error in photoreduction. A random error affects an edge or a portion of an edge independently of the rest of the symbol. A systematic error affects all the edges in a similar manner.

Errors in platemaking contain a systematic component, increasing or decreasing the size of all the marks, and a random component. Edge roughness is a random effect whose intensity depends on the printing process and the paper. Extraneous ink and voids affect edges only when sufficiently large to be identified falsely as a mark or when intrusive into the edge of a mark.

The error of smear is due to a systematic ink deposit in the direction of motion of the paper. The effect of smear on marks of differing shape is shown in Figure 1. Arrows indicate optimal directions of traversal of the scanner spot with respect to the marks, ignoring the effect of smear. The presence of smear on each of the marks affects the optimal trajectory by differing amounts. We observe that mark D of Figure 1 succeeds in neutralizing the smear. The spot does not traverse any edge affected by smear over the effective range of the mark.

The error of inkspread is due to over-inking (or conversely un-

der-inking) which increases (or decreases) the size of all the marks, systematically. The effect of inkspread on sequences of marks of differing width is shown in Figure 2. If the spot of the scanner follows a straight trajectory across the marks, the effect of inkspread is to add a constant increment to the width of each mark.

Figure 1 Effect of smear on marks

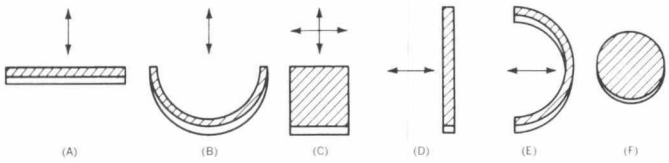
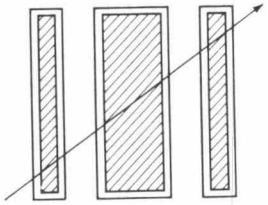


Figure 2 Effect of ink-spread on mark sequences



Other systematic effects result in a change in scale of one dimension with respect to the orthogonal dimension, i.e., an apparent expansion or contraction of length or width of the symbol.

neutralizing systematic errors

The effects of systematic errors are isolated and controlled by using a rectilinear bar-coded symbol and an appropriate decoding scheme. The symbol is a sequence of long rectangular marks of several widths separated by spaces of several widths (Figure 5). By printing the symbol such that the bars are aligned with the motion of the paper through the printing press, the adverse effects of smear are controlled. The other systematic effects are controlled by the decoding technique.[3]

decoding technique

In this decoding technique, the ratio of the measurement across a mark and an adjacent space to a reference measurement within the character is used. Practically, the measure cannot be one of length; the time to effect the transitions is actually measured. Since the decoding device is not attuned to the signal, the reference measurement is necessary, and since the scanning spot does not necessarily cover the marks at a uniform velocity, the reference measurement should be proximate to the decoding measurement.

We illustrate the power of this decoding technique by examining the phenomenon of inkspread, separating its error effect from the other errors that contribute to edge dislocation. Schematically, the section across a sequence of marks and spaces through which the scanning spot passes is as shown in Figure 3 where a_0 through a_5 represent the locations of the edges of the marks, with a_0, a_2, and a_4 denoting space-to-mark transitions and a_1, a_3, and a_5 denoting transitions from mark to space. We assume that each of the edges is dislocated due to various errors, but that dislocation due to inkspread is not included. Suppose, in addition, that we have the means of decoding the symbol based on data a_i.

Now let us apply an error, ξ, due to inkspread, where ξ is unknown (in fact, will vary from print run to print run) but is consistent throughout any single print run. After the error ξ is applied, the sectioned symbol is like Figure 4 where b_i represent the locations of the edges of the bars corresponding to locations denoted by a_i under the transformation: $b_i = a_i - (-1)^i\xi$. The even-indexed locations are shifted to the left, and the odd-indexed locations are shifted to the right, yielding an increase in the width of each mark of 2ξ, independent of the original width of the mark. The locations b_i are the only edges perceivable to the scanner or to the eye.

Figure 3 Section through symbol, ignoring effects of systematic errors

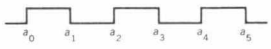


Figure 4 Section through symbol, accounting for effects of systematic errors

Now for $i \geq 2$:

$$b_i - b_{i-2} = a_i - (-1)^i\xi - a_{i-2} + (-1)^{i-2}\xi$$
$$= a_i - a_{i-2}$$

which yields sufficient information for the decoding of the symbol. Hence the process of decoding by measuring the distances between the leading edges of adjacent marks and those between the trailing edges of adjacent marks isolates and controls the effect of the error, ξ, due to inkspread. Each such distance spans a mark and an adjacent space. By the same argument, any systematic error that widens or narrows each bar by the same amount is circumvented by this decoding technique.

A class of suitable codes

Desirable properties of a suitable optical code are

1. Each character representation should be of fixed length and stand alone, independent of adjacent characters. This feature enables label makers to print the code simply and facilitates artwork preparation of source-marked labels.
2. Each character representation should contain a fixed reference measurement from the leading (trailing) edge of a mark to the leading (trailing) edge of another known mark. The code should be decodable forwards, in which the reference measurement lies between the leading edges, or backwards, in which the reference measurement lies between trailing edges.

A class of suitable codes is the class of (n, k) *codes*. An (n, k) code represents each character uniquely by n bits containing k runs of one bits and k runs of zero bits, $1 \leq k \leq n/2$. The first bit of a character is a one bit. In the terminology of marks and spaces, we can define an (n, k) code as one in which the representation of each character is by a unique set of k marks and k spaces spanning exactly n modules and beginning with a mark.

The definition is related to forward scanning and decoding. The reference length is n, measured between leading edges k marks apart. An (n, k) code is decodable backwards, in which case the reference measurement is found between the trailing edges of marks.

An (n, k) code represents $\binom{n-1}{2k-1}$ distinct characters. This follows from the fact that there are $\binom{n-1}{r-1}$ distinguishable ways of

placing n indistinguishable items into r buckets such that no bucket remains empty.[4]

Table 1 shows the number of distinct characters representable in (n, k) codes for $2 \leq n \leq 10$, $1 \leq k \leq 4$. We note that, for the least necessary number of bits, a (6, 2) code will represent the set of decimal characters adequately, a (7, 2) code will represent the set of decimal characters with additional recognition characters, and a (9, 2) or (9, 3) code will represent a full alphanumeric character set.

Table 1 Distinct characters in (n, k) codes

	k→ 1	2	3	4
n 2	1	0	0	0
↓ 3	2	0	0	0
4	3	1	0	0
5	4	4	0	0
6	5	10	1	0
7	6	20	6	0
8	7	35	21	1
9	8	56	56	8
10	9	84	126	36

If parity (odd or even) is desired, a (7, 2) code will represent the set of decimal characters, an (8, 2) code will represent the set of decimal characters with additional recognition characters, and a (10, 3) code will represent a full alphanumeric character set. In summary, if the code contains six bits, we have decimal representation, if seven bits, we also have either additional characters or parity, if eight bits, we have both additional characters and parity, if nine bits, we have alphanumeric representation without parity, and if 10 bits, we also have parity.

Given an (n, k) code, we can identify each character represented by the code by an integer $2k$ − tuple $(c_1, c_2, \cdots, c_{2k})$ $c_i \geq 1$, $\Sigma_{i=1}^{i=2k} c_i = n$. c_i is the length of each run of one or zero bits, or, equivalently, the number of modules contained in each mark and space. We have noted that in the scanning and decoding of the symbol, the determination of the values of c_i may be difficult due to the expansion or contraction of the printed marks. This difficulty is largely overcome by the decoding technique in which the distance encompassed by a bar and adjacent space is measured.

Let

$$t_i = c_i + c_{i+1}, \quad i = 1, 2, \cdots, 2k-1$$

$$t_{2k} = c_{2k}$$

t is in one-to-one correspondence with c; in fact

$$c_i = \sum_{j=i}^{2k} (-1)^{i+j} t_j$$

We note that t_i is the number of bits covered by two adjacent runs, except for t_{2k} which is the length of the last run of zero bits in a character. The decoding of a character by its t representation rather than its c representation will be less susceptible to the error of mark expansion or contraction. We cannot, however, dispense with t_{2k} because $(t_1, \cdots, t_{2k-1})$ may correspond to more than one value of $(c_1, \cdots, c_{2k})$, hence will not decode uniquely. t_{2k-1} is redundant, since

$$\sum_{i=1}^{k} t_{2i-1} = n$$

The set of $\{t_1, \cdots, t_{2k-2}\}$ is connected at integer points, i.e., given a value of $(t_1, \cdots, t_{2k-2})$ it is always possible to change at least one of the t_i by one integer (up or down) to yield another value of $(t_1, \cdots, t_{2k-2})$ which, of course, corresponds to a different character or characters. Parity on $(t_1, \cdots, t_{2k-2})$ is a protective device; it would be useful if parity on the values of t were equivalent to parity on the one bits of the character, i.e.,

$$\sum_{i=1}^{k} c_{2i-1}$$

$$\left(\sum_{i=1}^{k} c_{2i-1}\right) \bmod k = \left(\sum_{i=1}^{k} \sum_{j=2i-1}^{2k} (-1)^{2i+j-1} t_j\right) \bmod k$$

$$= \left(\sum_{i=1}^{k-1} i(t_{2i-1} - t_{2i})\right) \bmod k$$

Figure 5 UPC symbol

For an $(n, 2)$ code we have

$$(c_1 + c_3) \bmod 2 = (t_1 - t_2) \bmod 2$$

$$= (t_1 + t_2) \bmod 2$$

For an $(n, 3)$ code we have

$$(c_1 + c_3 + c_5) \bmod 3 = ((t_1 - t_2) + 2(t_3 - t_4)) \bmod 3$$

$$= (t_1 - t_2 - t_3 + t_4) \bmod 3$$

For $k > 3$ we cannot express the parity of

$$\left(\sum_{i=1}^{k} c_{2i-1}\right) \bmod k$$

as a mod k linear function of t with unit coefficients.

Therefore, both $(n, 2)$ codes and $(n, 3)$ codes can use parity that is equivalent for both the sum of the one bits and sums (or differences) of the values of $(t_1, \cdots, t_{2k-2})$; for the $(n, 2)$ code the parity is the "traditional" binary parity; for the $(n, 3)$ code a ternary parity must be used. (Note that $(t_1 \pm t_2 \pm t_3 \pm t_4) \bmod 2 = (c_1 + c_5) \bmod 2$; in general, $\Sigma_{i=1}^{2k-2} \pm t_i \bmod 2 = (c_1 + c_{2k-1}) \bmod 2$.)

The decoding process will use the measurements of the variables t_i normalized by the measurement of the reference n. In the next section, we describe the UPC code selected, after which we discuss the details of its decoding.

Figure 6 Binary and bar-coded representation of even-parity characters

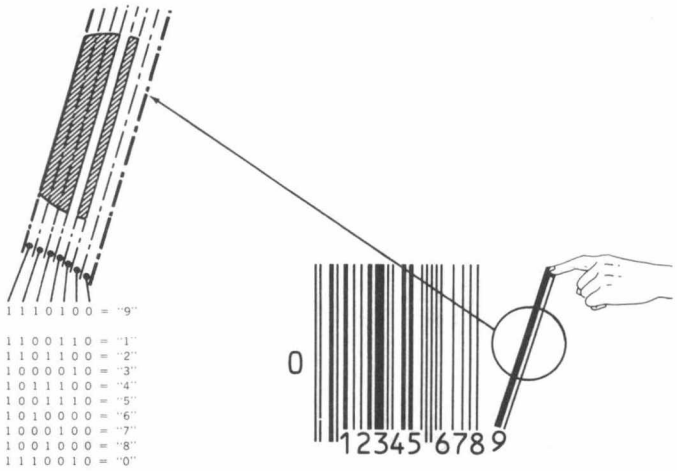


Figure 7 Decomposition into two symbol blocks

Description of UPC code and symbol

Of the (n, k) codes, $(n, 2)$ codes contain parity equivalence between the parity of one bits in the character and the parity of the bits contained in the modules whose measurements are necessary for decoding by the technique previously discussed. Since the (7, 2) code contains sufficient characters for decimal representation with parity, this was the code selected for the UPC symbol[5] in a rectilinear bar-code format.

Each character is represented by two dark and two light bars, of an integral number of modules each, spanning a total of seven modules. The character begins on a dark bar. The symbol of Figure 5 is broken out in Figure 6 to show the representation of a character and the binary representation of the bars in which a dark (light) bar occupying i modules is represented by a run of one bits (zero bits) of length i.

The symbol contains 12 characters (including a modulus check character), broken into two symbol blocks of six characters each which are scanned independently. If the symbol were unbroken, a symbol of comparable decodability and module size would require significantly longer bars than the symbol selected. This point will be discussed with the omnidirectionality of scanning. Each symbol block is delineated by two dark guard bars, separated by light bars, of one module each. The center pattern of guard bars is shared by both symbol blocks (Figure 7). The left block is distinguishable from the right block by the parity of the included characters—odd parity characters fill the left block, even parity characters fill the right block. The beginning of a character (dark bar) is closest to the center pattern. The binary representation of the complete character set is shown in Figure 8. Note that the odd parity characters complement the even parity characters, and characters are unambiguous when reflected, permitting backwards scanning.

Figure 8 Binary representation of character set

ODD PARITY ON THE LEFT
EVEN PARITY ON THE RIGHT

	LEFT CHARACTERS	RIGHT CHARACTERS
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

With a rectilinear bar code, omnidirectional scanning is achievable by an X-pattern, comprising one or more Xs (Figure 9). Under omnidirectionality, with a simple X-pattern (one in which all Xs are identical), when the legs of the X subtend an angle (α) of 45° to the normal to the mean direction of item movement, the length of the bars is minimized,[6] at a value $b = a + ut$, where a is the width of the symbol block, u is the item velocity, and t is the period of the scanning cycle[2] (Figure 10).

Figure 9 Omnidirectional scanning using X-pattern

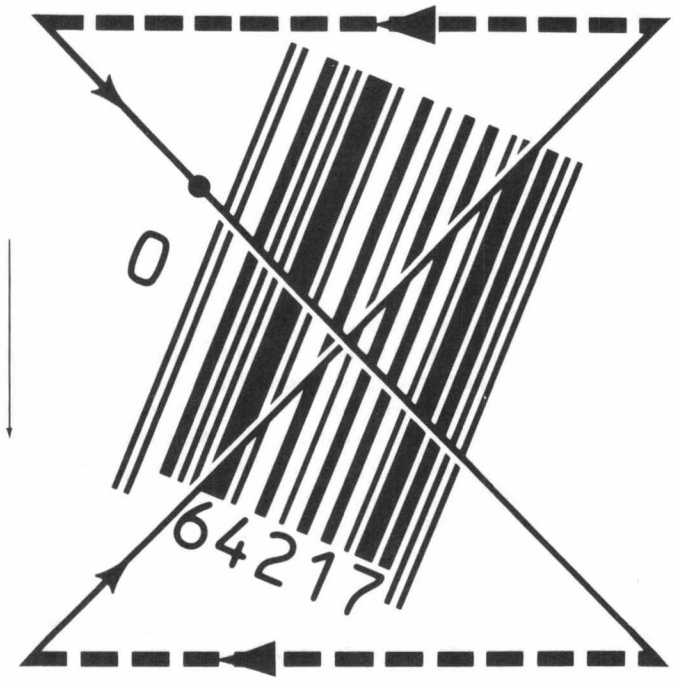
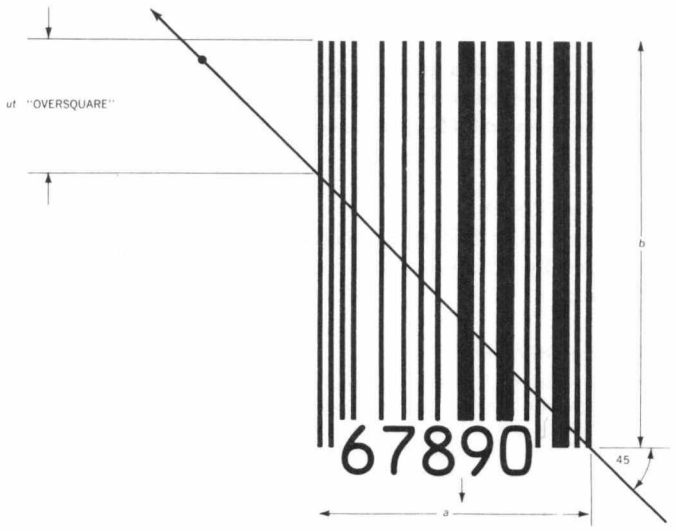


Figure 10 Dimensional parameters for omnidirectional scanning



The value of ut to which the scanner is designed is called the oversquare, for obvious reasons. It should now be clear why the symbol is designed in two blocks; if only one were used, the necessary symbol area would be close to $2a(2a + ut)$ which is significantly larger than the necessary $2a(a + ut)$ of the UPC symbol. It might be inferred that four blocks are superior to two; however, each block requires specific identification within the symbol. The UPC blocks are identified by their parity. If four blocks were used, so many additional characters would be required for block separation and identification as to negate the advantage of the reduced bar length.

The power of parity is slightly weakened, therefore, in the UPC code; parity is used both for block identification and for character validation. Characters are assumed to be of the correct parity if all six characters in a block possess the same parity. The probability of decoding error due to this weakening is vanishingly small. We shall show subsequently that there is a variant of the UPC symbol (Version E) with even weaker parity structure—this symbol contains only one block with three odd and three even parity characters. The error detection and correction methodology is discussed in detail in References 6 and 7; however, in brief, the following interpretations are made:

- If six odd(even) parity characters are scanned, a left (right) block is assumed, to be confirmed if an assumed right (left) block is also found.
- If five odd(even) parity characters are scanned, a left (right) block containing an invalid character is assumed, to be confirmed if an assumed right (left) block is also found.
- If three odd and three even parity characters are scanned, a Version E block is assumed, to be confirmed if no assumed right or left block is found.
- If four odd or even parity characters are scanned, the information is useless and ignored.

The various versions of the UPC symbol are described in the Appendix.

Decoding the UPC symbol

From Figure 8 we can represent the characters in both c and t notation of (n, k) codes as shown in Table 2.

We have previously noted that (t_1, t_2, t_3) are related to measurements of the decoding technique, and also that t_3 is redundant since $t_1 + t_3 = 7$ in all cases. Representing each character by (t_1, t_2) we obtain the matrix of Figure 11.

The decoding process is to first ascertain values of t_1 and t_2 from measurements of the decoding technique, and then to discriminate, where necessary, between one and seven and between two and eight. Measurements T_1, T_2 and T (the reference measurement)[6] are indicated in Figure 12.

Figure 11 Decimal characters as functions of t_1 and t_2

t_1 \ t_2	2	3	4	5
2	E 6	0 0	E 4	0 3
3	0 9	E 2 E 8	0 1 0 7	E 5
4	E 9	0 2 0 8	E 1 E 7	0 5
5	0 6	E 0	0 4	E 3

NOTE: ODD (O) AND EVEN (E) PARITY

Figure 12 The T measurements of an even-parity nine

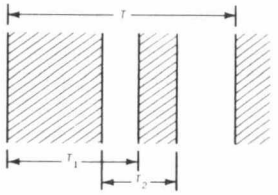


Table 2 Decoding tables

Character	*c-notation*		*t-notation*	
	odd	*even*	*odd*	*even*
0	1123	3211	2353	5321
1	1222	2221	3442	4431
2	2212	2122	4332	3342
3	1141	1411	2551	5521
4	2311	1132	5421	2452
5	1321	1231	4531	3541
6	4111	1114	5221	2254
7	2131	1312	3441	4432
8	3121	1213	4331	3343
9	2113	3112	3243	4232

We establish the following decision rules with respect to the interpretation of the value of t_i:

$$\frac{T_i}{T} \leq \frac{2\frac{1}{2}}{7} \Rightarrow At_i = 2$$

$$\frac{2\frac{1}{2}}{7} < \frac{T_i}{T} \leq \frac{3\frac{1}{2}}{7} \Rightarrow At_i = 3$$

$$\frac{3\frac{1}{2}}{7} < \frac{T_i}{T} \leq \frac{4\frac{1}{2}}{7} \Rightarrow At_i = 4$$

$$\frac{4\frac{1}{2}}{7} < \frac{T_i}{T} \qquad \Rightarrow At_i = 5$$

where At_i is the value assigned to t_i as a consequence of the decision. The decision is correct when $At_i = t_i$. We note (Figure 11) that an error of one unit in the interpretation of either t_i or t_2 but not both will cause a parity error in the decoded character.

There are two approaches to the resolution between one and seven and between two and eight. One approach, discussed in detail in References 6 and 7, is to extend t_4 to cover the first mark of the subsequent character (or guard bar if no character follows), and to measure an associated T_4. Then, having decoded the subsequent character, its value c_1 is known, which, when subtracted from At_4 will yield the unextended t_4. Another approach is to measure the mark corresponding to c_1 (in the case of the one-seven discrimination) or to c_3 (in the case of the two-eight discrimination) directly, correcting for systematic errors,

deciding whether the measured value corresponds to one or two modules and decoding accordingly.

This decoding process is suited to decoding in real time. Since characters can be decoded both forwards and backwards, if a T_4 measurement is required for a character and if the symbol block is scanned backwards, then the resolution of the character is immediate. Resolution must be delayed if the block is scanned forwards.

Symbol decodability

A probabilistic model of decoding reliability or decodability is presented in Reference 6. In this section, we will qualitatively discuss factors that contribute to the symbol decodability.

When an item is passed over the scanning window, one of the following three consequences will occur, in order of decreasing desirability:

1. The symbol will be correctly decoded and transmitted to the controller.
2. No code will be transmitted, the scan will be rejected, and the system will request another item pass.
3. An incorrect code will be transmitted to the controller.

Symbol decodability is the probability of each of these outcomes. It depends both on how successfully the symbol is decoded and on how well the system recognizes decoding error if it exists. The probability of decoding error depends on the likelihood of making the correct decoding interpretations discussed in the previous section. Three devices are contained in the code and symbol to assist in recognition and possible correction of decoding errors should they occur. They are redundancy, parity, and a modulus check.

As the item is passed over the scanning window, each symbol block will be scanned a variable number of times, depending upon the length of the bars of the symbol, the item velocity, and its orientation with respect to the window. Each scan will collect information from a different "slice" of the label so that it is unlikely for a specific error due to a random print imperfection to occur on more than one of the scans. If an error occurs as a consequence of the incorrect interpretation of a T_1 or a T_2 measurement, but not both, then the parity of the decoded character will be reversed (Figure 11).

A correctly decoded scan will satisfy the modulus check. If all characters except one are correctly decoded and the incorrect character is also of incorrect parity, then the code can be corrected by using the pointer of parity violation in conjunction with the value of the syndrome or checking number.

Simultaneous use of redundancy, parity, and modulus check yields an adequate level of decodability, given sufficient decoding accuracy, which in turn depends on print quality and on the precision of the scanning device.

Summary

Optical scanning components of supermarket systems require machine-readable code. The need for industry-wide cooperation and standardization of a code and symbol was shown. The sources of decoding error in printed optical codes were discussed and a class of codes, the (n, k) codes, relatively resistant to error, were presented. The UPC code is a (7, 2) code.

The UPC code and a method of decoding it were described. Decodability depends on several parameters; their quantitative relationships and effects upon decodability are found in another paper.[6]

ACKNOWLEDGMENT

The material from the UPC Symbol Specification in the Appendix is reprinted by permission of the copyright owner, Distribution Codes, Inc.

CITED REFERENCES

1. P. V. McEnroe, H. T. Huth, E. A. Moore, and W. W. Morris, III, "Overview of the Supermarket System and the Retail Store System," in this issue.
2. L. D. Dickson and R. L. Soderstrom, *The IBM Supermarket Scanner*, Technical Report (in preparation), IBM Corporation, System Development Division, Rochester, Minnesota.
3. P. V. McEnroe and J. E. Jones, *Identification Technology for the Retail Industry*, IBM Corporation, System Development Division, Research Triangle Park, North Carolina (October 1971).
4. W. Feller, *An Introduction to Probability Theory and its Applications*, Vol. 1, 2nd Ed., 37, John Wiley & Sons, Inc., New York, New York (1957).
5. *UPC Symbol Specification*, © Distribution Codes, Inc. (formerly Distribution Number Bank, Inc.), 1725 K Street, N. W., Washington, D. C. (1973).
6. D. Savir, *A Model of the Decodability of the Universal Product Code Symbol*, Technical Report TR29.0123, IBM Corporation, System Development Division, Research Triangle Park, North Carolina.
7. D. Savir, *The Effect of the Design of the IBM Proposed UPC Symbol and Code on Scanner Decoding Reliability*, IBM Corporation, System Development Division, Research Triangle Park, North Carolina (October 1972).

Appendix

There are five versions of the UPC symbol that are reserved for specific uses[5] as listed in Table 3. The format for the five versions varies as shown in Table 4. Of specific interest are Versions A and E. The other versions are modifications of Version A.

Except for Version E, the number system character identifies both the version and the type of item that is described; for example, both a grocery item whose information characters identify the item, and a weighed item, such as meat or produce, whose information characters contain a variable price, would be encoded in the same version but distinguished by the value of S.

The zero-suppression version (E) is similar to the portion of Version A (regular symbol) to the left of the center except for the following:

1. It has a right guard pattern which is coded 010101.
2. Three of the characters are coded in odd parity and three in even, as in Figure 4, except that the even parity characters are reflected.
3. Only two number systems are available: "0," used for regular UPC numbers; and "1", which is currently unassigned.
4. The coding of the zero-supression version is compressed into six characters of varying parity. The determination of whether a character's parity is odd or even is given in Table 5.

There is, therefore, no explicit character encodation of the number system or modulo check characters; their values are derived from the parity permutation of the six encoded characters.

Table 3 Five versions of the UPC symbol

Version	*Intended Use and Number of Information Characters*
A	*Basic version*, used to encode the 10-character Grocery Industry UPC as well as the present National Drug Code (NDC) and National Health Related Items Code (NHRIC).
B	*Special version*, reserved for encodation of the National Drug Code and National Health Related Items Code if expansion to 11 characters is required at a later date.
C	*Special version*, with 12 characters, reserved to promote industry-wide product code compatibility, expanding the existing family of compatible codes (Grocery UPC, NDC, NHRIC, Canadian Grocery Code, and the Distribution Code).
D	*The 12 + n character or variable-message length version* was adopted to provide a compatible version of the symbol for possible use in grocery stores that sell general merchandise or in general merchandise or department stores where more information may be needed in the symbol. Note that effective use of this version for source symbol marking implies agreement by general merchandise distributors and their suppliers on a common code numbering system.
E	*The zero-suppression version* of the symbol is included to facilitate source symbol marking on packages that would otherwise be too small to include a symbol. This is achieved by encoding the symbol in six characters in a special way that leaves out some zeros that can occur in the UPC code. For example, code 12300-00045 can be encoded in a symbol as 123453, effectively eliminating half of the area that would otherwise be required for the symbol.

Table 4 Format for UPC symbols

Version	*Title*	*Format*
A	Regular	SXXXXX XXXXXC
B	Drug B	SXXXXX XXXXXX
C	12-character	XSXXXXX XXXXXCX
D	12 + n-character	SXXXXX XXXXXCXX . . .
E	Zero suppression	XXXXXX

X = information character
S = number system character
C = modulo-10 check character

Table 5 Parity pattern of the zero-suppressed symbol

Number system	*Modulo check character value*	*Character location number* 1	2	3	4	5	6
0	0	E	E	E	O	O	O
0	1	E	E	O	E	O	O
0	2	E	E	O	O	E	O
0	3	E	E	O	O	O	E
0	4	E	O	E	E	O	O
0	5	E	O	O	E	E	O
0	6	E	O	O	O	E	E
0	7	E	O	E	O	E	O
0	8	E	O	E	O	O	E
0	9	E	O	O	E	O	E
1	0	O	O	O	E	E	E
1	1	O	O	E	O	E	E
1	2	O	O	E	E	O	E
1	3	O	O	E	E	E	O
1	4	O	E	O	O	E	E
1	5	O	E	E	O	O	E
1	6	O	E	E	E	O	O
1	7	O	E	O	E	O	E
1	8	O	E	O	E	E	O
1	9	O	E	E	O	E	O

S-100 BUS START-UP KIT FOR $79.95

If you have decided that the best way for you to get started in hobby computing is to build it yourself, then this project is for you.

After you have decided on the 8080 microprocessor, and looking through various computer magazines you can find many bargains in areas of memory cassette I/O video display, etc. Power supply and mother board usually don't pose much of a problem. Well, after you have all these cards built and plugged into your S-100 Bus, what next?

With the construction of this card you can have your operating system and BASIC loaded in 80 seconds from turn on! There is a direct Keyboard input port plus 1 additional in port and 2 output ports. It uses 2 easily programmed 32 x 8, 8223 PROMS. There is plenty of additional space for 2K, 2102 RAM or possibly VART for serial users. It is constructed using wire wrap techniques on a S-100 prototype card. Minimum RAM required is 2K at the top of memory and 8K at the beginning. The data on the PROMs and the software on Cassette provided is based on the use of the very popular Tarbell cassette interface and the Processer Technology VDM-1. Our lab computer is running with MITS CPU or Byte Shop CPU and Solid State Music, Vector Graphic, and Processer Technology Memory. We also use the Pennys 6536 tape recorder.

Parts if purchased separately cost over $100.00. Kit price is $79.95 post paid. Included Proto Card and all parts and complete documentation card separately $25.00 post paid. Data package $7.50 post paid. Available off the shelf from Sargent's Distributing, 4209 Knoxville, Lakewood, Ca. 90713. Bank America-Master Charge orders accepted.

SIGNETICS OFFERS SECOND-SOURCING OF INTEL AND TI 4K DYNAMIC RAMS

SUNNYVALE, Calif. – A 4096-bit Read/Write Dynamic Random Access Memory has been introduced by Signetics.

The 4K x 1 device, designated 2680, is fabricated using N-Channel silicon gate MOS technology. The unit is identical in specifications and in its 22-pin configuration to the Intel 2107B and the Texas Instruments 4060 RAMs.

Signetics designed the 2680 with minimum production costs as a prime criterion, using a single transistor cell to minimize the device area, according to Ralph Kaplan, Marketing Manager for Memories.

The single device cell, along with design features in the on-chip peripheral circuits, yields a high performance and low cost memory device for a variety of applications, including those requiring high speed, low cost, and/or large bit capacity.

The Signetics 2680 is priced competitively with the Intel and TI devices, Kaplan says. Delivery is two to four weeks in quantities up to 5,000; six to eight weeks for larger quantities.

For further information contact Signetics, 811 East Arques Avenue, Sunnyvale, California 94086. Telephone: (408) 739-7700.

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 15 **May 1977** **Volume 2, Issue 5**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

(BE IT EVER SO LATE)

What to Get Your Computer for its First Birthday, *by Jef Raskin* 5

Testing the DZ80-80 CPU: Nifty Z-80 Conversion for Altair & IMSAI Computers, *by Jef Raskin* 7

SOFTWARE

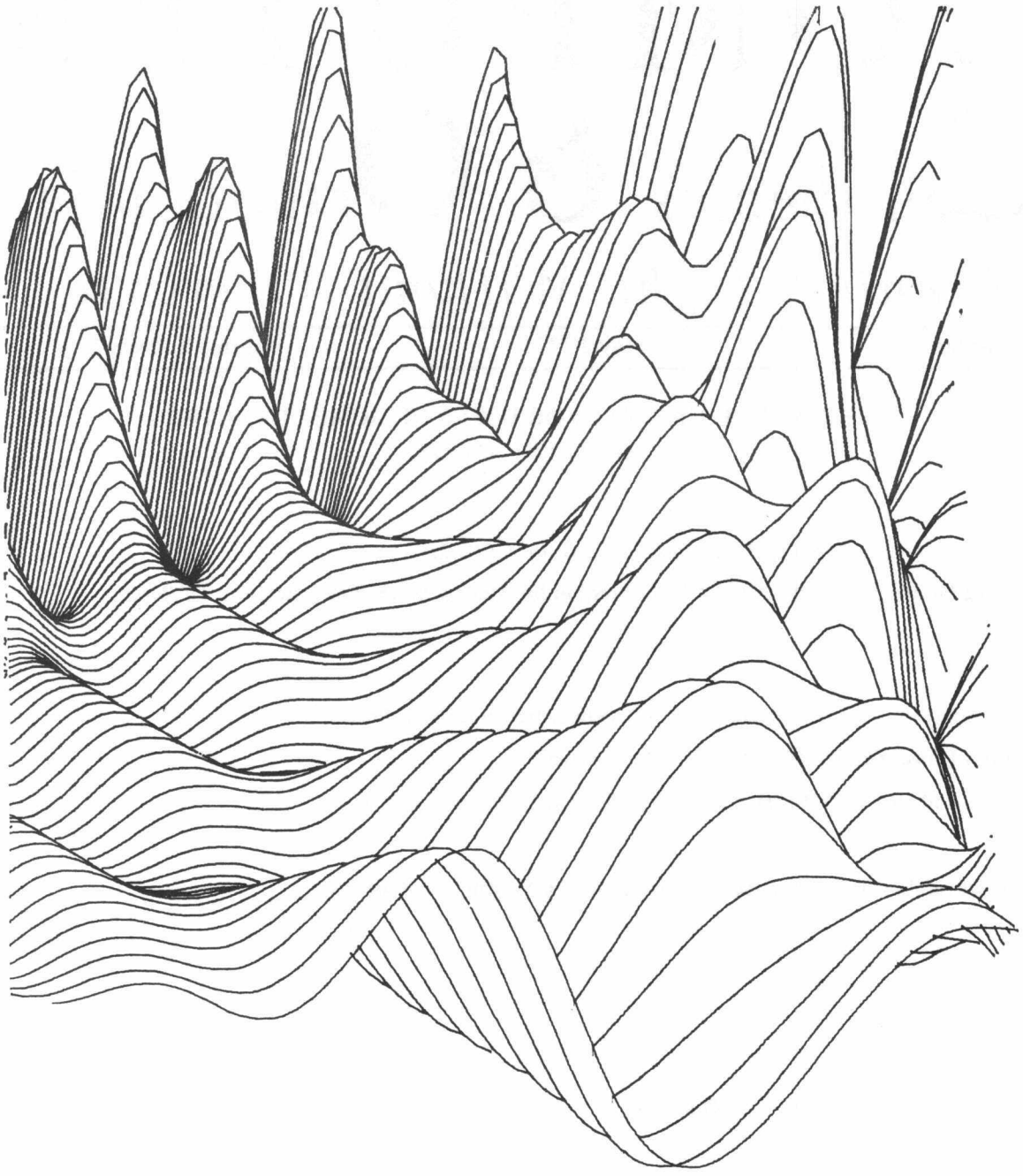
Jack Armstrong's Super Decoder Ring – Revisited, *Marvin R. Winzenread* 9

Video Chase for 8080/VDM, *Joseph J. Sanger* 10

Source Code for 8080 PILOT, Version 1.1, *John A. Starkweather, Ph.D.* 18

A Practical, Low-Cost Home/School Microprocessor System, *Joe Weisbecker* 34

. . . .and the usual glob of fascinating fillers



© David Dameron 1979

PROPER CREDIT & SWTPC PRAISE

Dear Jim, May 10,1977

I would like to take this opportunity to thank you and your staff for the very warm welcome I received when I wandered into the PCC offices on a recent trip to the west coast. I was very impressed by how such a diverse group of people all seem to be able to work so well together and put out an excellent group of publications. Keep up the good work!

I would also like to comment on Gary Alevy's 64 character modification for the TVT-II which appeared in the February *DDJ* without proper credits. It was first distributed by members of the Atlanta Area Microcomputer Hobbyist Club at the Dayton Hamfest in May, 1976. The modification is the work of one of our members, Dave Kemp, who spent many hours coming up with this very elegant modification. (Much more elegant, I might add than some kits that do the same for $10-$20!). While the original material was not copyrighted, I felt that your readers might like to know where the modification originated. **[WE CONCUR.–JW]**

On another subject . . .

There are large numbers of Southwest Technical Products and other 6800 systems in the Atlanta area ranging all the way from 48K systems with disks and printers down to single board evaluation kits. With the recent proliferation of 6800 software at affordable prices (usually free!) including at least 4 versions of BASIC and the simplicity of 6800 hardware, the number of 6800 based systems in use is growing rapidly. I am looking forward to seeing more 6800 articles in *DDJ* and would be interested in corresponding with anyone to exchange programs/information/ideas/problems/etc. on the 6800.

Finally . . . In reply to Mr. Goldsmith's letter in Feb. *DDJ*, I have built two SWTPC 6800 systems without the first problem and have nothing but praise for the quality of their kits; two compliments that I can not give to the MITS 8800 after spending weeks of fruitless debugging with my roommate trying to wade through several layers of corrections in their manual. I believe the SWTPC instruction book to be one of the best for any personal system that I have seen. There are many SWTPC systems in the Atlanta area and I know of no one who has had any problems in getting one up and running. My first SWTPC 6800 kit went from box opening to operation in less than 12 hours! . . . and it worked the first time that power was applied.

In closing . . . keep up the good work; *DDJ* is the best source of objective information in the computer hobby around.

Sincerely,

Jim Stratigos
2360 Clyde Dr.
Chamblee, GA 30341

COSMAC

Dear Sir: March 7, 1977

I have a 156 byte COSMAC operating. It plays music, measures time intervals and frequencies, interfaces both ways with a 20 mil teletype and interfaces to and from an ordinary cassette using the "KC" standard. I do all this with $5 worth of equipment in addition to the $87 worth in the Aug. 1976 *Popular Electronics* original. Software not hardware.

Is this of interest to you?

E.M. McCormick
8720 Ewing Drive
Bethesda, Maryland 20034

Yum, yum . . . YES! Send all the details and we'll be delighted to publish 'em. - JCW

COMPUTER COMPATIBLE CODE

Mr. Jim Warren, Editor
Dr. Dobbs Journal
Box E, PCC
Menlo Park, CA 94025

Reference is made to your discussion of Machine-Readable Programs in Magazine Format appearing in your Nov/Dec issue.

The provision of a computer compatible printed code is highly endorsed in this quarter. It can cut costs in both printing and reading.

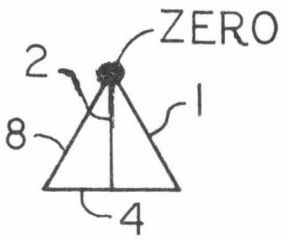
However, it is suggested that it might be better to move to a printed "character" which can also be easily read by humans and written by humans. We have tried to provide the latter. About the periphery of this letter you will find some historical dates and other printed in "computer compatible digits". These are easily read by humans as Arabic digits and more easily written. The digits are printed directly from computer words and can be read into a computer directly. No encoders are necessary.

The four element digit has been expanded into a ten-element "Computer Compatible Character" which permits the letters of the alphabet and all other characters to be represented.

I don't want to bore you with a lot of discussion. The move to a bar code is good. But it just might be better to move to a printed text which is also human readable and writeable.

George E. Row
Electronic Designer
Computer Compatible Inst. Co.
4719 Squire Drive
Indianapolis, IN 46241

Phone



MONTHLY UPDATE ON PERTEC'S MITS BUYOUT

News Release from Pertec *Received: 77-5-16*

ALBUQUERQUE, New Mexico, May 9, 1977 -- Shareholders of MITS, Inc., today approved the proposed acquisition of MITS by Pertec Computer Corporation.

The agreement calls for the exchange of 746,154 shares of PCC common stock for the approximately 966,900 shares of MITS currently outstanding.

The acquisition will become effective by May 20, pending the completion of certain legal formalities.

Pertec Computer Corporation is based in Los Angeles and designs, manufactures, markets and services key data entry and processing systems and computer peripheral equipment.

MITS, based in Albuquerque, was founded in 1969 and is a pioneer and leader in the development of microcomputer systems which are used by small businesses and computer hobbyists.

MITS' revenues for the year ending December 31, 1976 were approximately $5.7 million and the company reported net income of $370,000 or 39 cents per share.

Pertec Computer Corporation is expected to report near-record earnings per share on sales of more than $94 million for the year ended March 25, 1977.

NOSTALGIC?

Dear Sir: Nov. 4, 1976

As a reader of Dr. Dobb's Journal of Computer Calistenics I have noticed the absence of any columns on computer nostalgia and antique computers. Antique computer interest appears to be growing because it is perceived in perspective to and in contrast with microprocessors. Nostalgia discussions would compare old computers with microcomputers in the area of first cost, computing power, logical organization, speed, electricity consumption, size, wieght, reliability, etc.

Antique computers include the desk size drum and vacuum tube machines of the '50's, early magnetic core minis of the '60's, and the first Wang and HP desk top calculators of the late '60's. The oldest antique computers include the Royal-McBee LGP-30, Bendix G-15, Burroughs E101, IBM 650, Elcon 125, Alwac, Monrobot VI, Univac I and II, and the IBM 700 and 7000 series.

Since only a few of these machines were ever built, and many have already disappeared, there is a real danger they may become extinct. We can help preserve our computer heritage by publishing interesting articles on computer nostalgia and antique computers.

Yours truly,
Don Nyre
Computer Hobbyist

NIBL ERRATA & SUPPLEMENTARY COMMENTS

Dear Jim, Dec. 21, 1976

I have a few corrections, additions and comments on the NIBL article in *DDJ,* Vol. 1, No. 10:

First, I forgot to mention anywhere in the article that during execution of an INPUT statement, typing the Control/C combination will abort input and cause the NIBL system to return to edit mode.

Secondly, I got the meanings of "highest" and "lowest" precedence backwards in the table of operator precedence. The table is correct; but highest precedence means that the operator is applied last and lowest precedence means that it is applied first, which is the opposite of what the table said. I hope I didn't confuse too many readers on that point.

Finally, purists may find the grammar for NIBL a bit impure, since it mixes up the phrase structure grammar and the lexical grammar in one big mess, with the undefined stuff being explained in a footnote. This problem plagues nearly all language descriptions I've seen that use BNF or some other type of formal grammar. Maybe Tiny PASCAL will get done right for once.

As for my comment on the Burroughs 5500 machine, I wasn't being totally serious. But I have often thought about writing an emulator for some ideal stack machine that executes all the basic operations of a language like XPL. This could be the starting point for some fancier software like an XPL compiler written in this ideal machine language, and which emits code for the same machine. (Or maybe when microprogrammable microprocessors become practical, the emulator could be done in microcode.) But I don't have time for this right now – it's just a grand dream.

Sincerely,
Mark Alexander 2900 Semiconductor Dr. M.S. 690
National Semiconductor Corp. Santa Clara, CA 95051

P. S. In the paragraph "on reading the grammar" on page 37, the word "identities" should read "identifiers." Now that I look more closely, there are dozens of other typos. Sigh . . .

IN DEFENSE OF DECIMAL

For many years now, assembly language programmers have been brainwashed and browbeaten into learning binary arithmetic and often octal or hexadecimal as well. I believe the time has come to examine the necessity for this. Everyone knows that computers speak binary in the privacy of their internal logic. So it seems to make sense to speak to a computer in its native tongue—or does it?

Personally, I love hexadecimal. It works beautifully for any machine whose word length is a multiple of 4, and a hex dump is a joy to behold with its enigmatic blocks of alphabet soup spiced with arabic numerals. However, I must admit that newcomers to the joys and wonders of programming may not share my appreciation and enthusiasm for this magnificent artform. The plain truth is that learning binary, octal, or hex is a pain in the accumulator, and total acclimation is usually a lengthy process. Byters stuck with nothing but toggle switches have little choice but to use binary. Likewise, those blessed with octal keypads will learn to live with split octals. If you've never experienced a split octal you have no idea how painful it can be (sometimes a truss helps).

But most people have ASCII keyboards, and even primitive assemblers will usually accept decimal values in source programs. It would be no problem to rework an 8080 or 6800 assembler to list object code in decimal, with split-decimal addresses. I ask you, is 255:255 any worse than 377:377?

Jim Day 3/17/77

NOTES ON: GODBOUT & S.D. SALES 4K BOARDS, POLY VIDEO BOARDS, & MINI MICRO MART

Dear Dr. Dobbs: April 19, 1977

Due to a small budget I am expanding my Altair (now over two years old) at a snail's pace. This means 4K memory boards, and the cheapest available. I thought other readers might be interested in a comparison of the Godbout and S.D. Sales boards.

Godbout has DIP switch address select and memory protect (worthless on an IMSAI). The S.D. Sales board is not readily modified for DIP switch select. From here on S.D. beats Godbout. S.D. has solder mask boards, 4 regulators with heat sinks (as opposed to 3 without sinks), tri-state to the input buss (open collector on Godbout - OK but not standard), and a better schematic.

Neither company uses all the status and strobe lines available to control r/w, as Altair does; however some of those lines are unnecessary.

A word about Polymorphics video: a good board but runs hot. I inverted the control bit (bit 7) with an extra NAND gate on the board to conform to PTCo. VDM specs.

Finally a word about Mini Micro Mart. I have ordered some hard hardware (power xfmr, 9 slot mother, 100 pin connectors) from them on different occasions. Items are received quickly and prices are good. The xfmr is a real bargain for anyone plagued by Altair brownouts.

Thanks for your excellent magazine. I'll subscribe when I get a job!

Sincerely,
Glenn D. Nelson
Brown University
Providence, Rhode Island 02912

WHAT TO GET YOUR COMPUTER FOR ITS FIRST BIRTHDAY

Some inexpensive products you might want (or not want) to have around.

Jef Raskin

Some of you may have missed your computer's first birthday. If it isn't too offended you can celebrate its second birthday. Usually a computer shows that its feelings are hurt by not reading cassettes reliably. Here are some under $10 or so items that may put you in good with the CPU.

For $2.95 you can buy a Hexadaisy. This is a mechanical circular slide rule style hexadecimal calculator. I have somewhat mixed feelings about this device, and some colleagues have expressed similar feelings. First, however, the positive side. There is one pair of scales, marked from 00 to FF. The inner scale has the decimal equivalents of those 256 quantities. Black bands at every even number on both scales either line up or make a checkerboard pattern so it is easy to see when addends line up. The scales are linear. Thus two digit hexadecimal addition and subtraction are easy to do. For four or more digit sums, carries and borrows have to be done and the partial results carried in your head or on paper. The procedure is specified in the little instruction pamphlet that comes with the Hexadaisy. There is a special feature for finding relative address values on the 6800 and 6502 and similar microprocessors. Twos complements may be obtained by subtracting a number from zero.

Having 256 divisions around the circle is fitting things pretty tight, and this slows things down a bit. But the printing on the plastic discs is clear. The front also has the binary equivalents of the hex digits as well as the values (in decimal) of the first sixteen powers of two—and the first sixteen negative powers of two as well. The back of the Hexadaisy is blank.

The device does not help with hexadecimal multiplication or division--unless you want to reduce these to repeated addition and subtraction. I wouldn't want to. It is hard to see how a simple mechanical device for $2.95 *could* do these operations to the precision required for the computers. Nonetheless, when I first heard of the device I expected that it would be able to multiply and divide. After all, it does look a bit like a circular slide rule.

The biggest disappointment comes when you want to convert hexadecimal to decimal and vice versa. Quantities from 0 to 255 are listed of course, and this can be handy around the computer, but one hardly needs more than a chart for that. I have such a chart and refer to it all the time. There is a procedure listed in the manual. It begins: "HEXADECIMAL TO DECIMAL CONVERSION . . . This conversion can be simplified by the use of any simple four-function calculator . . ." Simplified my eye! If you have a four function (or fancier) calculator then of course you can convert hexadecimal to decimal. All you need do is form the sum of the decimal value of the rightmost digit and 16 times the next digit and 256 times the next digit and 4096 times the next and so on. Or you can do it by hand. For conversion the Hexadaisy is no more (or less) than a list of decimal equivalents of the two digit hexadecimal numbers. The same goes for conversion the other way. The real work is done by the calculator. The Hexadaisy is just a chart.

I find that mine gets used very seldom around my computer. I suspect a beginner will find it much more useful, at least until enough exposure to hex has made doing addition and subtraction quite natural. In any case, both the beginner and the expert will want a pocket calculator near their computer.

Pocket calculators start at $6.95 around here. The Hexadaisy is available from E & L Pfeiffer (nice people by the way) Computer Products, Box 2624, Sepulveda CA 91343.

Now for three products from Board Byters, all under $10. (Sounds like *Cosmopolitan*, doesn't it?)This inventive little company, which can be reached at P.O. Box 17512, Irvine, CA 92713, has a number of wooden products. My first thought was that they were crude and useless. Wrong. The cheapest has no name that I was told and is a piece of cheap pine with a bunch of slots in it. No moving parts. No IC's. I could make one in five minutes on my table saw. The significant thing is that I didn't. Board Byters did. I took mine home and chuckled. Who needs it. But that very same evening I was taking my computer apart to make some changes. The boards were scattered all over the couch as usual when I noticed the slab of pine with the slots. Aha! I put the boards in the slots. Instant neatness. The slots are just the right distance apart. One pair of slots is further apart for a board with tall components. OK Board Byters. This one *is* useful. Small production lines would also like to have a few of these.

Next in line is a little box with a dowel that is made for holding a paper tape, keeping it neat while it reads into the ol' TTY or what have you. There were no instructions at all, so I gave it to a colleague who has had a little experience with computers and she quickly built it. It was done correctly. I had seen one at the West Coast Computer Faire, but I didn't tell her how it went together. This is a clever little idea that will help keep your feet off the programs. I hate paper tape myself, but I just happen to have had a TTY around today. Board Byters wins again, and fills a little niche. (I've often built a device functionally similar by bending a coathanger and slipping one end under a leg of the Teletype.)

The item that gives the company its name is the Board Byter. This is a very complex device . . . only if it is compared to the board with the slots. It has three wooden peices, four screws, a nut, bolt, washer and wingnut. What it does is to hold a PC board up to a foot wide at a 45 degree angle to the table so that you can place components easily. At first I feared it far inferior to my much more expensive Panavise PC board holder. And in some ways it is. But for building boards it is not bad at all. At the 45 degree slant it is easy to insert components. Then you can turn the whole thing over (it is very light in weight) and solder the parts into place. The spreader bar gets in the way a bit.

I am not sure how any of these wooden aids were *intended* to be used. And I fear I may have missed something. But there were no instructions with any of them, and only the Board Byter has even a diagram. My colleague found them easy to assemble, and so far I have found the simplest the most useful. Happy Birthday.

FRIDEN SCHEMATIC

Dear Sirs: January 25, 1977

I noticed that you have little "advertisements" spaced throughout dealing with things people would like to have. Here is a possible contender. I have a printer – the Friden unit sold by Herbach-Rademann with the controller that has all of those strange guts. I have been able to procure the schematics and wiring diagrams for the beast. Many people would like to have these, as I understand they are as scarce as hens teeth. I am selling copies for $6.00 each,which is covering the costs of copying and first class mail.

Thank you very much,

Tim Ahrens

2200 Sorret Tree Cr.
Austin, TX 78744

A TINY SUGGESTION: A *LIST* OPTION FOR MULTIPLE–STATEMENT–PER–LINE BASICS

12/29/76

Li-Chen Wang's excellent implementation of Tiny Basic allows multiple commands per line. This capability is useful but makes the logic of a Basic program hard to follow in a listing, especially for people unfamiliar with the program and/or the language. So it might be kind of nice to have an optional variation of the LIST command that would list a program one command per line. This shouldn't add much to the interpreter, since most of the code required is there already. Simply replacing all semicolons with carriage returns in the LIST output would be one way of doing it. (Yes, I know. Some strings would look funny.) Maybe such a Modified-LIST command could be called MIST, which could be abbreviated M.

Jim Day

A RESIDENT PL/M COMPILER FOR THE 6800 – SOON(?)

Dear Jim: Dec. 30, 1976

In Volume 1, Number 10 of *DDJ* Larry Walker put a high priority on a procedure-oriented language that can be compiled. I agree, and thought I would let you people know that for the last several months I have been working on a 6800-resident PL/M compiler.

I chose PL/M since it is ideally suited to systems programming; in addition it appears easily extendable for other applications (more about that later). One of the best things about PL/M is that it can be compiled in only one pass; this means you can have the source coming in off one cassette drive, and the object going out to another with no intermediate files.

So far I have completed the overall design of the compiler, plus coding of the lexical scanner, parser, and symbol table routines. I am currently writing the code-generation routines. For any compiler freaks out there, I am using operator precedence for parsing arithmetic expressions, and recursive descent for everything else.

I am writing the compiler in a subset of PL/M, and then hand translating to assembly language; once the first version is up and running it will be able to compile itself. The subset includes only those features of the language necessary to write the compiler. Notable exceptions are the lack of parameters in procedures (using global variables instead); simplified expressions; no based variables; and no GOTO statements. Later I will add these features incrementally, until the full language is implemented.

Since I have a Sphere 330 system, the first released version will be for that machine. Although the 330 has a 20K RAM, I am hoping the compiler will fit in a 16K system. The code for the initial (subset) version looks like it will be around 6K. After the Sphere version is up I plan on releasing a MIKBUG version. Hope to have something available by this summer.

Then come extensions, such as:

1) carriage control and a title capability
2) qualified variables and structure declarations down to the bit level
3) "real" macros, implemented as %procedures (as compared to the DECLARE LITERALLY)
4) multiple-dimensioned arrays
5) floating point arithmetic (probably TSC format, i.e. 9 significant digits with exponent range -99 to +99)
6) varying character strings, and string functions

It is probably not feasible to implement all of these in a single version of the compiler, and still keep the size reasonable. I have already been involved in adding the first three extensions to Intel's PL/M compiler via a SPITBOL preprocessor; the resulting language has been used for the past year in writing control software for an 8080.

Note that I am not proposing formatted I/O, even though PL/M supports only single character input/output. I feel that formatted I/O can best be handled via a library of suitable procedures, called at run-time.

I welcome suggestions from other PL/M users regarding possible extensions.

Sincerely yours,

Thomas W. Crosley

14-3 King Arthur Ct.
Northlake, IL 60164

Tiny BASIC on a Honeywell 112 in Europe

Dear Sir, Dec. 15, 1976

While visiting America recently I had the opportunity to buy several copies of *Dr. Dobb's Journal of Computer Calisthenics and Orthodontia* and enjoyed very much the general style of the magazine and the interesting software listings.

I don't own an 8800 or 6800 but a special 12 bit machine with a similar instruction set to PDP-8, however it usually doesn't prove too difficult to convert most of the programs. I was especially interested in Tiny BASIC for my machine with one or two additions for ASC11 character string handling.

My system comprises a 12 bit Honeywell 112 mini with 8k of core memory, a creed 5 unit keyboard/printer/paper tape punch plus a separate 5 unit creed paper tape reader, cassette tape working at 3,300BPS which allows loading of 4k in under 15 seconds a Digital group video display (still under construction) and a South West Tech. keyboard (for ASC11 input).

Yours sincerely,

Ian D. Spencer

Am-Museum 3,
5353 Mechernich
West Germany

6502 RESIDENT ASSEMBLER PROGRAM AND TINY BASIC INTERPRETIVE PROGRAM NOW AVAILABLE IN ROM FOR $200

Santa Clara, California, April 1, 1977 . . . Microcomputer Associates Inc. today announced shipment of their Resident Assembler Program (RAP) and Tiny BASIC interpretive program on ROM chips. Two 2K x 8 ROMs comprise the software ROM package, housing the 1.75K Resident Assembler and the 2.2K Tiny BASIC program.

Formerly contained in seven 1702A PROMs, RAP is the only single pass Resident 6502 assembler available today. Statements are entered either from paper tape or directly from a terminal keyboard. RAP generates a listing and places object code into RAM for immediate execution. A minimum of 4K x 8 RAM memory is needed with the users' 6502 microcomputer. RAP allows a 5602 microcomputer to function economically as a microcomputer development system. Following assembly the programs can be debugged using the debugging facilities of DEMON, Microcomputer Associates' DE-bug MONitor program housed in the 1K ROM section of a 6530 ROM/RAM-I/O-Interval timer circuit. A text editor is included.

For those microcomputer users who prefer to use a somewhat higher level language Tiny BASIC permits immediate entry and execution of Tiny BASIC language programs. Statements include: LET, IF . . . , THEN, INPUT, PRINT, GOTO, GO SUB, RETURN, END, REM, CLEAR, LIST, RUN, and functions RND (Random Number generator) and USR (User SubRoutine) that allows branching, with arguments to assembly language subroutines. ROM software has been designed so that most I/O devices can be used.

The ROMs are totally pin-compatible with 2708-type PROMs.

The RAP/Tiny BASIC ROM package (SW101) is priced at $200 and includes full documentation with deliveries from stock to 30 days ARO.

RAP is also available on a set of seven 1702A PROMs (SW200) for $295. Tiny BASIC is available either in paper tape format (SW300) for $25 or on a set of nine 1702A PROMs (sw201) for $275. All software is fully documented with deliveries from stock to 30 days ARO.

For further information contact Darrell Crow, Microcomputer Associates, 2589 Scott Blvd., Santa Clara, CA 95050, (408)247-8940.

TESTING THE DZ80-80 CPU
NIFTY Z-80 CONVERSION FOR ALTAIR AND IMSAI COMPUTERS

There isn't a whole lot to say—and that is much to the credit of Dutronics' Z-80 conversion kit. It was ridiculously easy, and it worked. But let the winged chariot of my story take you back a few months . . .

Going out of the Palo Alto Byte Shop I was hailed by Mr. Dutra of Dutronics. Since I had tested his 8K boards and given them a justifiably good review, he was not adverse to letting me try out his new Z-80 kit. The idea behind the DZ-80 is simple. The Z-80 CPU chip is quite similar to the original 8080, but nothing is for nothing. Along with the new capabilities (such as a duplicate set of registers, and block move instructions, to mention a few) comes the other side of every improvement: incompatibility. You can't just plug a Z-80 chip in where the 8080 goes. It won't work. There are a host of differences. Little ones, to be sure, but you would need a small handful of gates to make the Z-80 exactly match the 8080.

And this is what Dutronics has done for you. On a small printed circuit they have enthroned a Z-80 with a court of attending IC's and this whole royal entourage *plugs in* to your 8080 socket! To enhance the impression of majesty the printed circuit is done in royal blue, with silver traces, and elegant red graphics labelling the components. It is a very pretty board. But back to our story.

Mr. Dutra and I walked out to his car, and in gallant fashion he opened a sealed box containing 5 of his kits. At random I selected one. This will assure the reader that I did not get a specially hand selected board. Gloating with my new treasure I drove home to my secret computer grotto. I rushed in and turned on the soldering iron, got out the strippers and cutters and meticulously opened the package. Oops. Turn off the soldering iron. Put away the strippers and cutters. (I was tempted to let one of the strippers stay; but I had work to do.) This is *not* a kit. It is fully assembled. And very nicely assembled it was, too.

So I left the grotto and went to the IMSAI computer sitting upstairs. I turned off the power and pulled the plug (don't be half-safe) and removed the CPU card. Here's what it takes to install the DZ80-80: With conductive foam behind the 8080's socket on the IMSAI CPU board, and using a chip puller, I removed the 8080 from its socket and put it into a block of black conductive foam. Ditto the 8212 on the CPU board. Fortunately, I always build my computers fully socketed for just such an occasion. Then I removed the protective plastic from the DZ80-80 miniboard and plugged it into the 8080's socket on the IMSAI CPU board. A little cable from the DZ80-80 runs to the 8212 socket. It has two clearly marked positions—one for the ALTAIR 8800 and one for the IMSAI 8080. I chose the appropriate one. Pin one went to pin one everywhere.

With trepidation I plugged in the IMSAI CPU card, then plugged in the computer, and (after re-reading the manual to make sure that I had done everything right) turned on the computer! No smoke. This was good, as I have a No Smoking sign prominently displayed in my office. In fact everything looked normal. The switches and lights did what they were supposed to. First time up, it worked. 10 points to Dutronics.

These are not the good old days when all you got in a kit was a handful of surplus parts, a hastily-drawn schematic, some moth-eaten solder, and a pink slip of paper saying, "Inspected by 87". Dutronics supplies both software and documentation. First of all there is a carefully written manual. The 4-inch square blue circuit board is tiny compared with the big bright orange binder. Is this beginning to sound like a fashion article? Blue boards, red silk screening, orange binders, porcelain white CPU's with 14 carat gold inlay and yellow strands of paper tape. Paper tape? Yes, indeed. Software. The "Dutz" monitor. This is really a hardware review article, but software there is. The monitor takes 1K bytes and is intended as a system check-out and program development tool. Since it was designed around a TTY and paper tape I/O, neither of which I have, I could not give it a proper test.

This is a real problem for software designers. What can you assume that your user has? In my case I hate teletypes: they are noisy and have poor print quality. And I am rich enough to have something a bit slicker for my own homebrew system. So I can't use the supplied monitor. I hope you can, because it looks neat. Like most microprocessor monitors it renders the front panel superfluous. You can load and dump memory, do hex arithmetic, examine and modify memory locations, execute a program at a given address, move blocks of information from one place in memory to another, search memory beginning at a specified location for a given character, look at the stack, verify memory locations.

Dutronics gives the loader for the tape, and the complete assembly listing of the monitor. Bravo! The whole manual is well done. If there are problems, they will be fixable by the user. Another plus is that the MOSTEK Z-80 technical manual is included in the $159.95 price. Anything you need to know about the Z-80 is probably there.

As the manuals note, there is one incompatibility that prevents some 8080 software from running on the Z-80. In practical terms it means that ALTAIR BASIC won't run on the Z-80. I tried it. It doesn't work. Other stuff that I tried worked fine. Details, details! Just one tiny little difference in the way the Parity flag is handled and zap! there goes $75 worth of software. It would be easy to re-write MITS BASIC to be compatible, but MITS wouldn't let me have a listing of the source code last time I asked. So I use somebody else's BASIC which I *can* fix when I have to.

Summary: If you want to try your hand at the Z-80, and have an ALTAIR or an IMSAI the Dutronics DZ80-80 is a possible way to go. The other ways to go are Z-80 CPU boards from a number of manufacturers. One small disadvantage to the piggyback board is that it protrudes far enough so that I can't put the IMSAI CPU board in the first slot (of the exorbitant 22 that I have) but have to move it back a position. This wastes a slot. Such is life. The compensating advantage is the somewhat smaller cost of the DZ80-80. A hundred and sixty bucks built isn't bad at all. Computer shops have the conversion kit, and you can write to Dutronics at PO BOX 9160, STOCKTON CA 95208.

About writing: I get a lot of letters. While I try to reply to every one (even those that seem to require writing a book in response) I feel much more like replying to those including a self-addressed stamped envelope. Personal friends and relatives may ignore this last paragraph.

Jef Raskin March 1977

BAR CODE PRINTER

Since this goodie is priced at $7,487 (including a display and keyboard), it's not for the novice. However, it appears that it might be just dandy for printing machine-readable "text" – programs and data – as is being proposed in *Byte* magazine (see their issues, beginning with Nov., 1976). It is two-hammer printer capable of printing a bar code plus a separate line of human-readable text. It uses a carbon ribbon to make the bar codes with the resolution and contrast necessary for scanning with an optical "wand."

The widgit is available from Interface Mechanisms, 5503 232nd St. SW, Mountlake Terrace, WA 98043.

RESULTS FROM SOME ARITHMETIC BENCHMARK KERNALS RUN ON MINIS & MICROS

Dear *DDJ,* Oct. 21, 1976

I began collecting this data about 1½ years ago when the MITS road caravan gave a demonstration of the 8800 in Palo Alto, CA. The last input came from the Astral 2000 demonstrated at the Computer Show in San Francisco in late 1976.

I began collecting the data to use in comparing various sytems for use in scientific calculations. Thus I tested the math routines. This information should not be taken as an indication of comparative speeds in non-analytical operations such as sorting. I encourage others to send in additional entries for other systems. They may also wish to establish some other test routines for sorting, etc.

The times given may be in error by as much as .1 seconds because of human reaction time. The very short times shown for the 2100 (with ANA BASIC) and HP-9825 (in HPL) were obtained by using the test routines with 10 times the specified loop count and dividing the result by 10.

There are a number of interesting things that can be learned by looking at the table. For instance it appears that all the 8080 and 6800 programs internally evaluate tan(x) = sin(x)/cos(x) whereas the Hewlett Packard systems use some other method that is more time-efficient, but probably not as memory-efficient.

Bruce Kendall
334A Camille Ct.
Mountain View, CA 94040
(415) 967-3613

TEST PROGRAMS FOR MATH ROUTINES
(mostly of BASIC Language Systems)

Programs:

```
A   10  X=3.1234567
    20  Y=4.98765432
    30  For I=1 to 500
    40  Z=(X+Y)*Y
    50  Next I
    60  End

B   10  S=2345
    20  For I=1 to 100
    30  Z=SQR(S)
    40  Next I
    50  End

C   10  S=2.71
    20  For I=1 to 100
    30  Z=Tan(S)
    40  Next I
    50  End

D   10  S=2.71
    20  For I=1 to 100
    30  Z=Sin(S)
    40  Next I
    50  End

E   10  S=2.71
    20  For I=1 to 100
    30  Z=Exp(S)
    40  Next I
    50  End

F   10  S=2.71
    20  For I=1 to 100
    30  Z=Log(S)
    40  Next I
    50  End
```

SYSTEM	PROGRAM EXECUTION TIMES IN SECONDS A	B	C	D	E	F	Z=ATN(S)	Z=S↑S
*HP – 9825	0.55	0.35	0.44	0.48	0.42	0.42		
*HP – 9830	11	3.5	8.5	12	7	6.5	9.5	12.2
HP – 2100 (with floating point hardware)	1.3	0.25	0.53	0.44	0.32	0.27	0.51	0.5
Altair-IMSAI with no wait states								
8K MITS BASIC 1.1	9.5	--	--	8	--	--		
8K MITS BASIC 3.1	4.8	5.2	5.4	2.8	3.3	2.5	3.7	5.3
12K MITS BASIC 3.2	5.9	5.2	5.4	2.8	3.3	2.5	3.7	5.3
IMSAI 4K BASIC	11.5	15	--	--	--	--		
IMSAI 8K BASIC 1.3	14.8	4.8	21.5	11.5	13	14.5		
PTC FOCAL	8.3	3.5	--	5.2	7	5.5		
PTC 5K BASIC	6.5	13	11	5.2	--	--		
ASTRAL 2000 (6800 μP) with 6 digit precision.	18.5	13	--	34				

* – Arithmetic precision greater than 6–digits

THREE NEW PUBS

Dear Jim, Dec. 23, 1976

New publications include:

The *Personal Computing Directory* ($4.95) containing names, addresses, phone number, etc. of people, places, and things of interest to computer hobbyists. Listing and advertising in the directory are said to be free of charge. Box 134, Harvard Square, Cambridge, MA 02138.

The Universal Elixir and Other Computer Projects Which Failed, by Miles Benson. ($9.00 pp.) Ed Bride, Editorial Services Dept., *Computerworld,* 797 Washington Street, Newton, MA 02160.

Compewters Newxer!! Mak Errors, by Ray Hadley and Henry Hudson. ($4.95) R. G. Hadley, P. O. Box 5306, Salem, OR 97304.

Sincerely,
Jim Day

17042 Gunther Street
Granada Hills, CA 91344

A DATAPOINT 2200 DISASSEMBLER

Gentlemen: 26 March 1977

Just a note to let any of your readers who might be interested know that I've written and have running a Disassembler for the Datapoint 2200 computer. I have it running under Datapoint's DOS.a. The Disassembler takes disc resident object code runnable under Datapoint's DOS and converts it to an Assembly listing. The output can be directed to a printer, CRT, or th a disk file. If directed to a disc file, it can be edited and reassembled with the ASM5 assembler.

Anyone interested in obtaining a copy of the Disassembler is invited to contact me at the address below.

Keep up the good work–Dr. Dobb's Journal is great!

Regards,

William T. Mitchell
8605 Farnham Court
Hanover Park IL 60103

LINE-BY-LINE RESIDENT ASSEMBLER NATIONAL'S SC/MP

Santa Clara, CA – National Semiconductor Corporation is now offering a line by line resident assembler firmware kit, designed for use with its SC/MP LCDS (Low Cost Development System). Known as SUPAK, the $300 assembler is contained in eight PROM/ROM devices that can be plugged into a blank ROM/PROM card (ISP-8C/004B) which is available from National. The entire assembled card is then inserted into the LCDS/teletype system.

"SUPAK is the only firmware kit of its kind. It greatly enhances the flexibility and capability of theSC/MP LCDS, making it both the most inexpensive and effective SC/MP development tool," said Hashmukh Patel, National's product marketing manager for low-cost microprocessors. SUPAK is a 4K byte package that consists of three programs: a line by line assembler, a paper tape line editor and a PROM tape punch program.

The line by line assembler accepts a program in limited assembly language from a keyboard or paper reader, and then assembles it directly into RAM.

The paper tape line editor, which allows insertion, deletion or replacement of lines of program source code, punches either leader or trailer.

The PROM tape punch program punches the contents of a specified memory range, in BPNF or complemented binary format, onto paper tape. This tape could be used to program memories using a standard, commercially available PROM programmer such as the DATA I/O.

SUPAK requires the LCDS firmware, but will run on either a SC/MP or a SC/MP II (n-channel) LCDS. It comes as a set of eight MM5204/MM5214 ROM IC's, designated ISP-8F/111.

JACK ARMSTRONG'S SUPER DECODER RING – REVISITED

by Marvin R. Winzenread

In *DDJ* Vol. 1, No. 9, Jon O. Stedman presented the Playfaire-Digraphic Cryptographic Cypher. I found the article fascinating and started to write a program that would do the coding and decoding.

The Playfaire method has three deficiencies to my way of thinking. First it requires a knowledge of semantics to decode (e.g. the choice of I or J and X or space). This is easy for a human but difficult for a computer. Second it is not symmetric; coding and decoding are not the same operation when the two characters are in the same row or column. Third it translates only messages and not general text including punctuation symbols and numerals.

The following changes were made to remove the first two deficiencies:

1. Omit X and Y from the array and leave intact any character not in the array. See Statement #500.
2. Define the conjugate of (I,J) to be (6-I, 6-J) and use that for double letters and letters in the same row or column. See Statement #520.

Here is a LIST and RUN of the program. It is written in ALTAIR 8K BASIC.

This RUN shows the 5 x 5 array. It can be suppressed by deleting statements 301 and 311.

If you want the program to code and decode generalized text (including numberals and punctuation symbols) expand the array to a 7 x 7 array by changing statements 80, 110, 120, 440 and 520.

```
LIST

70 CLEAR 500
80 DIM M$(5,5)
90 M=0
100 PRINT"KEYWORD ";:INPUT K$
110 C1$="ABCDEFGHIJKLMNOPQRSTUVWY "
120 L=0:FOR I=1 TO 5: FOR J=1 TO 5
130 IF L=LEN(K$) GOTO 200
140 L=L+1 :C$=MID$(K$,L,1)
145 IF L=1 GOTO 300
150 FOR K=1TOL-1 :IF C$=MID$(K$,K,1) GOTO 130
155 NEXT K
160 GOTO 300
200 M=M+1
205 C$=MID$(C1$,M,1)
210 FOR K=1TOLEN(K$): IF C$=MID$(K$,K,1) GOTO 130
220 NEXT K
300 M$(I,J)=C$
301 PRINT M$(I,J);
310 NEXT J
311 PRINT
320 NEXT I
400 PRINT"TYPE IN STRING TO BE TRANSLATED.":INPUT K$
401 O$=""
410 FOR K=1 TO LEN(K$) STEP 2:C$=MID$(K$,K,1) :B$=MID$(K$,K+1,1)
420 IF B$=""GOTO 600
430 C1=0:C2=0:B1=0:B2=0
440 FOR I=1 TO 5: FOR J=1 TO 5
450 IF C$<>M$(I,J) GOTO 460
455 C1=I :C2=J
460 IF B$<>M$(I,J) GOTO 470
465 B1=I :B2=J
470 NEXT J :NEXT I
500 IF C1=0 OR B1=0 GOTO 600
510 IF C1<>B1 AND C2<>B2 GOTO 550
520 B$=M$(6-C1,6-C2):C$=M$(6-B1,6-B2):GOTO 600
550 B$=M$(C1,B2) :C$=M$(B1,C2)
600 O$=O$+C$+B$
610 NEXT K
700 PRINT "THATS:":PRINT O$
800 GOTO 400
OK
RUN
KEYWORD ? JACK ARMSTRONG THE ALL AMERICAN BOY
JACK
RMSTO
NGHEL
IBYDF
QUVW
TYPE IN STRING TO BE TRANSLATED.
? NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THIEF COUNTRY
THATS:
ELPJFYOKGHOKFBKLIFJOG FNMLFTOALCOKFIS CTOKFIESKLBJKFFIOKYNTNUFWSFEIS
TYPE IN STRING TO BE TRANSLATED.
? ELPJFYOKGHOKFBKLIFJOG FNMLFTOALCOKFIS CTOKFIESKLBJKFFIOKYNTNUFWSFEIS
THATS:
NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THIEF COUNTRY
TYPE IN STRING TO BE TRANSLATED.
?

OK
```

Fill in Keyword

Fill in other characters

Set up matrix M$(5, 5)

Pick out pair of characters C$, B$

Find location of character pair in matrix C$=M$(C1,C2), B$=M$(B1,B2)

If not in matrix leave intact

Same row or column omit conjugates otherwise follow Playfaire Cypher

the 5x5 matrix

To code:

Or decode:

VIDEO CHASE FOR 8080/VDM

by Joseph Jay Sanger

245 E. 25th No.11-B
New York NY 10010

March 14, 1977

There have been many interesting games published to run on personal computers. However, the majority, since they are written in BASIC, assume the existence of a TTY or a TTY-like device; i.e., a display device utilizing a 'line-at-a-time' scrolling mode of display. However, many are lucky enough to have memory-mapped video displays - and those who happen to own Processor Technology VDM-1's in particular will be able to take advantage of this program.

I was particularly intrigued by the game 'CHASE', recently published in Kilobaud, issue number 2 (written in BASIC). The game is played within a high voltage fence, sprinkled with high voltage posts. Five security robots are in residence, assigned the task of catching intruders such as yourself. As you move about, they move directly towards you - your only chance to survive is to maneuver the robots into the high voltage posts - before one of them catches you.

In my version, which is written in assembly language for the 8080, you specify in which direction to move by typing the appropriate key on your keyboard:

```
T   Y   U
  \ ^ /
G - H - J
  / ! \
B   N   M
```

THIS COMPASS ROSE IS CONTINUALLY DISPLAYED IN THE UPPER-RIGHT CORNER OF THE SCREEN DURING THE GAME

Θ = YOU

Ω = ROBOTS

X = FENCE/POSTS

Note - you can remain in place by typing 'H'.

The computer keeps score (and displays it in the lower right corner) and the first 'team' to reach 9 games is the winner of the series. At the end of a game, type any key to begin a new one.

You may specify the level of difficulty of each game at the beginning of the series or at any time by typing a carriage return; 0 is the easiest level, 9 the hardest. You can force a new series at any time by typing 'ESC' (01BH), with score 0:0.

If you really get into a bind, I have included a 'panic' button. By typing 'P', you may be lucky enough to be instantaneously transported out of danger to a new (random) position - but the probability of a successful transport depends on the level of difficulty of the game - approximately 50% for level 0 down to a mere 12.5% at level 9! At level 9, there is also an additional penalty for use of this feature. . . .

One difference from other versions of CHASE - if two robots collide in this version, they do no annhialate, but travel as a pair - with some strangely unpredictable consequences! (This was originally a 'bug' in my program, but so cute that I decided to leave it in!)

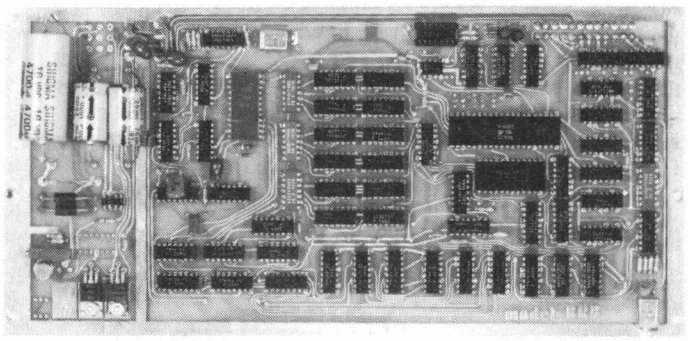
```
HARDWARE SPECIFICATIONS FOR MY VERSION INCLUDE:
      - VDM SITTING AT 0CC00H
            CONTROL PORT 0C8H
      - 3P+S WITH STATUS PORT 0
                  DATA PORT 1
                  DAV 40H (GOES HIGH ON KEYPRESS)
      - RAM REQ'D: ~ 0600H RAM (1.5K BYTES)
                        (EXCLUDING STACK SPACE)
OTHER HARDWARE IN MY SYSTEM INCLUDE:
      - IMSAI 8080 COMPUTER
      - PROC. TECH. ALS-8 DEVELOPMENT SYSTEM
                  & SIM-1, TXT-2
      - ADDITIONAL 28K RAM
      - 2.5K EPROM (6834'S )
      - TARBELL CASSETTE INTERFACE
      - LA36-II DECWRITER
```

I intend to write two more versions of the game - one allowing to players to compete, and the other making the robots move towards you in real-time, demanding constant evasive action on your part.

There are many other games that can be adapted to run on memory - mapped video displays with great success (a case in point is Processor Technology's TREK-80 - a really fine version of STARTREK). I hope that this article acts as a stimulus for some of you lazier types out there (like myself) to get to work!

VIDEO TERMINAL BOARD PRICE REDUCED 20% (DOWN to $395)

WINTEK Corp. reduced the price on their Model BRB video terminal OEM board from $495 to $395. The single printed circuit board contains all the electronics for an upper case ASCII alphanumeric CRT terminal. It generates composite video for displaying 16 rows of 80 characters on any TV monitor. It includes an RS-232C interface for all standard rates to 9600 baud, a parallel keyboard interface, and comes with or without power supply. WINTEK Corp., 902 N. 9th St., Lafayette, IN 47904; (317) 742-6802.



READER'S GUIDE TO COMPUTER HOBBYIST PERIODICAL LITERATURE

The *Periodical Guide for Computerists* is a new 20 page book that indexes over 1,000 personal computing articles from 15 magazines for January - December 1976. The articles are indexed under more than 100 subject categories. Indexed are magazine articles, letters from readers, book reviews and editorials from both hobbyist and professional publications.

The books are available form: E. Berg Publications, 1360 S.W. 199th Ct., Aloha, Oregon 97005 for $2.50 each postpaid and also from local computer stores.

A forthcoming issue in July will index January - June 1977 articles.

```
0000                    0000 *         * * * * * * * * * * * * * * * *
0000                    0010 *         *     THE GAME OF CHASE       *
0000                    0020 *         *      (VERSION 2.1)          *
0000                    0030 *         * * * * * * * * * * * * * * * *
0000                    0040 *
0000                    0050 *        WRITTEN BY JOSEPH JAY SANGER (4/5/77)
0000                    0060 *           IN 8080 ASSEMBLY LANGUAGE
0000                    0070 *             USING PROC. TECH. VDM
0000                    0080 *
0000                    0090 *
0000 AF                 0100 BEGIN  XRA    A          * STORE 0:0 SCORE
0001 32 E7 05           0110        STA    YSCORE     *   YOUR SCORE
0004 32 E6 05           0120        STA    RSCORE     *   ROBOT'S SCORE
0007 DB 01              0130 ENTER  IN     DATA       * RESET 3P+S
0009 31 50 06           0140        LXI    SP,STACK
000C CD 4E 04           0150        CALL   CLEAR      * CLEAR SCREEN
000F CD 92 04           0160        CALL   SCORE      * DRAW SCOREBOARD
0012                    0170 *
0012 21 10 CD           0180        LXI    H,MS2      * WRITE GAME HEADING
0015 01 E8 04           0190        LXI    B,MNAME
0018 CD D2 04           0200        CALL   MSGO
001B                    0210 *
001B 21 CC CF           0220 DIFF   LXI    H,MS1       * MORE HEADING
001E 01 C5 05           0230        LXI    B,MDIFF
0021 CD D2 04           0240        CALL   MSGO
0024 CD A9 01           0250        CALL   INP        * INPUT DIDFFICULTY
0027 E6 0F              0260        ANI    0FH            FACTOR
0029 32 E5 05           0270        STA    DFST       * STORE IT
002C CD 4E 04           0280 START  CALL   CLEAR
002F CD 92 04           0290        CALL   SCORE
0032 16 00              0300        MVI    D,DCORN    * POINT TO UPPER LEFT
0034 1E 02              0310        MVI    E,ECORN      CORNER OF FIELD
0036 06 19              0320 TOP    MVI    B,WIDTH    * DRAW TOP
0038 CD 87 03           0330 T1     CALL   PLOT
003B 36 58              0340        MVI    M,'X'
003D 13                 0350        INX    D
003E CD 87 03           0360        CALL   PLOT
0041 36 20              0370        MVI    M,' '
0043 13                 0380        INX    D
0044 05                 0390        DCR    B
0045 C2 38 00           0400        JNZ    T1
0048 06 0D              0410 RSIDE  MVI    B,HEIGHT   * DRAW RIGHT SIDE
004A 1D                 0420        DCR    E
004B 1D                 0430        DCR    E
004C 14                 0440 R1     INR    D
004D CD 87 03           0450        CALL   PLOT
0050 36 58              0460        MVI    M,'X'
0052 05                 0470        DCR    B
0053 C2 4C 00           0480        JNZ    R1
0056 06 18              0490 BOTM   MVI    B,WIDTH-1 * DRAW BOTTOM
0058 1D                 0500 B1     DCR    E
0059 CD 87 03           0510        CALL   PLOT
005C 36 20              0520        MVI    M,' '
005E 1D                 0530        DCR    E
005F CD 87 03           0540        CALL   PLOT
0062 36 58              0550        MVI    M,'X'
0064 05                 0560        DCR    B
0065 C2 58 00           0570        JNZ    B1
0068 06 0D              0580 LSIDE  MVI    B,HEIGHT   * DRAW LEFT SIDE
006A 15                 0590 L1     DCR    D
006B CD 87 03           0600        CALL   PLOT
006E 36 58              0610        MVI    M,'X'
0070 05                 0620        DCR    B
0071 C2 6A 00           0630        JNZ    L1
0074                    0640 *
0074 3A E5 05           0650 SEED   LDA    DFST       * LAY ELECTRIC POSTS
0077 07                 0660        RLC                  (ACTUAL NO. DEPENDS
0078 47                 0670        MOV    B,A           ON DIFF. FACTOR)
0079 3E 1E              0680        MVI    A,1EH
007B 90                 0690        SUB    B
007C 47                 0700        MOV    B,A
007D CD BF 03           0710 LOOP1  CALL   RND
0080 E6 0F              0720        ANI    0FH
0082 57                 0730        MOV    D,A
0083 CD BF 03           0740        CALL   RND
0086 E6 7F              0750        ANI    7FH
0088 5F                 0760        MOV    E,A
0089 CD A9 03           0770        CALL   LIMIT
008C DA 7D 00           0780        JC     LOOP1
008F CD 87 03           0790        CALL   PLOT
0092 7D                 0800        MOV    A,L
0093 E6 FE              0810        ANI    0FEH
0095 6F                 0820        MOV    L,A
0096 36 58              0830        MVI    M,'X'
0098 05                 0840        DCR    B
0099 C2 7D 00           0850        JNZ    LOOP1
009C                    0860 *
009C 21 F6 CC           0870 ROSE   LXI    H,RCORN    * PRINT COMPASS ROSE
009F 36 54              0880        MVI    M,'T'
00A1 21 FA CC           0890        LXI    H,RCORN+4
00A4 36 59              0900        MVI    M,'Y'
00A6 21 FE CC           0910        LXI    H,RCORN+8
00A9 36 55              0920        MVI    M,'U'
00AB 21 38 CD           0930        LXI    H,RCORN+42H
00AE 36 5C              0940        MVI    M,5CH
00B0 21 3A CD           0950        LXI    H,RCORN+44H
00B3 36 5E              0960        MVI    M,5EH
00B5 21 3C CD           0970        LXI    H,RCORN+46H
00B8 36 2F              0980        MVI    M,2FH
00BA 21 76 CD           0990        LXI    H,RCORN+80H
00BD 36 47              1000        MVI    M,'G'
00BF 21 78 CD           1010        LXI    H,RCORN+82H
00C2 36 0D              1020        MVI    M,0DH
00C4 21 7A CD           1030        LXI    H,RCORN+84H
00C7 36 48              1040        MVI    M,'H'
00C9 21 7C CD           1050        LXI    H,RCORN+86H
00CC 36 09              1060        MVI    M,09H
00CE 21 7E CD           1070        LXI    H,RCORN+88H
00D1 36 4A              1080        MVI    M,'J'
00D3 21 B8 CD           1090        LXI    H,RCORN+0C2H
00D6 36 2F              1100        MVI    M,2FH
00D8 21 BA CD           1110        LXI    H,RCORN+0C4H
00DB 36 0B              1120        MVI    M,0BH
00DD 21 BC CD           1130        LXI    H,RCORN+0C6H
00E0 36 5C              1140        MVI    M,5CH
00E2 21 F6 CD           1150        LXI    H,RCORN+100H
00E5 36 42              1160        MVI    M,'B'
00E7 21 FA CD           1170        LXI    H,RCORN+104H
00EA 36 4E              1180        MVI    M,'N'
00EC 21 FE CD           1190        LXI    H,RCORN+108H
00EF 36 4D              1200        MVI    M,'M'
00F1                    1210 *                        * INTITIAL ROBOT PLACEMENT
00F1 CD 15 01           1220 ROBOT  CALL   SEL        * SELECT A RANDOM SPOT
00F4 22 EA 05           1230        SHLD   ROB1       * STORE ROBOT#1'S POSITION
00F7 CD 15 01           1240        CALL   SEL        * ETC...
00FA 22 EC 05           1250        SHLD   ROB2
00FD CD 15 01           1260        CALL   SEL
0100 22 EE 05           1270        SHLD   ROB3
0103 CD 15 01           1280        CALL   SEL
0106 22 F0 05           1290        SHLD   ROB4
0109 CD 15 01           1300        CALL   SEL
010C 22 F2 05           1310        SHLD   ROB5
010F CD 37 01           1320        CALL   YOU        * DO SAME FOR YOU
0112 C3 58 01           1330        JMP    YMOVE
0115 CD BF 03           1340 SEL    CALL   RND        * GET RND NUMBER
0118 E6 0F              1350        ANI    0FH        * STRIP OFF 4 MSB'S
011A 57                 1360        MOV    D,A
011B CD BF 03           1370        CALL   RND
011E E6 FE              1380        ANI    0FEH       * STRIP OF MSB
0120 5F                 1390        MOV    E,A
0121 CD A9 03           1400        CALL   LIMIT      * IS DE POINTING TO VALID
0124 DA 15 01           1410        JC     SEL           POINT ON FIELD?
0127 CD 87 03           1420        CALL   PLOT
012A 7E                 1430        MOV    A,M        * IS SPOT VACANT
012B FE 20              1440        CPI    ' '
012D C2 15 01           1450        JNZ    SEL        * IF NOT, REPEAT
0130 36 07              1460        MVI    M,7        * IF SO, PUT ROBOT IN
0132 05                 1470        DCR    B
0133 C0                 1480        RNZ
0134 C3 58 01           1490        JMP    YMOVE
0137                    1500 *
0137 CD BF 03           1510 YOU    CALL   RND        * YOUR INITIAL POSITION
```

```
013A E6 0F          1520          ANI     0FH     * STRIP OF 4 MSB'S
013C 57             1530          MOV     D,A
013D CD BF 03       1540          CALL    RND
0140 E6 FE          1550          ANI     0FEH    * STRIP OF MSB
0142 5F             1560          MOV     E,A
0143 CD A9 03       1570          CALL    LIMIT   * IS IT A VALID POINT?
0146 DA 37 01       1580          JC      YOU     * IF NOT, REPEAT
0149 CD 87 03       1590          CALL    PLOT
014C 7E             1600          MOV     A,M
014D FE 20          1610          CPI     ' '     * IS IT VACANT?
014F C2 37 01       1620          JNZ     YOU     * REPEAT IF NOT
0152 36 0F          1630          MVI     M,0FH   * PUT YOU ON MAP
0154 22 E8 05       1640          SHLD    YOUP    * STORE YOUR CURRENT POS.
0157 C9             1650          RET
0158                1660 *
0158 CD 0C 04       1670 YMOVE    CALL    NEAR    * IS A ROBOT ADJACENT?
015B CD A9 01       1680 YMOV1    CALL    INP     * GET KBD INPU
015E 2A E8 05       1690          LHLD    YOUP    * GET YOUR CURRENT POS
0161 36 20          1700          MVI     M,' '   * LIFT YOU OFF MAP
0163 FE 4A          1710          CPI     'J'     * IS 'J' PRESSED?
0165 CC 6C 03       1720          CZ      RT      * MOVE RIGHT IF SO
0168 FE 55          1730          CPI     'U'     * ETC...
016A CC 6F 03       1740          CZ      UR                (UR = UPPER RIGHT
016D FE 59          1750          CPI     'Y'                LL = LOWER LEFT
016F CC 5F 03       1760          CZ      UP                 ETC...
0172 FE 54          1770          CPI     'T'
0174 CC 75 03       1780          CZ      UL
0177 FE 47          1790          CPI     'G'
0179 CC 69 03       1800          CZ      LF
017C FE 42          1810          CPI     'B'
017E CC 7B 03       1820          CZ      LL
0181 FE 4E          1830          CPI     'N'
0183 CC 64 03       1840          CZ      DN
0186 FE 4D          1850          CPI     'M'
0188 CC 81 03       1860          CZ      LR
018B FE 50          1870          CPI     'P'     * IS PANIC BUTTON PUSHED?
018D CC 9D 02       1880          CZ      PANIC   *   CALL PANIC IF SO
0190 FE 48          1890          CPI     'H'
0192 CA 9B 01       1900          JZ      Y1
0195 7E             1910          MOV     A,M     * IS NEW SPOT VACANT?
0196 FE 20          1920          CPI     ' '
0198 C2 85 02       1930          JNZ     DEATH   * IF NOT, YOU ARE DEAD
019B 36 0F          1940 Y1       MVI     M,0FH   * IF SO, PUT YOU THERE
019D 22 E8 05       1950          SHLD    YOUP    * AND STORE NEW POS.
01A0 CD BF 01       1960          CALL    RMOVE   * ROBOT'S TURN TO MOVE
01A3 CD E6 03       1970          CALL    END     * IS SERIES OVER?
01A6 C3 58 01       1980          JMP     YMOVE   * AND AGAIN...
01A9                1990 *
01A9 DB 00          2000 INP      IN      STAT    * KBD SERVICE ROUTINE
01AB E6 40          2010          ANI     RDA
01AD CA A9 01       2020          JZ      INP
01B0 DB 01          2030          IN      DATA
01B2 E6 7F          2040          ANI     7FH
01B4 FE 0D          2050          CPI     0DH     * CARRIAGE RETURN
01B6 CA 07 00       2060          JZ      ENTER       RESTARTS GAME
01B9 FE 1B          2070          CPI     1BH     * 'ESC'
01BB CA 00 00       2080          JZ      BEGIN       RESTARTS SERIES
01BE C9             2090          RET
01BF                2100 *                        * ROBOT'S TURN
01BF 2A EA 05       2110 RMOVE    LHLD    ROB1    * GET CURRENT POS
01C2 CD 09 02       2120          CALL    RDEAD   * IS THIS ROBOT DEAD?
01C5 DA CE 01       2130          JC      RMOV1   * IF SO, SKIP
01C8 CD 38 02       2140          CALL    SEEK    * IF NOT,MOVE ROBOT
01CB 22 EA 05       2150          SHLD    ROB1    * STORE NEW POS.
01CE 2A EC 05       2160 RMOV1    LHLD    ROB2    * ETC...
01D1 CD 09 02       2170          CALL    RDEAD
01D4 DA DD 01       2180          JC      RMOV2
01D7 CD 38 02       2190          CALL    SEEK
01DA 22 EC 05       2200          SHLD    ROB2
01DD 2A EE 05       2210 RMOV2    LHLD    ROB3
01E0 CD 09 02       2220          CALL    RDEAD
01E3 DA EC 01       2230          JC      RMOV3
01E6 CD 38 02       2240          CALL    SEEK
01E9 22 EE 05       2250          SHLD    ROB3
01EC 2A F0 05       2260 RMOV3    LHLD    ROB4
01EF CD 09 02       2270          CALL    RDEAD
01F2 DA FB 01       2280          JC      RMOV4
01F5 CD 38 02       2290          CALL    SEEK
01F8 22 F0 05       2300          SHLD    ROB4
01FB 2A F2 05       2310 RMOV4    LHLD    ROB5
01FE CD 09 02       2320          CALL    RDEAD
0201 D8             2330          RC
0202 CD 38 02       2340          CALL    SEEK
0205 22 F2 05       2350          SHLD    ROB5
0208 C9             2360          RET
0209                2370 *
0209 7E             2380 RDEAD    MOV     A,M     * IF 'X' ON ROBOT'S SPOT,
020A FE 58          2390          CPI     'X'     * THEN HE IS DEAD; RET.
020C 37             2400          STC             * WITH CARRY SET
020D C8             2410          RZ
020E 3F             2420          CMC
020F C9             2430          RET
0210                2440 *
0210 7E             2450 CHEK     MOV     A,M     * CHECK IF ROBOTS BUMPED ANYONE
0211 FE 0F          2460          CPI     0FH     *  AND MAKE APPROPRIATE MOVE
0213 CA 85 02       2470          JZ      DEATH   * YOU DIE IF HE BUMPS YOU
0216 FE 58          2480          CPI     'X'     * HE DIES IF HE BUMPS 'X'
0218 CA 1E 02       2490          JZ      FLASH
021B 36 07          2500          MVI     M,7     * OTHERWISE HE LIVES
021D C9             2510          RET
021E 16 A0          2520 FLASH    MVI     D,0A0H  * ROBOT'S DEATH DANCE
0220 36 04          2530 FLAS1    MVI     M,4
0222 1E FF          2540          MVI     E,0FFH
0224 1D             2550 FLAS2    DCR     E
0225 C2 24 02       2560          JNZ     FLAS2
0228 1E FF          2570          MVI     E,0FFH
022A 36 84          2580          MVI     M,84H
022C 1D             2590 FLAS3    DCR     E
022D C2 2C 02       2600          JNZ     FLAS3
0230 15             2610          DCR     D
0231 C2 20 02       2620          JNZ     FLAS1
0234 36 58          2630          MVI     M,'X'
0236 AF             2640          XRA     A
0237 C9             2650          RET
0238                2660 *
0238 36 20          2670 SEEK     MVI     M,' '       * LIFT ROBOT FROM MAP
023A E5             2680          PUSH    H
023B CD 71 02       2690          CALL    SUBH        * FIND HORIZ. DIRECTION
023E E1             2700          POP     H
023F CD 7E 02       2710          CALL    DECH        * MOVE IN APPROPRIATE DIR.
0242 E5             2720          PUSH    H
0243 CD 4D 02       2730          CALL    SUBV        * FIND VERT. DIR.
0246 E1             2740          POP     H
0247 CD 5E 02       2750          CALL    DECV        * AND MOVE IN IT
024A C3 10 02       2760          JMP     CHEK
024D                2770 *
024D 7D             2780 SUBV     MOV     A,L         * ENTER WITH ROBOT'S POS.
024E E6 C0          2790          ANI     0C0H            IN HL
0250 6F             2800          MOV     L,A         * STRIP OF HORIZ. INFORM.
0251 EB             2810          XCHG
0252 2A E8 05       2820          LHLD    YOUP        * YOUR POS. IN DE
0255 7D             2830          MOV     A,L
0256 E6 C0          2840          ANI     0C0H        * STRIP OFF HORIZ. INFORM.
0258 93             2850          SUB     E           * FIND DIFFERENCE
0259 4F             2860          MOV     C,A
025A 7C             2870          MOV     A,H
025B 9A             2880          SBB     D
025C 47             2890          MOV     B,A
025D C9             2900          RET
025E                2910 *
025E 07             2920 DECV     RLC             * MOVE UP IF NEC.
025F DA 5F 03       2930          JC      UP
0262 79             2940          MOV     A,C
0263 0F             2950          RRC
0264 DA 5F 03       2960          JC      UP
0267 AF             2970          XRA     A
0268 80             2980          ADD     B
0269 C2 64 03       2990          JNZ     DN      * MOVE DOWN IF NEC.
026C 81             3000          ADD     C
026D C2 64 03       3010          JNZ     DN
0270 C9             3020          RET             * OR STAY ON THIS ROW
0271                3030 *
```

```
0271 7D
0272 E6 3F
0274 6F
0275 EB
0276 2A E8 05
0279 7D
027A E6 3F
027C 93
027D C9
027E
027E C8
027F F2 6C 03
0282 C3 69 03
0285
0285 36 0C
0287 CD 6C 04
028A 3A E6 05
028D 3C
028E 32 E6 05
0291 CD 92 04
0294 CD 01 03
0297 CD F7 02
029A C3 2C 00
029D
029D F5
029E 3A E5 05
02A1 FE 09
02A3 C2 C8 02
02A6 0E 07
02A8 CD BF 03
02AB A1
02AC FE 01
02AE CA B4 02
02B1 C3 ED 02
02B4 E5
02B5 2A EA 05
02B8 36 07
02BA 2A EC 05
02BD 36 07
02BF 2A EE 05
02C2 36 07
02C4 E1
02C5 C3 EF 02
02C8 3A E5 05
02CB DE 03
02CD F2 D5 02
02D0 0E 01
02D2 C3 E4 02
02D5 3A E5 05
02D8 DE 06
02DA 0E 03
02DC F2 E2 02
02DF C3 E4 02
02E2 0E 07
02E4 CD BF 03
02E7 A1
02E8 FE 01
02EA CA EF 02
02ED F1
02EE C9
02EF CD 37 01
02F2 F1
02F3 F1
02F4 C3 58 01
02F7
02F7 DB 00
02F9 E6 40
02FB CA F7 02
02FE DB 01
0300 C9
0301
0301 3A E6 05
0304 FE 09
0306 CA 31 03
0309 3A E7 05
030C FE 09

3040 SUBH   MOV   A,L    * ROBOT'S HORIZ. POS IN L
3050        ANI   3FH    * STRIP OFF VERT. INFORM.
3060        MOV   L,A
3070        XCHG
3080        LHLD  YOUP   * YOUR HORIZ. POS IN L
3090        MOV   A,L
3100        ANI   3FH    * STRIP OFF VERT. INFORM.
3110        SUB   E      * FIND DIFF.
3120        RET
3130 *
3140 DECH   RZ           * STAY IN COLUMN IF NEC.
3150        JP    RT     * OR MOVE RIGHT
3160        JMP   LF     * OR LEFT
3170 *
3180 DEATH  MVI   M,0CH  * PUT FLOWERS ON SPOT
3190        CALL  MSG2   * PUT NOTICE ON SCREEN
3200        LDA   RSCORE * GIVE ROBOT'S A POINT
3210        INR   A
3220        STA   RSCORE
3230        CALL  SCORE  * PUT NEW SCORE UP
3240        CALL  OVER   * IS SERIES OVER?
3250        CALL  WAIT   * WAIT FOR KP
3260        JMP   START  * NEW GAME
3270 *
3280 PANIC  PUSH  PSW
3290        LDA   DFST   *  'P' PRESS WILL LIFT YOU
3300        CPI   9         OFF TO NEW RND POS.WITH
3310        JNZ   PAN3      PROBABILITY DEPENDENT
3320        MVI   C,7       ON DIFF. FACTOR
3330        CALL  RND
3340        ANA   C
3350        CPI   1
3360        JZ    PAN5
3370        JMP   PAN4
3380 PAN5   PUSH  H      * LEVEL 9 PENALTY:  SOME
3390        LHLD  ROB1      ROBOTS MAY REGENERATE
3400        MVI   M,7
3410        LHLD  ROB2
3420        MVI   M,7
3430        LHLD  ROB3
3440        MVI   M,7
3450        POP   H
3460        JMP   PANN
3470 PAN3   LDA   DFST
3480        SBI   3
3490        JP    PAN1
3500        MVI   C,1
3510        JMP   PANOK
3520 PAN1   LDA   DFST
3530        SBI   6
3540        MVI   C,3
3550        JP    PAN2
3560        JMP   PANOK
3570 PAN2   MVI   C,7
3580 PANOK  CALL  RND
3590        ANA   C
3600        CPI   1
3610        JZ    PANN
3620 PAN4   POP   PSW
3630        RET
3640 PANN   CALL  YOU
3650        POP   PSW
3660        POP   PSW
3670        JMP   YMOVE
3680 *
3690 WAIT   IN    STAT   * PAUSE
3700        ANI   RDA
3710        JZ    WAIT
3720        IN    DATA
3730        RET
3740 *
3750 OVER   LDA   RSCORE * DOES ROBOT HAVE 9 POINTS?
3760        CPI   09H
3770        JZ    END1
3780 OVER1  LDA   YSCORE * DO YOU?
3790        CPI   09H

030E C0
030F CD F7 02
0312 CD 4E 04
0315 21 10 CD
0318 01 8B 05
031B CD D2 04
031E 3A E7 05
0321 C6 30
0323 32 27 CD
0326 3A E6 05
0329 C6 30
032B 32 29 CD
032E C3 50 03
0331 CD F7 02
0334 CD 4E 04
0337 21 10 CD
033A 01 70 05
033D CD D2 04
0340 3A E6 05
0343 C6 30
0345 32 27 CD
0348 3A E7 05
034B C6 30
034D 32 29 CD
0350 21 90 CD
0353 01 A6 05
0356 CD D2 04
0359 CD F7 02
035C C3 00 00
035F
035F 11 C0 FF
0362 19
0363 C9
0364 11 40 00
0367 19
0368 C9
0369 2B
036A 2B
036B C9
036C 23
036D 23
036E C9
036F CD 5F 03
0372 C3 6C 03
0375 CD 5F 03
0378 C3 69 03
037B CD 64 03
037E C3 69 03
0381 CD 64 03
0384 C3 6C 03
0387
0387 21 00 CC
038A C5
038B 7A
038C E6 0F
038E 06 40
0390 CA 9C 03
0393 23
0394 05
0395 C2 90 03
0398 3D
0399 C2 8E 03
039C 7B
039D E6 7F
039F CA A7 03
03A2 23
03A3 3D
03A4 C3 9F 03
03A7 C1
03A8 C9
03A9
03A9 7A
03AA DE 01
03AC 37
03AD F8
03AE 7A

3800        RNZ
3810 ENDGAME CALL  WAIT    * PAUSE
3820        CALL   CLEAR   * CLEAR SCREEN
3830        LXI    H,ENDM1 * DISPLAY YOUR WIN
3840        LXI    B,ENDB2      MESSAGE
3850        CALL   MSGO
3860        LDA    YSCORE  * AND FINAL SCORE OF
3870        ADI    30H           SERIES
3880        STA    0CD27H
3890        LDA    RSCORE
3900        ADI    30H
3910        STA    0CD29H
3920        JMP    END2
3930 END1   CALL   WAIT          * PAUSE
3940        CALL   CLEAR         * CLEAR SCREEN
3950        LXI    H,ENDM1       * DISPLAY ROBOT'S WIN
3960        LXI    B,ENDB1            MESSAGE
3970        CALL   MSGO
3980        LDA    RSCORE        * AND THE SCORE
3990        ADI    30H
4000        STA    0CD27H
4010        LDA    YSCORE
4020        ADI    30H
4030        STA    0CD29H
4040 END2   LXI    H,ENDM3
4050        LXI    B,ENDB3
4060        CALL   MSGO
4070 END3   CALL   WAIT          * PAUSE
4080        JMP    BEGIN         * NEW SERIES
4090 *
4100 UP     LXI    D,0FFC0H      * MOVEMENT ROUTINES
4110        DAD    D                (8 DIRECTIONS)
4120        RET
4130 DN     LXI    D,40H
4140        DAD    D
4150        RET
4160 LF     DCX    H
4170        DCX    H
4180        RET
4190 RT     INX    H        * NOTE: ONE UNIT TO RIGHT
4200        INX    H                IS ACTUALLY TWO VDM
4210        RET                     BOXES
4220 UR     CALL   UP
4230        JMP    RT
4240 UL     CALL   UP
4250        JMP    LF
4260 LL     CALL   DN
4270        JMP    LF
4280 LR     CALL   DN
4290        JMP    RT
4300 *
4310 PLOT   LXI    H,0CC00H  * ENTER WITH D,E SPECIFY-
4320        PUSH   B                ING COORD; RETRUNS WITH
4330 VERT   MOV    A,D              H,L POINING TO CORRESP-
4340        ANI    0FH              ONDING MEM. LOC ON VDM
4350 LP1    MVI    B,40H
4360 LP2    JZ     HORIZ
4370        INX    H
4380        DCR    B
4390        JNZ    LP2
4400        DCR    A
4410        JNZ    LP1
4420 HORIZ  MOV    A,E
4430        ANI    7FH
4440 LP3    JZ     DONE
4450        INX    H
4460        DCR    A
4470        JMP    LP3
4480 DONE   POP    B
4490        RET
4500 *
4510 LIMIT  MOV    A,D      * RETURNS WITH CARRY RESET
4520        SBI    LOWD      IF D,E POINTING TO VALID
4530        STC              POINT (ON FIELD)
4540        RM
4550        MOV    A,D
```

```
03AF DE 0C          4560          SBI     HID
03B1 37             4570          STC
03B2 F0             4580          RP
03B3 7B             4590          MOV     A,E
03B4 DE 03          4600          SBI     LOWE
03B6 37             4610          STC
03B7 F8             4620          RM
03B8 7B             4630          MOV     A,E
03B9 DE 31          4640          SBI     HIE
03BB 37             4650          STC
03BC F0             4660          RP
03BD 3F             4670          CMC
03BE C9             4680          RET
03BF                4690 *
03BF C5             4700 RND      PUSH    B       * RANDOM NUMBER GEN.
03C0 E5             4710          PUSH    H         RETURNS WITH NUMBER
03C1 21 E4 05       4720          LXI     H,SH+3    IN A REGISTER
03C4 06 08          4730          MVI     B,8
03C6 7E             4740          MOV     A,M       (FROM D.D.J.C.C.&.O.)
03C7 07             4750 RTOP     RLC
03C8 07             4760          RLC
03C9 07             4770          RLC
03CA AE             4780          XRA     M
03CB 17             4790          RAL
03CC 17             4800          RAL
03CD 2D             4810          DCR     L
03CE 2D             4820          DCR     L
03CF 2D             4830          DCR     L
03D0 7E             4840          MOV     A,M
03D1 17             4850          RAL
03D2 77             4860          MOV     M,A
03D3 2C             4870          INR     L
03D4 7E             4880          MOV     A,M
03D5 17             4890          RAL
03D6 77             4900          MOV     M,A
03D7 2C             4910          INR     L
03D8 7E             4920          MOV     A,M
03D9 17             4930          RAL
03DA 77             4940          MOV     M,A
03DB 2C             4950          INR     L
03DC 7E             4960          MOV     A,M
03DD 17             4970          RAL
03DE 77             4980          MOV     M,A
03DF 05             4990          DCR     B
03E0 C2 C7 03       5000          JNZ     RTOP
03E3 E1             5010          POP     H
03E4 C1             5020          POP     B
03E5 C9             5030          RET
03E6                5040 *
03E6 2A EA 05       5050 END      LHLD    ROB1    * ARE ANY ROBOT'S ALIVE
03E9 7E             5060          MOV     A,M
03EA FE 07          5070          CPI     7
03EC C8             5080          RZ
03ED 2A EC 05       5090          LHLD    ROB2
03F0 7E             5100          MOV     A,M
03F1 FE 07          5110          CPI     7
03F3 C8             5120          RZ
03F4 2A EE 05       5130          LHLD    ROB3
03F7 7E             5140          MOV     A,M
03F8 FE 07          5150          CPI     7
03FA C8             5160          RZ
03FB 2A F0 05       5170          LHLD    ROB4
03FE 7E             5180          MOV     A,M
03FF FE 07          5190          CPI     7
0401 C8             5200          RZ
0402 2A F2 05       5210          LHLD    ROB5
0405 7E             5220          MOV     A,M
0406 FE 07          5230          CPI     7
0408 C8             5240          RZ
0409 C3 76 04       5250          JMP     WIN
040C                5260 *
040C 2A E8 05       5270 NEAR     LHLD    YOUP    * ARE AY ROBOT'S NEARBY
040F CD 6C 03       5280          CALL    RT
0412 CD 43 04       5290          CALL    NEAR2
0415 CD 5F 03       5300          CALL    UP
0418 CD 43 04       5310          CALL    NEAR2
041B CD 69 03       5320          CALL    LF
041E CD 43 04       5330          CALL    NEAR2
0421 CD 69 03       5340          CALL    LF
0424 CD 43 04       5350          CALL    NEAR2
0427 CD 64 03       5360          CALL    DN
042A CD 43 04       5370          CALL    NEAR2
042D CD 64 03       5380          CALL    DN
0430 CD 43 04       5390          CALL    NEAR2
0433 CD 6C 03       5400          CALL    RT
0436 CD 43 04       5410          CALL    NEAR2
0439 CD 6C 03       5420          CALL    RT
043C CD 43 04       5430          CALL    NEAR2
043F CD C1 04       5440          CALL    MCLEAR
0442 C9             5450 NEAR3    RET
0443                5460 *
0443 7E             5470 NEAR2    MOV     A,M     * LOOK FOR ROBOT SYMBOL
0444 FE 07          5480          CPI     7
0446 C0             5490          RNZ
0447 CD 5E 04       5500          CALL    MSG1    * INFORM YOU OF ROBOT'S
044A E1             5510          POP     H               PROXIMITY
044B C3 42 04       5520          JMP     NEAR3
044E                5530 *
044E 21 00 CC       5540 CLEAR    LXI     H,0CC00H * CLEAR AND INITIALIZE
0451 36 20          5550          MVI     M,' '    *     THE VDM SCREEN
0453 23             5560          INX     H
0454 7C             5570          MOV     A,H
0455 FE D0          5580          CPI     0D0H
0457 C2 51 04       5590          JNZ     CLEAR+3
045A AF             5600          XRA     A
045B D3 C8          5610          OUT     0C8H
045D C9             5620          RET
045E                5630 *
045E E5             5640 MSG1     PUSH    H       * PUT MESSAGE ON SCREEN
045F C5             5650          PUSH    B
0460 21 CC CF       5660          LXI     H,MS1
0463 01 04 05       5670          LXI     B,M1
0466 CD DC 04       5680          CALL    MSGQ
0469 C1             5690          POP     B
046A E1             5700          POP     H
046B C9             5710          RET
046C                5720 *
046C 21 CC CF       5730 MSG2     LXI     H,MS1   * PUT ANOTHER ONE UP
046F 01 1C 05       5740          LXI     B,M2
0472 CD DC 04       5750          CALL    MSGQ
0475 C9             5760          RET
0476                5770 *
0476 21 CC CF       5780 WIN      LXI     H,MS1   * PUT WIN MESSAGE UP
0479 01 37 05       5790          LXI     B,MWIN
047C CD DC 04       5800          CALL    MSGQ
047F 3A E7 05       5810          LDA     YSCORE  * GIVE YOU A POINT
0482 3C             5820          INR     A
0483 32 E7 05       5830          STA     YSCORE
0486 CD 92 04       5840          CALL    SCORE   * DISPLAY NEW SCORE
0489 CD 01 03       5850          CALL    OVER    * IS GAME OVER?
048C CD F7 02       5860          CALL    WAIT    * PAUSE
048F C3 2C 00       5870          JMP     START   * NEW GAME
0492                5880 *
0492 21 77 CF       5890 SCORE    LXI     H,SRE   * DRAW SCOREBOARD
0495 01 59 05       5900          LXI     B,MSR   BEGINNING AT LOC.
0498 CD D2 04       5910          CALL    MSGO    'SRE'
049B 3A E7 05       5920          LDA     YSCORE  * GET YOUR SCORE
049E C6 30          5930          ADI     30H     * CONVERT TO ASCII
04A0 21 BD CF       5940          LXI     H,SCR
04A3 77             5950          MOV     M,A
04A4 3A E6 05       5960          LDA     RSCORE  * AND THEIR'S
04A7 C6 30          5970          ADI     30H
04A9 11 40 00       5980          LXI     D,40H
04AC 19             5990          DAD     D
04AD 77             6000          MOV     M,A
04AE 21 B8 CF       6010          LXI     H,MSY
04B1 01 63 05       6020          LXI     B,MSR1
04B4 CD D2 04       6030          CALL    MSGO
04B7 21 F5 CF       6040          LXI     H,MT1
04BA 01 68 05       6050          LXI     B,MSR2
04BD CD D2 04       6060          CALL    MSGO
04C0 C9             6070          RET
```

```
04C1                    6080 *
04C1 E5                 6090 MCLEAR PUSH   H       * ERASE MESSAGE
04C2 C5                 6100        PUSH   B
04C3 21 CC CF           6110        LXI    H,MS1
04C6 0E 20              6120        MVI    C,20H
04C8 36 20              6130 MCL1   MVI    M,' '
04CA 23                 6140        INX    H
04CB 0D                 6150        DCR    C
04CC C2 C8 04           6160        JNZ    MCL1
04CF C1                 6170        POP    B
04D0 E1                 6180        POP    H
04D1 C9                 6190        RET
04D2                    6200 *
04D2 0A                 6210 MSGO   LDAX   B       * PUT MESSAGE POINTED TO
04D3 FE FF              6220        CPI    0FFH      BY BC UP ON SCREEN, BEG-
04D5 C8                 6230        RZ               INNING AT LOCTION HL
04D6 77                 6240        MOV    M,A
04D7 03                 6250        INX    B
04D8 23                 6260        INX    H
04D9 C3 D2 04           6270        JMP    MSGO
04DC                    6280 *
04DC 0A                 6290 MSGQ   LDAX   B       * SAME AS MSGO, BUT WITH
04DD FE FF              6300        CPI    0FFH    * INVERTED VIDEO
04DF C8                 6310        RZ
04E0 F6 80              6320        ORI    80H
04E2 77                 6330        MOV    M,A
04E3 03                 6340        INX    B
04E4 23                 6350        INX    H
04E5 C3 DC 04           6360        JMP    MSGQ
04E8                    6370 *
04E8 20 2A 20 2A        6380 MNAME  ASC    # * * THE GAME OF CHASE * * #
     20 54 48 45
     20 47 41 4D
     45 20 4F 46
     20 43 48 41
     53 45 20 2A
     20 2A 20
0503 FF                 6390        DB     0FFH
0504 20 43 41 55        6400 M1     ASC    # CAUTION - ROBOT NEAR! #
     54 49 4F 4E
     20 2D 20 52
     4F 42 4F 54
     20 4E 45 41
     52 21 20
051B FF                 6410        DB     0FFH
051C 20 59 4F 55        6420 M2     ASC    # YOU HAVE BEEN DESTROYED! #
     20 48 41 56
     45 20 42 45
     45 4E 20 44
     45 53 54 52
     4F 59 45 44
     21 20
0536 FF                 6430        DB     0FFH
0537 20 43 4F 4E        6440 MWIN   ASC    # CONGRATULATIONS - YOU HAVE WON! #
     47 52 41 54
     55 4C 41 54
     49 4F 4E 53
     20 2D 20 59
     4F 55 20 48
     41 56 45 20
     57 4F 4E 21
     20
0558 FF                 6450        DB     0FFH
0559 2D 2D 53 43        6460 MSR    ASC    #--SCORE--#
     4F 52 45 2D
     2D
0562 FF                 6470        DB     0FFH
0563 59 4F 55 3A        6480 MSR1   ASC    #YOU:#
0567 FF                 6490        DB     0FFH
0568 52 4F 42 4F        6500 MSR2   ASC    #ROBOTS:#
     54 53 3A
056F FF                 6510        DB     0FFH
0570 54 48 45 20        6520 ENDB1  ASC    #THE ROBOTS HAVE WON  -  : #
     52 4F 42 4F
     54 53 20 48
     41 56 45 20
     57 4F 4E 20
     20 2D 20 20
     3A 20
058A FF                 6530        DB     0FFH
058B 59 4F 55 20        6540 ENDB2  ASC    #YOU HAVE BEATEN THEM -  : #
     48 41 56 45
     20 42 45 41
     54 45 4E 20
     54 48 45 4D
     20 2D 20 20
     3A 20
05A5 FF                 6550        DB     0FFH
05A6 48 49 54 20        6560 ENDB3  ASC    #HIT ANY KEY TO PLAY AGAIN.....#
     41 4E 59 20
     4B 45 59 20
     54 4F 20 50
     4C 41 59 20
     41 47 41 49
     4E 2E 2E 2E
     2E 2E
05C4 FF                 6570        DB     0FFH
05C5                    6580 *
05C5 20 4C 45 56        6590 MDIFF  ASC    # LEVEL OF DIFFICULTY (0-9)?#
     45 4C 20 4F
     46 20 44 49
     46 46 49 43
     55 4C 54 59
     20 28 30 2D
     39 29 3F
05E0 FF                 6600        DB     0FFH
05E1                    6610 *
05E1                    6620 SP     EQU    6
05E1                    6630 PSW    EQU    6
05E1                    6640 STAT   EQU    0
05E1                    6650 DATA   EQU    1
05E1                    6660 RDA    EQU    40H
05E1                    6670 STACK  EQU    0650H
05E1                    6680 RCORN  EQU    0CCF6H
05E1                    6690 DCORN  EQU    0
05E1                    6700 ECORN  EQU    2
05E1                    6710 HEIGHT EQU    0DH
05E1                    6720 WIDTH  EQU    19H
05E1                    6730 LOWD   EQU    DCORN+1
05E1                    6740 LOWE   EQU    ECORN+1
05E1                    6750 HID    EQU    DCORN+HEIGHT-1
05E1                    6760 HIE    EQU    LOWE+WIDTH+WIDTH-4
05E1                    6770 MS1    EQU    0CFCCH
05E1                    6780 MS2    EQU    0CD10H
05E1                    6790 SCR    EQU    0CFBDH
05E1                    6800 SRE    EQU    0CF77H
05E1                    6810 MSY    EQU    0CFB8H
05E1                    6820 MT1    EQU    0CFF5H
05E1                    6830 ENDM1  EQU    0CD10H
05E1                    6840 ENDM3  EQU    0CD90H
05E1 34 12              6850 SH     DW     1234H
05E3 78 56              6860        DW     5678H
05E5                    6870 DFST   DS     1
05E6                    6880 RSCORE DS     1
05E7                    6890 YSCORE DS     1
05E8                    6900 YOUP   DS     2
05EA                    6910 ROB1   DS     2
05EC                    6920 ROB2   DS     2
05EE                    6930 ROB3   DS     2
05F0                    6940 ROB4   DS     2
05F2                    6950 ROB5   DS     2

B1      0058    0570
BEGIN   0000    2080 4080
BOTM    0056
CHEK    0210    2760
CLEAR   044E    0150 0280 3820 3940 5590
DATA    0001    0130 2030 3720
DCORN   0000    0300
DEATH   0285    1930 2470
DECH    027E    2710
DECV    025E    2750
```

```
DFST    05E5    0270 0650 3290 3470 3520
DIFF    001B
DN      0364    1840 2990 3010 4260 4280 5360 5380
DONE    03A7    4440
ECORN   0002    0310
END     03E6    1970
END1    0331    3770
END2    0350    3920
END3    0359
ENDB1   0570    3960
ENDB2   058B    3840
ENDB3   05A6    4050
ENDGA   030F
ENDM1   CD10    3830 3950
ENDM3   CD90    4040
ENTER   0007    2060
FLAS1   0220    2620
FLAS2   0224    2560
FLAS3   022C    2600
FLASH   021E    2490
HEIGH   000D    0410 0580
HID     000C    4560
HIE     0031    4640
HORIZ   039C    4360
INP     01A9    0250 1680 2020
L1      006A    0630
LF      0369    1800 3160 4250 4270 5320 5340
LIMIT   03A9    0770 1400 1570
LL      037B    1820
LOOP1   007D    0780 0850
LOWD    0001    4520
LOWE    0003    4600
LP1     038E    4410
LP2     0390    4390
LP3     039F    4470
LR      0381    1860
LSIDE   0068
M1      0504    5670
M2      051C    5740
MCL1    04C8    6160
MCLEA   04C1    5440
MDIFF   05C5    0230
MNAME   04E8    0190
MS1     CFCC    0220 5660 5730 5780 6110
MS2     CD10    0180
MSG1    045E    5500
MSG2    046C    3190
MSGO    04D2    0200 0240 3850 3970 4060 5910 6030 6060 6270
MSGQ    04DC    5680 5750 5800 6360
MSR     0559    5900
MSR1    0563    6020
MSR2    0568    6050
MSY     CFB8    6010
MT1     CFF5    6040
MWIN    0537    5790
NEAR    040C    1670
NEAR2   0443    5290 5310 5330 5350 5370 5390 5410 5430
NEAR3   0442    5520
OVER    0301    3240 5850
OVER1   0309
PAN1    02D5    3490
PAN2    02E2    3550
PAN3    02C8    3310
PAN4    02ED    3370
PAN5    02B4    3360
PANIC   029D    1880
PANN    02EF    3460 3610
PANOK   02E4    3510 3560
PLOT    0387    0330 0360 0450 0510 0540 0600 0790 1420 1590
PSW     0006    3280 3620 3650 3660
R1      004C    0480
RCORN   CCF6    0870 0890 0910 0930 0950 0970 0990 1010 1030 1050 1070
                1090 1110 1130 1150 1170 1190
RDA     0040    2010 3700
RDEAD   0209    2120 2170 2220 2270 2320
RMOV1   01CE    2130
RMOV2   01DD    2180
RMOV3   01EC    2230
RMOV4   01FB    2280
RMOVE   01BF    1960
RND     03BF    0710 0740 1340 1370 1510 1540 3330 3580
ROB1    05EA    1230 2110 2150 3390 5050
ROB2    05EC    1250 2160 2200 3410 5090
ROB3    05EE    1270 2210 2250 3430 5130
ROB4    05F0    1290 2260 2300 5170
ROB5    05F2    1310 2310 2350 5210
ROBOT   00F1
ROSE    009C
RSCOR   05E6    0120 3200 3220 3750 3890 3980 5960
RSIDE   0048
RT      036C    1720 3150 4230 4290 5280 5400 5420
RTOP    03C7    5000
SCORE   0492    0160 0290 3230 5840
SCR     CFBD    5940
SEED    0074
SEEK    0238    2140 2190 2240 2290 2340
SEL     0115    1220 1240 1260 1280 1300 1410 1450
SH      05E1    4720
SP      0006    0140
SRE     CF77    5890
STACK   0650    0140
START   002C    3260 5870
STAT    0000    2000 3690
SUBH    0271    2690
SUBV    024D    2730
T1      0038    0400
TOP     0036
UL      0375    1780
UP      035F    1760 2930 2960 4220 4240 5300
UR      036F    1740
VERT    038B
WAIT    02F7    3250 3710 3810 3930 4070 5860
WIDTH   0019    0320 0490
WIN     0476    5250
Y1      019B    1900
YMOV1   015B
YMOVE   0158    1330 1490 1980 3670
YOU     0137    1320 1580 1620 3640
YOUP    05E8    1640 1690 1950 2820 3080 5270
YSCOR   05E7    0110 3780 3860 4010 5810 5830 5920
```

```
0000: AF 32 E7 05 32 E6 05 DB 01 31 50 06 CD 4E 04 CD
0010: 92 04 21 10 CD 01 E8 04 CD D2 04 21 CC CF 01 C5
0020: 05 CD D2 04 CD A9 01 E6 0F 32 E5 05 CD 4E 04 CD
0030: 92 04 16 00 1E 02 06 19 CD 87 03 36 58 13 CD 87
0040: 03 36 20 13 05 C2 38 00 06 0D 1D 1D 14 CD 87 03
0050: 36 58 05 C2 4C 00 06 18 1D CD 87 03 36 20 1D CD
0060: 87 03 36 58 05 C2 58 00 06 0D 15 CD 87 03 36 58
0070: 05 C2 6A 00 3A E5 05 07 47 3E 1E 90 47 CD BF 03
0080: E6 0F 57 CD BF 03 E6 7F 5F CD A9 03 DA 7D 00 CD
0090: 87 03 7D E6 FE 6F 36 58 05 C2 7D 00 21 F6 CC 36
00A0: 54 21 FA CC 36 59 21 FE CC 36 55 21 38 CD 36 5C
00B0: 21 3A CD 36 5E 21 3C CD 36 2F 21 76 CD 36 47 21
00C0: 78 CD 36 0D 21 7A CD 36 48 21 7C CD 36 09 21 7E
00D0: CD 36 4A 21 B8 CD 36 2F 21 BA CD 36 0B 21 BC CD
00E0: 36 5C 21 F6 CD 36 42 21 FA CD 36 4E 21 FE CD 36
00F0: 4D CD 15 01 22 EA 05 CD 15 01 22 EC 05 CD 15 01
0100: 22 EE 05 CD 15 01 22 F0 05 CD 15 01 22 F2 05 CD
0110: 37 01 C3 58 01 CD BF 03 E6 0F 57 CD BF 03 E6 FE
0120: 5F CD A9 03 DA 15 01 CD 87 03 7E FE 20 C2 15 01
0130: 36 07 05 C0 C3 58 01 CD BF 03 E6 0F 57 CD BF 03
```

USE A PROM FOR A CHARACTER GENERATOR

It probably has occurred to lots of people that the ubiquitous 1702A PROM might work as a character/graphics generator. Well, it will work, and now that some simple designs for construction of CRT alphanumeric displays have been published, the value of the 1702A as a character generator is much improved, at least for hobbyists.

The hard facts of the situation point to the slowness of the 1702A relative to the writing speed of the CRT, but things are not as bleak as the facts suggest. Slow 1702A's won't work, and even fast 1702A's must be used with care so as to allow them as much time as possible for character accessing. Fortunately, however, published access times are usually worst case or maximums, and actual devices usually turn out to be faster than the published data suggest. If, for instance, the CE line of the PROM is permanently tied low, the associated delay will be eliminated. Also, the 'address-to-data' delays vary depending on which address line is changing state, hence the A0-to-data delay may be much less than the A7-to-data delay.

The 1702A and similar PROMS are organized into 8-bit words, which suggests the use of an 8-by-8 dot matrix because it simplifies the addressing of the PROM. Address lines A0, A1, and A2 are used for row selection; lines A3 through A7 become character select. As such, 32 patterns, each an 8-by-8 matrix, can be stored in a single 1702A.

Suffice it to say that I have used an Advanced Micro Devices AM1702ADC to implement a Cyrillic character set on a 26-by-64 CRT display. The loads on the PROM outputs were slightly critical, as were the operating voltages, but it did work. In fact, 3 out of 4 PROMS worked, which is reasonable since the published specifications indicate that the devices were on the ragged edge of their speed capabilities. The existence of even faster PROMS guarantees their usefulness as character/graphics generators.

David M. Allen
Electronics Consultant

1317 Central Ave.
Kansas City, KS 66102
(913) 371-6136

POLY ASSEMBLED UNITS SEEM TO HAVE PROBLEMS (BUT THEY FIXED 'EM)

Jef Raskin

Your equipment reviewer received a letter from Harry Cohen of the OZNARI project at LaTrobe University in Bundoora, Australia. They seem to be having a bit of trouble with some assembled Polymorphic System's units after very good experiences with Polymorphic's kits. In the reviewer's experience, the Polymorphic kits have worked very well. However the assembled *boards* have been giving a few users some difficulty. A number of calls have come in from knowledgable people who have had trouble with assembled boards.. The complete assempled *systems* have been nearly flawless.

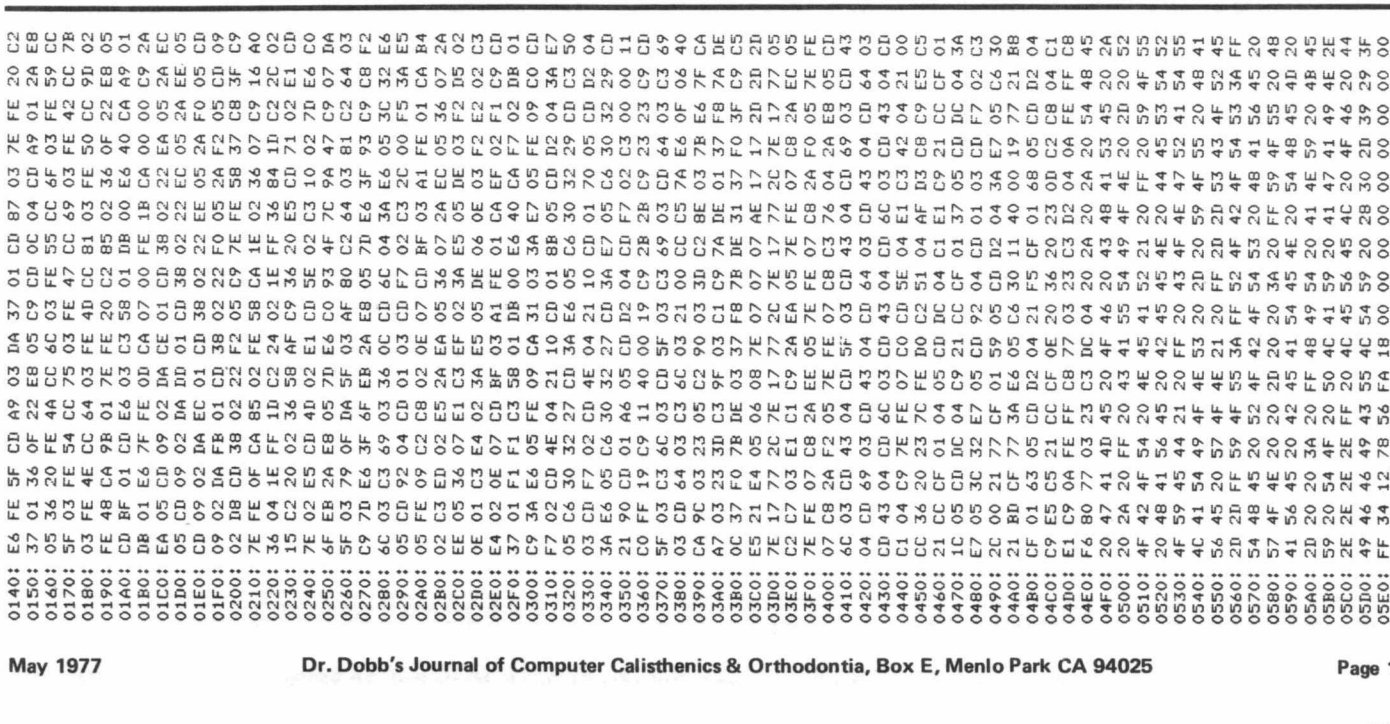
A call from *Dr. Dobb's* to Polymorphic's new home in Santa Barbara got a few reasons why things are the way they are. The complete systems are checked out in normal operation and are burned in. The kits give few problems since they are pretty well designed. But the individual boards have been given only a relatively cursory check. We are assured that this will be corrected. As usual, *Dr. Dobb's* will follow up on this.

The problem with Australian computers was not discovered by Polymorphic Systems, according to a spokesperson, since it was custom-modified for 220V AC as is common there. Also the purchaser had specified that the power cord was to be left *unterminated* so that they could supply their own plug. Murphy's law would have predicted that the custom modification for 220V would be done wrong. It was.

What is Poly doing about it? Putting in a 220V test stand.

Note to our beloved readers: If you have any problems or notable successes with equipment, software, service or documentation, let us know about it. The more the manufacturers realize that problems are going to be published, the more likely they are to prevent them in the first place. Power to the Purchasers!

Note to our beloved manufacturers: Want something reviewed? Give us a call to make the arrangements. Assembled units *do not* have to be *donated* to *Dr. Dobb's*, although we won't fight you if you don't want it back. When we do assemble kits, we can report on the quality of the assembly instructions, but we also expect to keep the fruits of our labors.



SOURCE CODE FOR 8080 PILOT, VERSION 1.1

Copyright © 1977 by John A. Starkweather, Ph.D., 77-Feb

The April 1977 issue of *Dr. Dobb's Journal* carried the complete documentation for this version of Dr. Starkweather's 8080 **PILOT**. The following is the complete assembly listing of his program.

Requests for PILOT-8080 in machine-readable form should be directed to:

PILOT-8080
Computer Technology Branch
Lister Hill Center for Biomedical Communication
National Library of Medicine
8600 Rockville Pike
Bethesda MD 20014
(301) 496-1647

```
                ;
                ;  P I L O T   8080   V E R S I O N   1.1
                ;
0040            ENTRY   EQU     40H             ;NORMAL ENTRY ADDRESS
0100            STKPR   EQU     100H            ;STACK POINTER AND JUMP TABLE
02E0            PSTRT   EQU     2E0H            ;START OF PILOT INTERPRETER
1000            PBUFB   EQU     1000H           ;BEGINNING OF PROGRAM BUFFER
1FFF            PBUFE   EQU     1FFFH           ;END OF PROGRAM BUFFER
0048            LINE    EQU     72              ;MAX INPUT LINE AT STARTUP
FE77            VIDEO   EQU     0FE77H          ;EXTERNAL VIDEO DISPLAY ADDR
E060            MNTR    EQU     0E060H          ;EXTERNAL MONITOR ADDRESS
                ;
0040                    ORG     ENTRY           ;NORMAL STARTUP
0040 C3E002             JMP     START           ;ENTRY SETS NORMAL I/O
                ;
0043 215200             LXI     H,CTV           ;ENTRY TO USE VIDEO
0046 220401             SHLD    CO+1            ;  FOR CONSOLE OUTPUT
0049 220A01             SHLD    LO+1            ;  FOR LIST OUTPUT
004C 220D01             SHLD    PO+1            ;  FOR PUNCH OUTPUT
004F C30F03             JMP     RSTRT           ;AVOID RESET OF JUMP TABLE
0052 C5         CTV:    PUSH    B               ;CALL VIDEO ROUTINE THAT
0053 41                 MOV     B,C             ;  EXPECTS CHAR IN B-REG.
0054 CD77FE             CALL    VIDEO
0057 C1                 POP     B
0058 C9                 RET
                ;
                ; JUMP TABLE FOR I/O ROUTINES
                ;  ONLY TTY ROUTINES ARE PROVIDED INTERNALLY
                ;
0100                    ORG     STKPR
0100 C3CE0D     CI:     JMP     CHI             ;TOP OF STACK
0103 C3A40D     CO:     JMP     CHO             ;CHAR INPUT TO   A REG.
0106 C3CE0D     RI:     JMP     CHI             ;CHAR OUTPUT FROM C REG.
0109 C3A40D     LO:     JMP     CHO             ;READER INPUT TO  A REG.
010C C3A40D     PO:     JMP     CHO             ;LIST OUTPUT FROM C REG.
010F C360E0     EXIT:   JMP     MNTR            ;PUNCH OUTPUT FROM C REG.
0112 C360E0     EDIT:   JMP     MNTR            ;RETURN TO MONITOR
0115 C3D90B     ASCAN:  JMP     BASIC           ;CALL TO EDITOR
                ;                               ;ALTERN INTERPRETER
                ; DATA AREAS
0118 0000       TOPP:   DW      0               ;TOP OF PROGRAM STORAGE
011A 0000       HLSAV:  DW      0               ;TEMPORARY POINTER (HL)
011C 0000       HL1SAV: DW      0               ;TEMPORARY POINTER (HL)
011E 0000       HL2SAV: DW      0               ;TEMPORARY POINTER (HL)
0120 0000       DESAV:  DW      0               ;TEMPORARY POINTER (DE)
0122 0000       LLSAV:  DW      0               ;LAST LINE POINTER
0124 0000       RETSAV: DW      0               ;ZERO LEVEL OF STACK
0126 0000               DW      0               ;LEVEL 1
0128 0000               DW      0               ;LEVEL 2
012A 0000               DW      0               ;LEVEL 3
012C 0000               DW      0               ;LEVEL 4
012E 0000               DW      0               ;LEVEL 5
0130 0000               DW      0               ;LEVEL 6
0132 0000               DW      0               ;LEVEL 7 (TOP)
0134 0000       APTR:   DW      0               ;A SIMT POINTER
0136 0000       EPTR:   DW      0               ;ENTRY POINTER
0138 0000       CPTR:   DW      0               ;CHAR POINTER
013A 0000       IPTR:   DW      0               ;INPUT BUFFER POINTER
013C 0000       MPTR:   DW      0               ;M-SIMT POINTER
013E 0000       MEMTP:  DW      0               ;LAST MEMORY LOCATION
0140 0000       OUTADR: DW      0               ;CO,LO,PO OUTPUT VECTOR
0142 0000       SCANB:  DW      0               ;SCAN BEGINNING ADDR
0144            CHMAX:  DS      1               ;MAX CHARS ACCEPTED
0145            LEVEL:  DS      1               ;CURRENT RETURN LEVEL
0146            LNSKP:  DS      1               ;SKIP FOR DIFF EDITORS
0147            MBRCH:  DS      1               ;M-BREAK CHAR
0148            SCNT:   DS      1               ;STRING COUNT
```

```
0149            TEMP:   DS      1               ;TEMPORARY BINARY VALUE
014A            VARSAV: DS      1               ;VARIABLE SAVED
014B            YNSW:   DS      1               ;YN-SWITCH, 000: NO MATCH
014C            TSAVE:  DS      81              ;T-TEXT AREA
019D            EBUFF:  DS      81              ;ENTRY BUFFER AREA
01EE            MSAVE:  DS      81              ;M LIST AREA
023F            LABSAV: DS      12              ;LABEL SAVE AREA
024B            LASTOP: DS      11              ;LAST OP CODE
0256            NVAR:   DS      53              ;NUMERIC VARIABLE STORAGE
028B            WORD:   DS      81              ;WORD AREA
                ;
                ; START AND TERMINATION OF MAIN PROGRAM
                ;   ORG HERE CAN SET BEGINNING OF ROM AREA
                ;
02E0                    ORG     PSTRT
02E0 310001     START:  LXI     SP,STKPR        ;INITIALIZE STACK POINTER
02E3 21FE1F             LXI     H,PBUFE-1       ;INITIALIZE APTR
02E6 223401             SHLD    APTR
02E9 21F702             LXI     H,IOJMP         ;SET NORMAL I/O VECTORS
02EC 110001             LXI     D,CI
02EF 0E18               MVI     C,24
02F1 CD540C             CALL    BLKTFR
02F4 C30F03             JMP     RSTRT
02F7 C3CE0D     IOJMP:  JMP     CHI             ;COPY OF SID JMP TABLE
02FA C3A40D             JMP     CHO
02FD C3CE0D             JMP     CHI
0300 C3A40D             JMP     CHO
0303 C3A40D             JMP     CHO
0306 C360E0             JMP     MNTR
0309 C360E0             JMP     MNTR
030C C3D90B             JMP     BASIC
                ;
030F 310001     RSTRT:  LXI     SP,STKPR        ;INIT STACK ON RESTART
0312 CD5603             CALL    INIT            ;INITIALIZE THE REST
0315 CD8003             CALL    SCAN            ;SCAN THE BUFFER
0318 C30F03             JMP     RSTRT           ;START OVER
                ;
031B 30323036           DB      '020677',0DH
031F 37370D
                ;
0322 50494C4F           DB      'PILOT-8080, 1.1',0DH
0326 542D3830
032A 38302C20
032E 312E310D
                ;
0332 434F5059           DB      'COPYRIGHT (C) 1977,J.A.STARKWEATHER',0DH
0336 52494748
033A 54202843
033E 29203139
0342 37372C4A
0346 2E412E53
034A 5441524B
034E 57454154
0352 4845520D
                ;
                ;
                ; INITIALIZE DATA FOR NEW PROGRAM
                ;
0356 21720E     INIT:   LXI     H,IBUFF         ;RESET INPUT POINTER
0359 223A01             SHLD    IPTR            ;   TO FRONT OF BUFFER
035C 224201             SHLD    SCANB           ;SET SCAN BEGINNING
035F 21FF1F             LXI     H,PBUFE         ;SET LAST MEMORY LOC
0362 223E01             SHLD    MEMTP
0365 CDEE0B             CALL    NEWN            ;SET A-POINTER
0368 CDFD0B             CALL    INITV           ;INITIALIZE VARIABLES
036B 3601               MVI     M,1             ;     SET STOP
036D 3E48               MVI     A,LINE          ;RESET INMAX TO LINE
036F 324401             STA     CHMAX
0372 AF                 XRA     A               ;ZERO RETURN LEVEL
0373 324501             STA     LEVEL
0376 324601             STA     LNSKP           ;ZERO LN NO. SKIP
0379 210301             LXI     H,CO            ;RESET CONSOLE OUTPUT
037C 224001             SHLD    OUTADR
037F C9                 RET
                ;
                ; SCAN OF INPUT BUFFER
                ;   ENTER:   HL=BUFFER ADDR
                ;   RETURNS: HL=LAST ADDR, B=LAST CHAR (01)
                ;
0380 2A3A01     SCAN:   LHLD    IPTR            ;GET POINTER
0383 7E                 MOV     A,M             ;GET FIRST CHAR
0384 FE01               CPI     1               ;IF END MARKER
0386 C8                 RZ                      ;   THEN RETURN
0387 FE0D               CPI     0DH             ;IF NOT END OF LINE
0389 C29003             JNZ     CKEND           ;   THEN CK FOR SOURCE END
038C 23                 INX     H               ;   ELSE BUMP POINTER
038D C38303             JMP     SCAN+3          ;   AND CONTINUE
0390 CD9D07     CKEND:  CALL    CNTLN           ;HL=EOL, A=BR CHAR
0393 FE01               CPI     1               ;IF END MARKER
0395 C8                 RZ                      ;   THEN RETURN
0396 23                 INX     H               ;HL=START OF NEXT LINE
0397 223A01             SHLD    IPTR            ;SAVE THAT ADDR
039A 2B                 DCX     H               ;HL=BREAK CHAR
039B CDAE07             CALL    BACKUP          ;RESET HL TO CURRENT LINE
039E CD7A04             CALL    SKLN            ;SKIP ANY LN NOS., ETC.
03A1 CD2C07             CALL    GETCH           ;   GET FIRST TEXT CHAR
03A4 FE3A               CPI     ':'             ;IF COLON
03A6 CCBA03             CZ      CONTIN          ;   THEN CONTINUE SAME OP
03A9 CA8003             JZ      SCAN            ;   IF CALLED THEN NEXT SCAN
03AC FE2A               CPI     '*'             ;IF ASTERISK
03AE CC3C07             CZ      GETWD           ;   THEN SKIP LABEL
03B1 CA8303             JZ      SCAN+3          ;IF CALLED THEN RESCAN
03B4 CDCE03             CALL    OPS             ;PROCESS OPERATIONS
03B7 C38003             JMP     SCAN
                ;
03BA 23         CONTIN: INX     H               ;COLON ADDR + 1
03BB 221A01             SHLD    HLSAV           ;SAVE IT
03BE 214B02             LXI     H,LASTOP        ;   ADDR LAST OP CODE
03C1 222201             SHLD    LLSAV           ;SAVE OP CODE ADDR
03C4 063A               MVI     B,':'
03C6 CD3708             CALL    INDX            ;ADDR COLON POS
03C9 CDE603             CALL    OLDOP           ;USE PART OF OPS
03CC AF                 XRA     A               ;   SET RETURN FLAG
03CD C9                 RET
                ;
                ; OP CODES-- INTERPRET OPERATION
                ;   ENTER: HL = FIRST NON-BLANK CHAR IN LINE
                ;   RETURNS: RETURN (ZERO) FLAG SET
                ;
03CE 222201     OPS:    SHLD    LLSAV           ;SAVE OP CODE ADDR
03D1 CD8E07             CALL    SAVOP           ;SAVE OP CODE
03D4 2A2201             LHLD    LLSAV           ;ADDR OP CODE
03D7 063A               MVI     B,':'           ;   LOOK FOR COLON
03D9 CD3708             CALL    INDX            ;IF NOT FOUND
03DC 79                 MOV     A,C
03DD B7                 ORA     A
03DE CAFC03             JZ      ALTSC           ;   THEN TRY ALTERN SCAN
03E1 23                 INX     H               ;COLON ADDR + 1
03E2 221A01             SHLD    HLSAV           ;SAVE IT
03E5 2B                 DCX     H               ;ADDR POS OF COLON
03E6 2B         OLDOP:  DCX     H               ;ADDR POS BEFORE COLON
03E7 CD8404             CALL    YNCHK           ;IF YN-SW OFF(Y) OR ON(N)
```

```
03EA B7              ORA   A             ;   (A=000)
03EB C8              RZ                  ;  THEN RETURN
03EC CDA704          CALL  VARCHK        ;IF VARIABLE PRESENT < 1
03EF C8              RZ                  ;  THEN RETURN
03F0 2A2201          LHLD  LLSAV         ;ADDR OP CODE
03F3 CD7107          CALL  GETCTL        ;GET THE CONTROL WORD
03F6 CDE604          CALL  CTLMCH        ;CALL SPECIFIC CONTROL
03F9 FE01            CPI   1             ;IF CONTROL FOUND
03FB C0              RNZ                 ;  THEN RETURN
03FC 2A2201  ALTSC:  LHLD  LLSAV         ;ADDR FIRST CHAR
03FF CD1501          CALL  ASCAN         ;TRY ALTERN SCAN
0402 C8              RZ                  ;IF OK, THEN RETURN
0403 2A2201          LHLD  LLSAV         ;ELSE ADDR FIRST CHAR
0406 221A01          SHLD  HLSAV         ;SET POINTER
0409 CD7908          CALL  TOP           ;DISPLAY TEXT
040C C9              RET
                ;
                ; TEXT CHECK FOR PRESENCE OF LINE FEEDS,
                ;   LINE COUNTS, OR LINE NUMBERS.
                ; SETS LNSKP TO NO. OF CHARS TO SKIP BEFORE TEXT
                ;
040D 210010  TXTCK:  LXI   H,PBUFF       ;ADDR PROGRAM TEXT
0410 CD9D07          CALL  CNTLN         ;ADDR CR
0413 23              INX   H
0414 7E              MOV   A,M           ;GET NEXT CHAR
0415 FE0A            CPI   0AH           ;IF NOT LF
0417 C22C04          JNZ   CKLC          ;  THEN CK FOR LINE COUNT
041A CD9D07          CALL  CNTLN         ;ELSE CK ANOTHER LINE
041D 23              INX   H
041E 7E              MOV   A,M
041F FE0A            CPI   0AH           ;IF NOT LF
0421 C22C04          JNZ   CKLC          ;  THEN CK FOR LINE COUNT
0424 3A4601          LDA   LNSKP         ;ELSE ADD 1 TO LNSKP
0427 C601            ADI   1
0429 324601          STA   LNSKP
042C 210010  CKLC:   LXI   H,PBUFF       ;CK FOR LINE COUNT
042F CD9D07          CALL  CNTLN         ;ADDR CR
0432 23              INX   H             ;  NEXT CHAR
0433 CD7A04          CALL  SKLN          ;SKIP ANY LF
0436 5E              MOV   E,M           ;GET POSSIBLE LINE COUNT
0437 1600            MVI   D,0
0439 1B              DCX   D             ;DECR IT
043A 19              DAD   D             ;ADDR LINE END
043B 7E              MOV   A,M           ;GET CHAR
043C FE0D            CPI   0DH           ;IF NOT CR
043E C25604          JNZ   CKLN          ;  THEN CK FOR LINE NO.
0441 23              INX   H             ;ELSE CK ANOTHER LINE
0442 CD7A04          CALL  SKLN
0445 5E              MOV   E,M
0446 1B              DCX   D
0447 19              DAD   D
0448 7E              MOV   A,M
0449 FE0D            CPI   0DH           ;IF NOT CR
044B C25604          JNZ   CKLN          ;  THEN CK FOR LINE NOS.
044E 3A4601          LDA   LNSKP         ;ELSE ADD 1 TO LNSKP
0451 C601            ADI   1
0453 324601          STA   LNSKP
0456 210110  CKLN:   LXI   H,PBUFF+1     ;NOW LOOK FOR LN NOS.
0459 CD9D07          CALL  CNTLN         ;ADDR CR
045C 23              INX   H             ;  NEXT CHAR
045D CD7A04          CALL  SKLN          ;SKIP LF OR LN CNT
0460 7E              MOV   A,M           ;GET CHAR AFTER LN CT
0461 CD6E04          CALL  NUM           ;IF NOT ASCII NUMBER
0464 C0              RNZ                 ;  THEN QUIT
0465 3A4601          LDA   LNSKP         ;ELSE ADD 4 TO LNSKP
0468 C604            ADI   4
046A 324601          STA   LNSKP
046D C9              RET
                ;
046E FE30    NUM:    CPI   '0'           ;CHECK FOR ASCII NUMBER
0470 F8              RM                  ;TOO LOW
0471 FE3A            CPI   '9'+1
0473 FA7804          JM    YNUM
0476 B4              ORA   H             ;TOO HIGH
0477 C9              RET
0478 AF      YNUM:   XRA   A             ;OK
0479 C9              RET
                ;
                ; SKIP LINE NUMBER AND LINE COUNT
                ;    BASED ON VALUE OF LNSKP
                ;
047A 3A4601  SKLN:   LDA   LNSKP         ;GET SKIP COUNT
047D B7              ORA   A             ;IF ZERO
047E C8              RZ                  ;  THEN RETURN
047F 23              INX   H             ;SKIP A CHARACTER
0480 3D              DCR   A             ;DECR COUNT
0481 C37D04          JMP   SKLN+3        ;MORE
                ;
                ;    CHECK FOR Y OR N CONDITIONS
                ; Y AND N FOLLOWING OP CODE
                ;      ACT AS A SWITCH ALONG WITH YN-SWITCH
                ;   ENTER:   HL = ADDR OF COLON
                ;   RETURNS: A = 000 IF NO ACTION REQUIRED
                ;                ELSE A = CHAR BEFORE COLON
                ;            HL = ADDR OF LAST CHAR BEFORE COLON
                ;
0484 CD3407  YNCHK:  CALL  GETLCH        ;GET LAST CHARACTER
0487 FE59            CPI   'Y'           ;IF Y
0489 CA9304          JZ    YCHK
048C FE4E            CPI   'N'           ;IF N
048E CA9C04          JZ    NCHK
0491 B7              ORA   A             ;  ELSE SET SWITCH ON
0492 C9              RET                 ;   AND RETURN WITH CHAR
0493 3A4B01  YCHK:   LDA   YNSW          ;IF YN-SWITCH
0496 B7              ORA   A             ;     SHOWS MATCH
0497 CAA504          JZ    DONT          ;   THEN QUIT
049A B4              ORA   H             ;  ELSE SET SWITCH ON
049B C9              RET                 ;   AND RETURN
049C 3A4B01  NCHK:   LDA   YNSW
049F B7              ORA   A             ;     SHOWS NO MATCH
04A0 C2A504          JNZ   DONT          ;   THEN QUIT
04A3 B4              ORA   H             ;  ELSE SET SWITCH ON
04A4 C9              RET                 ;   AND RETURN
04A5 AF      DONT:   XRA   A             ;  SET SWITCH OFF
04A6 C9              RET                 ;   AND RETURN
                ;
                ;     CHECK FOR NUMERIC VARIABLE CONDITIONS
                ; VARIABLE IN PARENTHESES AFTER OP CODE
                ;   CAUSES EXECUTION IF VALUE +1 OR MORE
                ;    ENTER:   A = LAST CHAR BEFORE COLON
                ;    RETURNS: ZERO FLAG OFF IF NO ACTION REQUIRED
                ;
04A7 FE29    VARCHK: CPI   ')'           ;IF VARIABLE PRESENT
04A9 CAAE04          JZ    VCHK          ;   THEN CHECK IT
04AC B4              ORA   H             ;  ELSE SET SWITCH ON
04AD C9              RET                 ;   AND RETURN
04AE 2B      VCHK:   DCX   H             ;DECR POINTER
04AF 2B              DCX   H             ;   TWICE
04B0 7E              MOV   A,M           ;    GET CHAR
04B1 FE28            CPI   '('           ;IF PAREN NOT PRESENT
04B3 C2CB04          JNZ   BADFRM        ;   THEN COMPLAIN
04B6 23              INX   H             ;BUMP POINTER
```

```
04B7 46             MOV   B,M          ;   SAVE CHAR IN B
04B8 CDD804         CALL  VARMCH       ;LOOK IT UP
04BB FE01           CPI   1            ;IF END MARKER
04BD CACB04         JZ    BADFRM       ;  THEN COMPLAIN
04C0 23             INX   H            ;  ELSE POINT AT VALUE
04C1 7E             MOV   A,M          ;   GET VALUE
04C2 FE01           CPI   01
04C4 FAC904         JM    VOFF         ;  THEN QUIT
04C7 B4             ORA   H            ; ELSE SET SWITCH ON
04C8 C9             RET                ;  AND RETURN
04C9 AF       VOFF: XRA   A            ; SET SWITCH OFF
04CA C9             RET                ;  AND RETURN
                    ;
04CB 2A2201   BADFRM: LHLD  LLSAV        ;SHOW THE LINE
04CE CD7C08         CALL  TOP+3
04D1 21080E         LXI   H,EXPMSG
04D4 CDDE0D         CALL  ERROR
04D7 C9             RET
                    ;
                    ; VARIABLE MATCH - LOOKUP OF VARIABLE NAME/VALUE LIST
                    ;    ENTER:   VARIABLE NAME CHAR IN B REGISTER
                    ;    RETURNS: HL = ADDR OF MATCHED NAME
                    ;             IF VAR NOT IN LIST THEN A = 01
                    ;
04D8 215602   VARMCH: LXI   H,NVAR
04DB 7E             MOV   A,M
04DC FE01           CPI   1            ;IF LIST END
04DE C8             RZ                 ;  THEN RETURN
04DF B8             CMP   B            ;IF MATCH
04E0 C8             RZ                 ;  THEN RETURN
04E1 23             INX   H            ;  ELSE LOOK AGAIN
04E2 23             INX   H
04E3 C3DB04         JMP   VARMCH+3
                    ;
                    ; CONTROL MATCH- CALLS SPECIFIC CONTROL OPERATIONS
                    ;   ENTER:   'WORD':CONTROL WORD
                    ;   RETURNS: IF WORD NOT IN LIST, THEN 01 RETURNED
                    ;            HL: START OF NEXT WORD
                    ;
04E6 111105   CTLMCH: LXI   D,CTLST      ;  DE=CONTROL LIST ADDR
04E9 CDFC04         CALL  LSTMCH       ;LOOK FOR WORD
04EC FE01           CPI   1            ;IF NOT FOUND
04EE C8             RZ                 ;  THEN RETURN
04EF EB             XCHG
04F0 23             INX   H
04F1 11FA04         LXI   D,RTRN       ;PUT RETURN ON STACK
04F4 D5             PUSH  D
04F5 5E             MOV   E,M
04F6 23             INX   H
04F7 56             MOV   D,M
04F8 D5             PUSH  D            ;CALL ADDR ON STACK
04F9 C9             RET
04FA AF       RTRN: XRA   A
04FB C9             RET
                    ;
                    ; LIST MATCH - LOOKUP OF WORD/ADDRESS LIST
                    ;   ENTER:   'WORD' = WORD TO BE FOUND
                    ;            DE = ADDR OF BEGINNING OF LIST
                    ;   RETURNS: DE = ADDR OF POINTER (L BYTE)
                    ;            IF WORD NOT IN LIST THEN A = 01
                    ;
04FC 218B02   LSTMCH: LXI   H,WORD       ;  HL=INPUT WORD
04FF CDFB06         CALL  CMPR         ;COMPARE WORD WITH LIST
0502 B7             ORA   A            ; IF MATCH
0503 C0             RNZ                ;  THEN RETURN
0504 23             INX   H            ;ELSE HL = DE
0505 EB             XCHG
0506 23             INX   H            ;AND INCR HL TO
0507 23             INX   H            ;NEXT LIST ADDR
0508 23             INX   H
0509 7E             MOV   A,M          ;   GET NEXT LIST CHAR
050A FE01           CPI   1            ;IF END MARKER
050C C8             RZ                 ;  THEN RETURN
050D EB             XCHG               ;ELSE RESET DE TO NEXT ITEM
050E C3FC04         JMP   LSTMCH       ;  AND TRY IT
                    ;
                    ; CONTROL LIST - OP CODES AND KEYWORDS
                    ;
0511 540D     CTLST:  DB    'T',0DH
0513 7908             DW    TOP
0515 410D             DB    'A',0DH
0517 5809             DW    AOP
0519 4D0D             DB    'M',0DH
051B 260A             DW    MOP
051D 4D430D           DB    'MC',0DH
0520 1E0A             DW    MC
0522 4A0D             DB    'J',0DH
0524 CB05             DW    JOP
0526 520D             DB    'R',0DH
0528 DB0B             DW    ROP
052A 430D             DB    'C',0DH
052C 090B             DW    COP
052E 550D             DB    'U',0DH
0530 0806             DW    UOP
0532 450D             DB    'E',0DH
0534 DC0B             DW    EOP
0536 590D             DB    'Y',0DH
0538 7908             DW    TOP
053A 4E0D             DB    'N',0DH
053C 7908             DW    TOP
053E 4C4F4144         DB    'LOAD',0DH          ;LOAD NEW PROGRAM
0542 0D
0543 E70B             DW    LOAD
0545 494E4D41         DB    'INMAX',0DH         ;LIMITS CHARS ACCEPTED
0549 580D
054B 8905             DW    INMAX
054D 4E455724         DB    'NEW$',0DH          ;ERASE $TEXT
0551 0D
0552 EE0B             DW    NEWN
0554 44500D           DB    'DP',0DH            ;DISPLAY PROGRAM
0557 7A0D             DW    DPRG
0559 5052494E         DB    'PRINT',0DH         ;PRINT PROGRAM
055D 540D
055F 840D             DW    LPRG
0561 53415645         DB    'SAVE',0DH          ;SAVE PROGRAM
0565 0D
0566 940D             DW    SPRG
0568 4945500D         DB    'IEP',0DH           ;INTERPRET EXIST PROG
056C 7C05             DW    IEP
056E 4259450D         DB    'BYE',0DH
0572 0F01             DW    EXIT
0574 45444954         DB    'EDIT',0DH
0578 0D
0579 1201             DW    EDIT
057B 01               DB    1
                    ;
                    ;COMMON DATAPOINT PILOT CODES NOT IN OPERATION
                    ;       DB    'CA',0DH
                    ;       DW    CURSR
                    ;       DB    'CE',0DH
                    ;       DW    CLRE
                    ;       DB    'CL',0DH
```

```
                ;          DW     CLRL
                ;          DB     'CH',0DH
                ;          DW     CLRH
                ;          DB     'RL',0DH
                ;          DW     ROLL
                ;          DB     'WA',0DH
                ;          DW     WAIT
                ;
                ;CURSR:    RET
                ;CLRE:     RET
                ;CLRL:     RET
                ;CLRH:     RET
                ;ROLL:     RET
                ;WAIT:     RET
                ;
                ; INTERPRET EXISTING PROGRAM
                ;    STARTS SCAN OF PROGRAM BUFFER
                ;
057C 210010     IEP:    LXI   H,PBUFF
057F 223A01             SHLD  IPTR          ;SET POINTER
0582 224201             SHLD  SCANB         ;  AND SCAN BEGINNING
0585 CD0D04             CALL  TXTCK         ;CK FOR CHARS TO SKIP
0588 C9                 RET
                ;
0589 CD9A05     INMAX:  CALL  NMCTL         ;E = NUMBER CONTROL
058C 7B                 MOV   A,E
058D FE49               CPI   73            ;LIMIT TO 72
058F FA9505             JM    INMX2
0592 3E48               MVI   A,72
0594 5F                 MOV   E,A
0595 214401     INMX2:  LXI   H,CHMAX       ;  SET INPUT CHAR MAX
0598 73                 MOV   M,E
0599 C9                 RET
                ;
                ; CHAR TO BINARY CONVERSION FOR CONTROL ARGUMENTS
                ; NUMBER CONTROL - FINDS 1 OR 2 DIGIT NUMBER OR NAME
                ;  OF VARIABLE IN NEXT WORD. NEGATIVE VALUES SET TO ZERO
                ;    ENTER:   HLSAV = EXPRESSION ADDRESS
                ;    RETURNS: E = BINARY VERSION OF THE NUMBER
                ;             A = 0DH IF ALREADY AT END OF LINE
                ;             HLSAV = BR CHAR   ADDR
                ;
059A 2A1A01     NMCTL:  LHLD  HLSAV         ;EXPRESSION ADDR
059D CD2C07             CALL  GETCH         ;GET CHAR
05A0 FE0D               CPI   0DH           ; IF CR
05A2 C8                 RZ                  ;  THEN RETURN
05A3 CD3C07             CALL  GETWD         ;GET NEXT WORD
05A6 2B                 DCX   H             ;BACK UP TO BR CHAR
05A7 221A01             SHLD  HLSAV         ;SAVE POINTER
05AA 218B02             LXI   H,WORD        ;    IN 'WORD'
05AD CD6008             CALL  LETTER        ;IF NOT LETTER
05B0 C2C205             JNZ   CVNUM         ;  THEN CONVERT A NUMBER
05B3 46                 MOV   B,M           ;   ELSE SAVE CHAR IN B
05B4 CDD804             CALL  VARMCH        ;LOOK IT UP
05B7 FE01               CPI   1             ;IF END MARKER
05B9 CCCB04             CZ    BADFRM        ;  THEN COMPLAIN
05BC C8                 RZ                  ;  AND RETURN
05BD 23                 INX   H             ;ELSE POINT AT VALUE
05BE 5E                 MOV   E,M           ;   PUT VALUE IN E
05BF C3C505             JMP   CVNUM+3       ;  AND QUIT
05C2 CD3106     CVNUM:  CALL  GETNM         ;CONVERT NUMBER
05C5 7B                 MOV   A,E           ;      GET VALUE
05C6 B7                 ORA   A
05C7 F0                 RP                  ;RETURN IF POSITIVE
05C8 1E00               MVI   E,0           ;  ELSE SET TO ZERO
05CA C9                 RET
                ;
                ; JUMP TO LABEL NAME
                ;   ENTER:   HLSAV = EXPRESSION FIELD
                ;   RETURNS: HL RESET OR MESSAGE
                ;
05CB 2A1A01     JOP:    LHLD  HLSAV         ;ADDR EXPRESSION
05CE CD2C07             CALL  GETCH         ;GET FIRST CHAR
05D1 FE2A               CPI   '*'           ;IF *
05D3 CAE705             JZ    JOP2          ;  THEN MOVE WORD
05D6 3E2A               MVI   A,'*'
05D8 328B02             STA   WORD          ;ELSE ADD *
05DB 118C02             LXI   D,WORD+1      ;  THEN MOVE WORD
05DE 2A1A01             LHLD  HLSAV
05E1 CD5407             CALL  WDTFR
05E4 C3EA05             JMP   JOP2+3        ;AND CONTINUE
05E7 CD3C07     JOP2:   CALL  GETWD         ;GET NEXT WORD
05EA 2A4201             LHLD  SCANB         ;  START OF SCAN AREA
05ED CD9D06             CALL  LOOKL         ;LOOK FOR IT
05F0 FE01               CPI   1             ;IF LABEL NOT FOUND
05F2 CCFB05             CZ    NTFND         ;  THEN COMPLAIN
05F5 C8                 RZ                  ;  AND RETURN
05F6 23                 INX   H
05F7 223A01             SHLD  IPTR          ;NEW SCAN POSITION
05FA C9                 RET                 ;RESTART SCAN
                ;
                ;
05FB 218B02     NTFND:  LXI   H,WORD        ;  SHOW THE LABEL
05FE CD490D             CALL  DSPLY
0601 21E30D             LXI   H,BLMSG
0604 CDDE0D             CALL  ERROR
0607 C9                 RET
                ;
0608 CD130C     UOP:    CALL  SAVRET        ;SAVE RETURN POINTER
060B C3CB05             JMP   JOP
                ;
                ; SET A BLOCK OF LENGTH C TO CHAR B
                ;
060E 70         BLKSET: MOV   M,B           ;   STORE ONE CHAR
060F 23                 INX   H             ;BUMP ADDR
0610 79                 MOV   A,C           ;   DECR COUNT
0611 D601               SUI   1
0613 4F                 MOV   C,A           ;      IF COUNT NOT ZERO
0614 C20E06             JNZ   BLKSET        ;  THEN STORE ANOTHER
0617 C9                 RET
                ;
                ; BLANK THE INPUT BUFFER
                ;
0618 2A3401     BLKBF:  LHLD  APTR          ;DE=TOP OF BUFFER
061B EB                 XCHG
061C 210010             LXI   H,PBUFF       ;HL=BOTTOM OF BUFFER
061F 0620               MVI   B,' '
0621 70         BLKB2:  MOV   M,B
0622 23                 INX   H
0623 CD2A06             CALL  ADRCMP
0626 C22106             JNZ   BLKB2
0629 C9                 RET
                ;
                ; ADDRESS COMPARISON - COMPARES HL + DE
                ;    RETURNS: ZERO AND SIGN FLAGS SET AS THOUGH
                ;             A CONTAINED HL AND DE WAS COMPARED
                ;    CALLED BY BLKBF
                ;
062A 7C         ADRCMP: MOV   A,H           ;      GET H
062B BA                 CMP   D             ;COMPARE D
062C F8                 RM                  ;IF D > H THEN RETURN
062D C0                 RNZ                 ;IF D NOT = H THEN RETURN
```

```
062E 7D             MOV   A,L           ;       GET L
062F BB             CMP   E             ;COMPARE E
0630 C9             RET                 ;  AND RETURN
                ;
                ;    CHARACTER TO BINARY CONVERSION
                ; GET A DECIMAL NUMBER-- UP TO 99
                ;   ENTER:   HL= CHAR ADDR OF ONE OR TWO DIGIT NUMBER
                ;   RETURNS: BINARY NUMBER IN E
                ;      IF INPUT NOT NUMERIC, THEN E = 0
                ;
0631 1E00       GETNM:  MVI   E,0           ;  INIT. OUTPUT VALUE
0633 23             INX   H             ;LOOK AT NEXT CHAR
0634 CDB907         CALL  BRCHAR        ;IF BREAK CHAR
0637 CA6106         JZ    SDIG          ;  THEN SINGLE DIGIT
063A FE2B           CPI   '+'
063C CA6106         JZ    SDIG
063F FE2D           CPI   '-'
0641 CA6106         JZ    SDIG
0644 2B             DCX   H             ;ELSE BACK UP
0645 7E             MOV   A,M           ;   GET FIRST CHAR
0646 FE30           CPI   '0'           ;LIMIT RANGE
0648 F8             RM                  ;  TO NUMERALS
0649 FE3A           CPI   '9'+1
064B F0             RP
064C D630           SUI   '0'           ;REMOVE ASCII BIAS
064E 87             ADD   A             ;MULT. BY 10
064F 5F             MOV   E,A           ;         E=A*2
0650 87             ADD   A             ;  A*4
0651 87             ADD   A             ;  A*8
0652 83             ADD   E             ;  A+E=A*10
0653 5F             MOV   E,A           ;       SAVE IT
0654 23             INX   H             ;HL=HL+1
0655 7E         UNITS:  MOV   A,M       ;   GET SECOND CHAR
0656 FE30           CPI   '0'           ;LIMIT RANGE
0658 F8             RM                  ;  TO NUMERALS
0659 FE3A           CPI   '9'+1
065B F0             RP
065C D630           SUI   '0'           ;REMOVE ASCII BIAS
065E 83             ADD   E             ;ADD NEW DIGIT
065F 5F             MOV   E,A           ;         TO E
0660 C9             RET
0661 2B         SDIG:   DCX   H         ;BACK UP POINTER
0662 C35506         JMP   UNITS         ;CONVERT UNITS POSITION
                ;
                ;    BINARY TO CHARACTER CONVERSION
                ; PUT BINARY NUMBER IN DECIMAL CHARS -99 TO +99
                ;    ENTER:   BINARY NUMBER IN E
                ;             HL = CHARACTER AREA
                ;    RETURNS: HL = ADDR OF ODH AFTER RIGHT DIGIT
                ;
0665 0E00       PUTNM:  MVI   C,0           ;  INITIALIZE C
0667 7B             MOV   A,E           ;       GET BINARY NUMBER
0668 B7             ORA   A             ; IF NEGATIVE
0669 FC8E06         CM    NEG           ;  THEN SHOW MINUS SIGN
066C FE0A           CPI   10            ;IF < 10
066E FA7B06         JM    FRMCH         ;  THEN FORM CHAR
0671 D60A           SUI   10            ;  ELSE SUBTR 10
0673 5F             MOV   E,A           ;         SAVE IN E
0674 79             MOV   A,C           ;   INCR TENS COUNT
0675 C601           ADI   1
0677 4F             MOV   C,A
0678 C36706         JMP   PUTNM+2       ;  AND LOOP
067B 79         FRMCH:  MOV   A,C       ;   GET TENS COUNT
067C C630           ADI   '0'           ;ADD ASCII BIAS
067E FE30           CPI   '0'           ;IF CHAR IS 0
0680 CA8506         JZ    FRMU          ;  THEN FORM UNITS
0683 77             MOV   M,A           ;   STORE THE CHAR
0684 23             INX   H             ;BUMP CHAR ADDR
0685 7B         FRMU:   MOV   A,E       ;       GET THE UNITS
0686 C630           ADI   '0'           ;ADD ASCII BIAS
0688 77             MOV   M,A           ;   STORE THE CHAR
0689 23             INX   H             ;BUMP POINTER
068A 3E0D           MVI   A,0DH         ;   STORE EOL
068C 77             MOV   M,A
068D C9             RET
                ;
068E 3E2D       NEG:    MVI   A,'-'     ;  STORE MINUS SIGN
0690 77             MOV   M,A
0691 23             INX   H             ;BUMP CHAR ADDRESS
0692 AF             XRA   A             ; MAKE BINARY POSITIVE
0693 93             SUB   E
0694 5F             MOV   E,A           ;       SAVE IN E
0695 C9             RET
                ;
                ; LOOK FOR *LABEL OR $NAME OF STRING VARIABLE
                ;   (LOOKL OR LOOKS)
                ;   ENTER: 'WORD'=LABEL TO BE FOUND, HL=SCAN ADDR
                ;   RETURNS: HL = ADDR OF BLANK AFTER LABEL
                ;            IF LABEL NOT FOUND THEN A = 01
                ;
0696 AF         LOOKS:  XRA   A             ;SET TEMP FOR NO SKIPS
0697 324901         STA   TEMP
069A C3A406         JMP   LOOK
069D B4         LOOKL:  ORA   H             ;SET TEMP ON FOR SKIPS
069E 324901         STA   TEMP
06A1 CD7A04         CALL  SKLN          ;SKIP ANY LINE NOS.
06A4 CD2C07     LOOK:   CALL  GETCH         ;NEXT CHAR
06A7 FE01           CPI   1             ;IF DATA END
06A9 C8             RZ                  ;  THEN RETURN
06AA FE2A           CPI   '*'           ;IF *
06AC CAC206         JZ    CHK           ;  THEN CHECK THE LABEL
06AF FE24           CPI   '$'           ;IF $
06B1 CAC206         JZ    CHK           ;  THEN CHECK STRING NAME
06B4 CD9D07         CALL  CNTLN         ;ELSE GO TO NEXT LINE
06B7 23             INX   H
06B8 3A4901         LDA   TEMP          ;IF TEMP=0
06BB B7             ORA   A
06BC CAA406         JZ    LOOK          ;  THEN NO SKIPS
06BF C3A106         JMP   LOOK-3        ;ELSE SKIP LN NOS.
06C2 221A01     CHK:    SHLD  HLSAV     ;SAVE POINTER
06C5 CD4607         CALL  CNTWD         ;C = WORD LENGTH
06C8 79             MOV   A,C
06C9 FE0D           CPI   13            ;LIMIT TO 12 CHARS
06CB FAD006         JM    MVLAB
06CE 0E0C           MVI   C,12
06D0 2A1A01     MVLAB:  LHLD  HLSAV     ;RETRIEVE POINTER
06D3 113F02         LXI   D,LABSAV      ;  DESTIN ADDR
06D6 CD540C         CALL  BLKTFR        ;MOV   A,BSAV=LABEL
06D9 EB             XCHG                ;HL:DESTIN BR CHAR+1
06DA 2B             DCX   H             ;   DESTIN BR CHAR
06DB 3E0D           MVI   A,0DH         ;   REPLACE WITH ODH
06DD 77             MOV   M,A
06DE 213F02         LXI   H,LABSAV
06E1 11BB02         LXI   D,WORD        ;    WORD ADDR
06E4 CDFB06         CALL  CMPR          ;COMPARE THEM
06E7 B7             ORA   A             ; LOOK AT A REGISTER
06E8 C2F406         JNZ   LFND          ;IF MATCH THEN LABEL FOUND
06EB 2A1A01         LHLD  HLSAV         ;ELSE RETRIEVE POINTER
06EE CD4607         CALL  CNTWD         ;SKIP LABEL
06F1 C3A406         JMP   LOOK          ;  AND CONTINUE
06F4 2A1A01     LFND:   LHLD  HLSAV     ;RETRIEVE POINTER
06F7 CD4607         CALL  CNTWD         ;SKIP LABEL
```

```
06FA C9             RET                 ;  AND RETURN
                    ;
                    ; COMPARE STRINGS X AND Y
                    ;   ENTER:   HL= X ITEM ADDR, DE= Y ITEM ADDR
                    ;            BOTH ITEMS TERMINATE IN ODH
                    ;   RETURNS: A=0 FOR NO MATCH,
                    ;            HL AND DE AT ODH ADDRESS
                    ;
06FB 7E     CMPR:   MOV   A,M           ;   GET X CHAR
06FC FE0D           CPI   0DH           ; IF END OF LINE
06FE CA1307         JZ    XEND          ;  THEN END OF X ITEM
0701 4F             MOV   C,A           ;       SAVE X CHAR IN C
0702 23             INX   H             ;ADDR Y ITEM
0703 EB             XCHG
0704 7E             MOV   A,M           ;   GET Y CHAR
0705 FE0D           CPI   0DH           ; IF END OF LINE
0707 CA2307         JZ    YENDB         ;  THEN END OF Y ITEM
070A B9             CMP   C             ;IF A(Y) NOT= C(X)
070B C22007         JNZ   NOMCH         ;THEN NO MATCH
070E 23             INX   H             ;ADDR NEXT X ITEM
070F EB             XCHG
0710 C3FB06         JMP   CMPR          ;START OVER
0713 EB     XEND:   XCHG                ;ADDR Y ITEM
0714 7E             MOV   A,M           ;   GET Y CHAR
0715 FE0D           CPI   0DH           ; IF END OF LINE
0717 CA2907         JZ    MCH           ;  THEN MATCH FOUND
071A CD4607         CALL  CNTWD         ;ADDR Y BR CHAR
071D EB             XCHG                ;SET DE
071E AF             XRA   A             ;  NOMATCH
071F C9             RET
0720 CD4607 NOMCH:  CALL  CNTWD         ;ADDR Y BR CHAR
0723 EB     YENDB:  XCHG                ;  SET DE
0724 CD4607         CALL  CNTWD         ;ADDR X BR CHAR
0727 AF             XRA   A             ; NO MATCH
0728 C9             RET
0729 EB     MCH:    XCHG                ;SET DE & HL
072A B4             ORA   H             ; MATCH
072B C9             RET
                    ;
                    ; GET CHARACTER-- SKIPS LEADING BLANKS
                    ;   ENTER:   HL=SOURCE ADDR
                    ;   RETURNS: HL=NEXT NON-BLANK ADDR, A=CHAR
                    ;
072C 7E     GETCH:  MOV   A,M           ;   GET CHARACTER
072D FE20           CPI   20H           ;IF NOT BLANK
072F C0             RNZ                 ;  THEN RETURN
0730 23             INX   H             ;  ELSE GET NEXT CHAR
0731 C32C07         JMP   GETCH
                    ;
                    ; GET LAST CHAR - SCANS BACKWARD, SKIPS BLANKS
                    ;   ENTER:   HL = STRING ADDR
                    ;   RETURNS: HL = LAST NON-BLANK CHAR, A = CHAR
                    ;
0734 7E     GETLCH: MOV   A,M           ;   GET CHARACTER
0735 FE20           CPI   20H           ;IF NOT BLANK
0737 C0             RNZ                 ;  THEN RETURN
0738 2B             DCX   H             ;ELSE GET NEXT CHAR
0739 C33407         JMP   GETLCH
                    ;
                    ; GET WORD -- UP TO FIRST BREAK CHARACTER
                    ;            IGNORES LEADING BLANKS
                    ;   ENTER:   HL=SOURCE ADDR
                    ;   RETURNS: 'WORD'=SOURCE STRING + ODH
                    ;            HL= BR CHAR+1 ADDR, B= BR CHAR
                    ;            DE= 'WORD' ADDR AFTER ODH
                    ;            C = NO OF CHARS MOVED INCL BR CHAR
                    ;
073C CD2C07 GETWD:  CALL  GETCH         ;IGNORE LEADING BLANKS
073F 118B02         LXI   D,WORD        ;  DESTIN ADDR
0742 CD5407         CALL  WDTFR         ;MOVE IT
0745 C9             RET
                    ;
                    ; COUNT WORD
                    ;   ENTER:   HL=SOURCE ADDR
                    ;   RETURNS: HL=BR CHAR ADDR
                    ;            A,B=BR CHAR, C=COUNT INCL BR CHAR
                    ;
0746 0E01   CNTWD:  MVI   C,1           ;  COUNT=1
0748 CDB907         CALL  BRCHAR        ;IF CHAR=BREAK
074B C8             RZ                  ;  C=CHAR COUNT
074C 79             MOV   A,C           ;   GET COUNT
074D C601           ADI   1             ;C=C+1
074F 4F             MOV   C,A           ;       STORE IT
0750 23             INX   H             ;HL=NEXT
0751 C34807         JMP   CNTWD+2       ;NEXT CHAR
                    ;
                    ; WORD TRANSFER
                    ;   MOVES STRING FROM HL TO DE + ODH ADDED
                    ;   ENTER:   HL= SOURCE ADDR, DE= DESTIN ADDR
                    ;   RETURNS: HL= SOURCE ADDR AFTER BR CHAR
                    ;            DE= DESTIN ADDR AFTER ODH
                    ;            B= BR CHAR
                    ;            C= NO OF CHARS MOVED INCL BR CHAR
                    ;
0754 0E01   WDTFR:  MVI   C,1           ;  INIT COUNT
0756 CDB907         CALL  BRCHAR        ;IF BREAK CHAR
0759 CA6807         JZ    MVBR          ;  THEN END OF SOURCE
075C 23             INX   H             ;HL= DESTIN ADDR
075D EB             XCHG
075E 70             MOV   M,B           ;   MOVE CHARACTER
075F 79             MOV   A,C           ;   INCR COUNT
0760 C601           ADI   1
0762 4F             MOV   C,A
0763 23             INX   H             ;HL= NEXT SOURCE ADDR
0764 EB             XCHG
0765 C35607         JMP   WDTFR+2
0768 23     MVBR:   INX   H             ;HL= DESTIN BR CHAR ADDR
0769 EB             XCHG
076A 3E0D           MVI   A,0DH         ;   REPLACE WITH ODH
076C 77             MOV   M,A
076D 23             INX   H             ;HL= SOURCE BR CHAR ADDR+1
076E EB             XCHG
076F AF             XRA   A             ; SET RETURN FLAG
0770 C9             RET
                    ;
                    ; GET CONTROL WORD IN 'WORD'
                    ;   REPLACES FINAL Y OR N WITH ODH
                    ;   ENTER:   HL = SOURCE ADDR
                    ;
0771 118B02 GETCTL: LXI   D,WORD        ;  DESTIN ADDR
0774 CD5407         CALL  WDTFR         ;MOVE WORD
0777 79             MOV   A,C           ;   GET COUNT
0778 FE03           CPI   3             ;IF < 3 CHARS MOVED
077A F8             RM                  ;  THEN RETURN
077B 62             MOV   H,D           ;   ADDR WORD
077C 6B             MOV   L,E
077D 2B             DCX   H             ;AVOID COLON
077E 2B             DCX   H
077F CD3407         CALL  GETLCH        ;GET LAST CHAR
0782 FE59           CPI   'Y'           ;IF Y
0784 CA8A07         JZ    YNOUT         ;  THEN REMOVE IT
0787 FE4E           CPI   'N'           ;IF NOT N
```

```
0789 C0             RNZ                 ;  THEN RETURN
078A 3E0D     YNOUT: MVI   A,0DH         ;   REPLACE Y OR N
078C 77              MOV   M,A          ;     WITH 0DH
078D C9              RET
                ;
                ; SAVE OP CODE THROUGH COLON IN LASTOP
                ;
078E 063A     SAVOP: MVI   B,':'
0790 CD3708          CALL  INDX         ;COUNT CHARS TO COLON
0793 2A2201          LHLD  LLSAV        ;ADDR OP CODE
0796 11B802          LXI   D,LASTOP
0799 CD540C          CALL  BLKTFR       ;MOVE CHAR STRING
079C C9              RET
                ;
                ; COUNT LINE
                ;   ENTER:   HL=SOURCE ADDR
                ;   RETURNS: C=CHAR COUNT INCL 0DH OR 01
                ;            HL=BREAK POS., A=BR CHAR
                ;
079D 0E01     CNTLN: MVI   C,1           ;   COUNT=1
079F 7E              MOV   A,M          ;    GET CHARACTER
07A0 FE0D            CPI   0DH          ;  IF 0DH
07A2 C8              RZ                 ;    C=CHAR COUNT
07A3 FE01            CPI   1            ; IF 01
07A5 C8              RZ                 ;   C=CHAR COUNT
07A6 79              MOV   A,C          ;    GET COUNT
07A7 C601            ADI   1            ;C=C+1
07A9 4F              MOV   C,A          ;       STORE IT
07AA 23              INX   H            ;HL=NEXT
07AB C39F07          JMP   CNTLN+2      ;NEXT CHAR
                ;
                ; BACKUP-- DECREMENTS HL BY VALUE OF C-1
                ;   ENTER:   HL START VALUE, C=COUNT
                ;   RETURNS: NEW HL VALUE
                ;
07AE 79       BACKUP: MOV   A,C          ;    GET COUNT
07AF FE01            CPI   1            ;IF COUNT=1
07B1 C8              RZ                 ;   THEN RETURN
07B2 D601            SUI   1            ;C=C-1
07B4 4F              MOV   C,A          ;       STORE C
07B5 2B              DCX   H            ;HL=HL-1
07B6 C3AE07          JMP   BACKUP
                ;
                ; BREAK CHARACTER SEARCH
                ;   ENTER:   HL=CHAR ADDR
                ;   RETURNS: A, B = CHARACTER
                ;            IF BR CHAR THEN Z FLAG TRUE
                ;
07B9 7E       BRCHAR: MOV   A,M          ;    GET CHAR
07BA 47              MOV   B,A          ;    AND SAVE IT
07BB FE20            CPI   ' '          ;CHECK FOR VARIOUS
07BD C8              RZ                 ;BREAK CHARACTERS
07BE FE0D            CPI   0DH          ;    END OF LINE
07C0 C8              RZ
07C1 FE2C            CPI   ','
07C3 C8              RZ
07C4 FE3B            CPI   ';'
07C6 C8              RZ
07C7 FE3A            CPI   ':'
07C9 C8              RZ
07CA FE2E            CPI   '.'
07CC C8              RZ
07CD FE3F            CPI   '?'
07CF C8              RZ
07D0 FE21            CPI   21H          ;EXCLAMATION
07D2 C8              RZ
07D3 FE22            CPI   '"'          ;DOUBLE QUOTE
07D5 C8              RZ
07D6 FE28            CPI   '('          ;L PARENS
07D8 C8              RZ
07D9 FE29            CPI   ')'          ;R PARENS
07DB C8              RZ
07DC FE27            CPI   27H          ;APOSTROPHE
07DE C8              RZ
07DF FE01            CPI   1            ;END OF LIST
07E1 C9              RET                ;BR CHAR NOT FOUND
                ;
                ; INDEX - FIND CHAR POSITION OF MATCHED STRING
                ;    ENTER: HLSAV = STRING ADDR, HLLSAVE = SUBSTR ADDR
                ;    RETURNS: C = CHAR POS OF MATCH, IF NOMATCH, C=0
                ;             HLSAV = STRING ADDR OF FIRST MATCHED CHAR
                ;             EPTR = ADDR OF NEXT CHAR AFTER MATCH
                ;
07E2 AF       INDEX: XRA   A
07E3 324801          STA   SCNT         ;INIT STRING COUNT
07E6 2A1A01   INDE2: LHLD  HLSAV        ;ADDR STRING
07E9 EB              XCHG               ;DE = STRING ADDR
07EA 2A1C01          LHLD  HLLSAV       ;ADDR SUBSTRING
07ED 46              MOV   B,M          ;FIRST SUBSTR CHAR IN B
07EE 23              INX   H            ;ADDR STRING
07EF EB              XCHG
07F0 CD3708          CALL  INDX         ;LOOK FOR FIRST CHAR
07F3 79              MOV   A,C
07F4 B7              ORA   A            ; IF NOT FOUND
07F5 C8              RZ                 ;   THEN RETURN
07F6 221A01          SHLD  HLSAV        ;   ELSE SAVE POINTER
07F9 3A4801          LDA   SCNT         ;GET OLD STRING COUNT
07FC 81              ADD   C            ;ADD NEW COUNT
07FD 324801          STA   SCNT         ;   IN SCNT
0800 2A1C01          LHLD  HLLSAV       ;ADDR SUBSTR
0803 CD9D07          CALL  CNTLN        ;COUNT SUBSTR CHARS
0806 79              MOV   A,C          ;    REDUCE COUNT TO
0807 D601            SUI   1            ;   ALPHA CHARS
0809 4F              MOV   C,A
080A 11B802          LXI   D,WORD       ;   MOVE SAME NUMBER OF
080D 2A1A01          LHLD  HLSAV        ;   CHARS FROM STRING
0810 CD540C          CALL  BLKTFR       ;   TO 'WORD'
0813 223601          SHLD  EPTR         ;SAVE NEXT CHAR ADDR
0816 EB              XCHG               ;ADDR END OF 'WORD'
0817 3E0D            MVI   A,0DH        ;    TERMINATE WITH 0DH
0819 77              MOV   M,A
081A 2A1C01          LHLD  HLLSAV       ;ADDR SUBSTR
081D EB              XCHG               ;DE = SUBSTR ADDR
081E 21B802          LXI   H,WORD       ;   ADDR PORTION OF STRING
0821 CDFB06          CALL  CMPR         ;COMPARE THEM
0824 B7              ORA   A            ; IF FOUND
0825 C23208          JNZ   SETCNT       ;   THEN SET POSITION COUNT
0828 2A1A01          LHLD  HLSAV        ;   ELSE GET STRING POINTER
082B 23              INX   H            ;   BUMP IT
082C 221A01          SHLD  HLSAV        ;   SAVE IT
082F C3E607          JMP   INDE2        ;   AND TRY AGAIN
0832 214801   SETCNT: LXI   H,SCNT       ;   PUT STRING COUNT IN C
0835 4E              MOV   C,M
0836 C9              RET                ;     AND RETURN
                ;
                ; INDX - FIND CHARACTER POSITION OF SINGLE LETTER
                ;    ENTER:    HL = STRING ADDR, B= CHAR
                ;    RETURNS:  C = CHAR POS OF MATCH, IF NOMATCH, C=0
                ;              HL = ADDR OF MATCHED CHAR OR EOL
                ;
0837 0E01     INDX:  MVI   C,1           ;   INIT C REGISTER
0839 7E              MOV   A,M          ;    GET CHAR
```

```
083A B8             CMP   B             ;IF B-CHAR FOUND
083B C8             RZ                  ;  THEN RETURN
083C FE0D           CPI   0DH           ; IF END OF LINE
083E CA4908         JZ    ZC            ;  THEN ZERO COUNT
0841 79             MOV   A,C           ;     ELSE
0842 C601           ADI   1             ;  BUMP COUNT
0844 4F             MOV   C,A
0845 23             INX   H             ;  BUMP ADDR
0846 C33908         JMP   INDX+2        ;  GO TO NEXT
0849 AF       ZC:   XRA   A             ; RETURN WITH
084A 4F             MOV   C,A           ;        C = 0
084B C9             RET
                ;
                ; SINDX - SPECIAL INDEX FOR POSITION OF $ OR #
                ;    ENTER:   HL = STRING ADDRESS
                ;    RETURNS: BA = $, #, OR 0DH;  C = CHAR POS
                ;             HL = ADDR OF MATCHED CHAR
                ;
084C 0E01       SINDX: MVI   C,1           ;  INIT C REGISTER
084E 7E             MOV   A,M           ;   GET CHAR
084F FE24           CPI   '$'           ;IF $
0851 C8             RZ                  ;  THEN RETURN
0852 FE23           CPI   0430          ;IF #
0854 C8             RZ                  ;  THEN RETURN
0855 FE0D           CPI   0DH           ; IF EOL
0857 C8             RZ                  ;  THEN RETURN
0858 79             MOV   A,C           ;    ELSE
0859 C601           ADI   1             ;BUMP COUNT
085B 4F             MOV   C,A
085C 23             INX   H             ;BUMP ADDR
085D C34E08         JMP   SINDX+2       ;GO TO NEXT
                ;
                ; LETTER TESTS WHETHER CHARACTER IS UPCASE A-Z
                ;    ENTER:   HL = ADDR OF CHAR
                ;    RETURNS: ZERO FLAG TRUE IF IT IS
                ;             B = CHARACTER
                ;
0860 7E         LETTER: MOV   A,M           ;   GET CHAR
0861 46             MOV   B,M           ;   SAVE IN B
0862 FE41           CPI   41H           ;CHECK RANGE
0864 FA6E08         JM    NOTL          ;TOO LOW?
0867 FE5A           CPI   5AH
0869 F26E08         JP    NOTL          ;TOO HIGH?
086C AF             XRA   A             ;    ELSE RESET ZERO FLAG
086D C9             RET                 ;  AND RETURN IF LETTER
086E B4         NOTL:   ORA   H             ; RETURN IF NOT LETTER
086F C9             RET
                ;
                ; SETUP GETS CHAR COUNT AND SETS ADDR FOR TEXT MOVES
                ;
0870 CD9D07     SETUP:  CALL  CNTLN         ;C=CHAR COUNT
0873 41             MOV   B,C           ;   SAVE COUNT
0874 CDAE07         CALL  BACKUP        ;RESET HL
0877 48             MOV   C,B           ;   RESET COUNT
0878 C9             RET
                ;
                ; T OPERATION--DISPLAY 'T' STATEMENT
                ;   ENTER:   HLSAV= FIRST CHAR OF T EXPRESSION FIELD
                ;
0879 2A1A01     TOP:    LHLD  HLSAV         ;RETRIEVE POINTER
087C 223801         SHLD  CPTR          ;SAVE CHAR POINTER
087F 222201         SHLD  LLSAV         ;SAVE FIRST CHAR ADDR
0882 214C01         LXI   H,TSAVE       ;  DESTIN START ADDR
0885 222001         SHLD  DESAV         ;SAVE DESTIN ADDR
0888 EB             XCHG                ;AND KEEP IN DE
0889 2A3801     TMORE:  LHLD  CPTR          ;GET CHAR POINTER
088C CD4C08         CALL  SINDX         ;LOOK FOR $ OR #
088F FE24           CPI   '$'           ;IF $ FOUND
0891 CA9C08         JZ    GETXT         ;  THEN GET LABELED TEXT
0894 FE23           CPI   0430          ;IF # FOUND
0896 CAAD08         JZ    GETNUM        ;  THEN GET NUMBER
0899 C3B308         JMP   TMOVE         ;ELSE MOVE REST OF TEXT
089C CD1109     GETXT:  CALL  INSERT        ;INSERT TEXT
089F FE01           CPI   1             ;IF FOUND (A NOT 01)
08A1 C28908         JNZ   TMORE         ;  THEN CONTINUE
08A4 114C01         LXI   D,TSAVE       ;  ELSE DISPLAY THE LINE
08A7 2A2201         LHLD  LLSAV
08AA C3BA08         JMP   TALL
08AD CDCC08     GETNUM: CALL  INSNUM        ;INSERT NUMBER
08B0 C39F08         JMP   GETXT+3       ;  AND SEE IF FOUND
08B3 2A2001     TMOVE:  LHLD  DESAV         ;DESTIN ADDR
08B6 EB             XCHG
08B7 2A3801         LHLD  CPTR          ;ADDRESS INPUT
08BA CD7008     TALL:   CALL  SETUP         ;C = CHAR COUNT
08BD CD540C         CALL  BLKTFR        ;MOVE T-TEXT
08C0 3E0D           MVI   A,0DH         ;   TERMINATE
08C2 EB             XCHG
08C3 77             MOV   M,A
08C4 214C01         LXI   H,TSAVE       ;  ADDRESS TEXT
08C7 CD490D         CALL  DSPLY         ;DISPLAY T STATEMENT
08CA AF             XRA   A             ; SET RETURN FLAG
08CB C9             RET
                ;
                ; INSERT NUMERIC VALUE INTO T-STATEMENT
                ;    ENTER:   C = POSITION OF '#'
                ;    RETURNS: TSAVE CONTAINS ASSEMBLED T-TEXT
                ;             IF LABEL NOT FOUND, THEN A = 01
                ;
08CC 79         INSNUM: MOV   A,C           ;   GET POSITION OF #
08CD D601           SUI   1             ;REDUCE COUNT BY 1
08CF CADD08         JZ    VBL           ;IF 0 THEN GET VARIABLE
08D2 4F             MOV   C,A
08D3 2A2001         LHLD  DESAV         ;DESTIN ADDR
08D6 EB             XCHG
08D7 2A3801         LHLD  CPTR          ;GET CHAR POINTER
08DA CD540C         CALL  BLKTFR        ;MOVE FRONT OF TEXT
08DD 44         VBL:    MOV   B,H
08DE 4D             MOV   C,L
08DF EB             XCHG
08E0 222001         SHLD  DESAV         ;SAVE DESTIN POINTER
08E3 60             MOV   H,B           ;   GET CHAR POINTER
08E4 69             MOV   L,C
08E5 23             INX   H             ;BUMP TO VAR NAME
08E6 4E             MOV   C,M           ;   SAVE NAME IN C
08E7 23             INX   H             ;BUMP ADDR
08E8 223801         SHLD  CPTR          ;SAVE CHAR POINTER
08EB 41             MOV   B,C           ;   PUT VAR NAME IN B
08EC CDD804         CALL  VARMCH        ;LOOK UP VAR NAME
08EF FE01           CPI   1             ;IF NOT FOUND (A = 01)
08F1 C8             RZ                  ;  THEN RETURN
08F2 23             INX   H             ;ELSE POINT AT VALUE
08F3 5E             MOV   E,M           ;   GET VALUE IN E
08F4 21B802         LXI   H,WORD        ;  PUT DIGITS IN WORD
08F7 CD6506         CALL  PUTNM
08FA 2A2001         LHLD  DESAV         ;DESTIN ADDR
08FD EB             XCHG
08FE 21B802         LXI   H,WORD        ;  DIGIT CHAR ADDR
0901 CD7008         CALL  SETUP         ;C = CHAR COUNT+1
0904 79             MOV   A,C
0905 D601           SUI   1
0907 4F             MOV   C,A           ;       C = CHAR COUNT
0908 CD540C         CALL  BLKTFR        ;MOVE DIGITS
```

```
090B EB             XCHG
090C 222001         SHLD  DESAV         ;SAVE DESTIN ADDR
090F AF             XRA   A             ; SET RETURN FLAG
0910 C9             RET
                    ;
                    ; INSERT LABELED TEXT INTO T-STATEMENT
                    ;    ENTER:   C = POSITION OF '$'
                    ;    RETURNS: TSAVE CONTAINS ASSEMBLED T-TEXT
                    ;             IF LABEL NOT FOUND, THEN A = 01
                    ;
0911 79     INSERT: MOV   A,C           ;   GET POSITION COUNT
0912 D601           SUI   1             ;REDUCE COUNT BY 1
0914 CA2209         JZ    LBL           ;IF 0 THEN POINT TO LABEL
0917 4F             MOV   C,A
0918 2A2001         LHLD  DESAV         ;DESTIN ADDR
091B EB             XCHG
091C 2A3801         LHLD  CPTR          ;GET CHAR POINTER
091F CD540C         CALL  BLKTFR        ;MOVE FRONT OF TEXT
0922 44     LBL:    MOV   B,H           ; SAVE CHAR POINTER IN BC
0923 4D             MOV   C,L
0924 EB             XCHG
0925 222001         SHLD  DESAV         ;SAVE DESTIN POINTER
0928 60             MOV   H,B           ;   GET CHAR POINTER
0929 69             MOV   L,C
092A CD3C07         CALL  GETWD         ;GET LABEL
092D 2B             DCX   H             ;ADDR BK CHAR
092E 223801         SHLD  CPTR          ;SAVE CHAR POINTER
0931 2A3401         LHLD  APTR          ;HL = START OF LIST
0934 23             INX   H
0935 CD9606         CALL  LOOKS         ;LOOK FOR STRING NAME
0938 FE01           CPI   1             ;IF NOT FOUND (A = 01)
093A C8             RZ                  ;   THEN RETURN
093B CD3F09         CALL  GETA          ;   ELSE GET A-TEXT
093E C9             RET
                    ;
                    ; GET A-TEXT POINTED TO BY MATCHED LABEL
                    ;    ENTER:   HL = BLANK AFTER MATCHED LABEL
                    ;             DESAV = DESTIN ADDRESS
                    ;    RETURNS: A-TEXT MOVED TO DESTINATION
                    ;             DESAV = NEXT DESTIN ADDRESS
                    ;
093F 23     GETA:   INX   H             ;ADDR FIRST A-CHAR
0940 44             MOV   B,H           ;   BC = A-TEXT ADDR
0941 4D             MOV   C,L
0942 2A2001         LHLD  DESAV         ;DESTIN ADDR
0945 EB             XCHG
0946 60             MOV   H,B           ;   HL = A-TEXT ADDR
0947 69             MOV   L,C
0948 CD7008         CALL  SETUP         ;C = CHAR COUNT
094B 79             MOV   A,C           ;   REDUCE COUNT TO
094C D601           SUI   1             ;   EXCLUDE MOVE OF ODH
094E 4F             MOV   C,A
094F CD540C         CALL  BLKTFR        ;MOVE A-TEXT
0952 EB             XCHG
0953 222001         SHLD  DESAV         ;SAVE DESTIN POINTER
0956 AF             XRA   A             ; SET RETURN FLAG
0957 C9             RET
                    ;
                    ; A OPERATION-- ACCEPT INPUT
                    ;   ENTER:    HLSAV = ADDR AFTER COLON
                    ;   RETURNS:  INPUT IN EBUFF
                    ;   'CTL Z' ALLOWS SINGLE STATEMENT EXECUTION OR QUIT
                    ;
0958 2A1A01 AOP:    LHLD  HLSAV         ;ADDR EXPRESSION
095B CD2C07         CALL  GETCH         ;GET FIRST CHAR
095E FE24           CPI   '$'           ;IF NOT $
0960 C27009         JNZ   NIN           ;   THEN LOOK FOR #
0963 113F02         LXI   D,LABSAV      ;   ELSE ADDR DESTIN
0966 CD5407         CALL  WDTFR         ;   SAVE THE LABEL
0969 CD8809         CALL  ENTR          ;GET THE ENTRY
096C CDB309         CALL  ASTORE        ;   STORE IT
096F C9             RET                 ;   THEN EXIT
0970 FE23   NIN:    CPI   0430          ;IF NOT #
0972 C28409         JNZ   CENT          ;   THEN CALL ENTRY
0975 23             INX   H             ;ELSE ADDR VARIABLE
0976 7E             MOV   A,M           ;    GET THE NAME
0977 324A01         STA   VARSAV        ;AND SAVE IT
097A CD8809         CALL  ENTR          ;GET THE ENTRY
097D CD9F09         CALL  CKNUM         ;REQUIRE NUMERIC
0980 CD050A         CALL  NSTORE        ;   STORE THE NUMBER
0983 C9             RET                 ;THEN EXIT
0984 CD8809 CENT:   CALL  ENTR
0987 C9             RET
                    ;
0988 CD650C ENTR:   CALL  KEYIN         ;GET ONE LINE IN EBUFF
098B 219D01         LXI   H,EBUFF       ;IF ENTRY NOT CTL Z
098E 7E             MOV   A,M
098F FE1A           CPI   1AH
0991 C0             RNZ                 ;THEN RETURN
0992 23             INX   H             ;ELSE BUMP POINTER
0993 7E             MOV   A,M           ;GET CHAR
0994 FE0D           CPI   0DH           ;IF CR
0996 CA0F03         JZ    RSTRT         ;   THEN RESTART
0999 CDCE03         CALL  OPS           ;ELSE DO IMMED OP
099C C38809         JMP   ENTR          ;   AND ACCEPT MORE INPUT
                    ;
099F FE30   CKNUM:  CPI   '0'           ;CHECK FOR NUMBER
09A1 FAA709         JM    NERR          ;TOO LOW
09A4 FE3A           CPI   '9'+1
09A6 F8             RM                  ;MUST BE NUMBER
09A7 214A0E NERR:   LXI   H,NMSG        ;OUT OF RANGE
09AA CDDE0D         CALL  ERROR         ;SEND MESSAGE
09AD CD8809         CALL  ENTR          ;TRY AGAIN
09B0 C39F09         JMP   CKNUM
                    ;
                    ; A ITEM STORE - STORAGE OF LABELED TEXT FROM ENTRY
                    ;    FROM TOP OF INPUT BUFFER AREA AND SETS POINTER
                    ;
09B3 219D01 ASTORE: LXI   H,EBUFF       ;   SOURCE ADDR
09B6 CD9D07         CALL  CNTLN         ;C = CHAR COUNT
09B9 41             MOV   B,C           ;    DUPL COUNT IN B
09BA 2A3401         LHLD  APTR          ;LAST A-ADDR
09BD CDE609         CALL  DECA          ;BACK UP DESTIN ADDRESS
09C0 79             MOV   A,C           ;    IF CHAR COUNT = 0
09C1 B7             ORA   A
09C2 C8             RZ                  ;   THEN RETURN
09C3 219D01         LXI   H,EBUFF       ;   SOURCE ADDR
09C6 CD540C         CALL  BLKTFR        ;MOVE TEXT
09C9 213F02         LXI   H,LABSAV      ;   ADDR LABEL
09CC CD9D07         CALL  CNTLN         ;C = CHAR COUNT
09CF 41             MOV   B,C           ;    DUPL COUNT IN B
09D0 2A3401         LHLD  APTR          ;GET A-POINTER
09D3 CDE609         CALL  DECA          ;BACK UP ADDR
09D6 79             MOV   A,C           ;    IF CHAR COUNT = 0
09D7 B7             ORA   A
09D8 C8             RZ                  ;   THEN RETURN
09D9 213F02         LXI   H,LABSAV      ;   SOURCE ADDR
09DC CD540C         CALL  BLKTFR        ;MOVE THE LABEL
09DF 6B             MOV   L,E           ;    ADDR A-TEXT
09E0 62             MOV   H,D
09E1 2B             DCX   H             ;BACK UP ONE CHAR
09E2 3E20           MVI   A,' '         ;    AND SET
```

```
09E4 77             MOV   M,A          ;     BLANK THERE
09E5 C9             RET
                ;
                ; DECREMENT ADDRESS FOR TEXT STORAGE
                ;     ENTER:  HL = LAST (LOWEST) ADDRESS USED (01)
                ;             B & C = CHAR COUNT IN WORD TO BE MOVED
                ;     RETURNS:DE = DESTIN ADDR,  C = CHAR COUNT
                ;        APTR POINTS AT STOP(01) BELOW LIST
                ;
09E6 5D         DECA:  MOV   E,L          ;DESTIN ADDR IN DE
09E7 54             MOV   D,H
09E8 2B             DCX   H              ;DECR POINTER
09E9 7E             MOV   A,M            ;   GET CHARACTER
09EA FE01           CPI   1              ;IF 01 (END OF SPACE)
09EC CAFC09         JZ    STOVF          ;  THEN STORAGE OVERFLOW
09EF 78             MOV   A,B            ;       DECR COUNT
09F0 D601           SUI   1
09F2 47             MOV   B,A
09F3 C2E609         JNZ   DECA           ;BACK UP AGAIN?
09F6 223401         SHLD  APTR           ;SAVE A-POINTER
09F9 3601           MVI   M,1            ;SET STOP
09FB C9             RET
                ;
09FC 21FF0D     STOVF: LXI   H,NRMSG     ;  COMPLAIN OF OVERFLOW
09FF CDDE0D         CALL  ERROR
0A02 0E00           MVI   C,0            ;  SET CHAR COUNT = 0
0A04 C9             RET
                ;
                ; NUMBER STORAGE - STORAGE OF NUMERIC VALUE FROM ENTRY
                ;             AS VALUE OF VARIABLE NAME IN A-STATEMENT
                ;
0A05 219D01     NSTORE: LXI   H,EBUFF    ;  SOURCE ADDR
0A08 CD2C07         CALL  GETCH          ;ADDR 1ST CHAR
0A0B CD3106         CALL  GETNM          ;GET THE NUMBER
0A0E 214A01         LXI   H,VARSAV       ;  GET THE NAME IN B
0A11 46             MOV   B,M
0A12 CDD804         CALL  VARMCH         ;LOOK IT UP
0A15 FE01           CPI   1              ;IF END MARKER
0A17 CCCB04         CZ    BADFRM         ;THEN COMPLAIN
0A1A C8             RZ                   ;  AND RETURN
0A1B 23             INX   H              ;  ELSE BUMP TO NEXT
0A1C 73             MOV   M,E
0A1D C9             RET
                ;
                ; M OPERATION - MOVING WINDOW STRING MATCH
                ;    COMPARE ITEMS IN LIST WITH LAST INPUT
                ;      M-ITEMS HAVE MULTIPLE BLANKS REDUCED TO ONE
                ;      INPUT HAS BLANK ADDED AT EACH END AND
                ;      MULTIPLE BLANKS REDUCED TO ONE.
                ; ENTER: HLSAV = ADDR AFTER LAST COLON, INPUT IN EBUFF
                ; RETURNS: YNSW = 0 IF MATCH NOT FOUND WITH LAST ENTRY
                ;
0A1E 214701     MC:    LXI   H,MBRCH     ;SET BR CHAR
0A21 365E           MVI   M,'^'          ;  TO CARET (SHIFT N)
0A23 C32B0A         JMP   MOP1
0A26 214701     MOP:   LXI   H,MBRCH     ;SET BR CHAR TO COMMA
0A29 362C           MVI   M,','
0A2B 2A1A01     MOP1:  LHLD  HLSAV       ;ADDR EXPRESSION FIELD
0A2E 223C01         SHLD  MPTR           ;INIT M-POINTER
0A31 CD800A     NEXTM: CALL  MMOV        ;MSAVE = M-ITEM
0A34 2A3C01         LHLD  MPTR           ;ADDR M-ITEM
0A37 221A01         SHLD  HLSAV
0A3A CDD70A         CALL  SQUEZ          ;REDUCE MULTIPLE BLANKS
0A3D 219D01         LXI   H,EBUFF
0A40 221A01         SHLD  HLSAV          ;HLSAV = STRING ADDR
0A43 CDA90A         CALL  PAD            ;ADD BLANKS AT EACH END
0A46 CDD70A         CALL  SQUEZ          ;REDUCE MULTIPLE BLANKS
0A49 219D01         LXI   H,EBUFF        ;  PUT EBUFF ADDR
0A4C 221A01         SHLD  HLSAV          ;  IN HLSAV
0A4F 21EE01         LXI   H,MSAVE        ;  PUT MSAVE ADDR
0A52 221C01         SHLD  HLLSAV         ; IN HLLSAV
0A55 CDE207         CALL  INDEX          ;LOOK FOR M-ITEM
0A58 79             MOV   A,C
0A59 B7             ORA   A              ; IF ITEM FOUND
0A5A C4760A         CNZ   SWY            ;  THEN SET SWITCH YES
0A5D C0             RNZ                  ;  AND RETURN
0A5E 2A3C01         LHLD  MPTR           ;RETRIEVE M-POINTER
0A61 2B             DCX   H              ;ADDR BR CHAR
0A62 7E             MOV   A,M            ;    IF END OF LINE
0A63 FE0D           CPI   0DH
0A65 CA720A         JZ    MDONE          ;  THEN QUIT
0A68 23             INX   H              ;ADDR NEXT CHAR
0A69 7E             MOV   A,M            ;    IF END OF LINE
0A6A FE0D           CPI   0DH
0A6C CA720A         JZ    MDONE          ;  THEN QUIT
0A6F C3310A         JMP   NEXTM          ;ELSE MORE M-ITEMS
0A72 CD7B0A     MDONE: CALL  SWN         ;SET SWITCH NO
0A75 C9             RET
                ;
0A76 B4         SWY:   ORA   H           ; SET YN SWITCH YES
0A77 324B01         STA   YNSW
0A7A C9             RET
                ;
0A7B AF         SWN:   XRA   A           ; SET YN SWITCH NO
0A7C 324B01         STA   YNSW
0A7F C9             RET
                ;
                ; M-MOVE: MOVE M-ITEM TO MSAVE
                ;   ITEMS ARE SEPARATED BY COMMA OR TERMINATED BY 0DH
                ;   ENTER:   MPTR = M-ITEM ADDRESS
                ;   RETURNS: HL & MPTR = NEXT M-ITEM ADDR
                ;            B = BR CHAR
                ;
0A80 2A3C01     MMOV:  LHLD  MPTR        ;GET M-POINTER
0A83 11EE01         LXI   D,MSAVE        ;  DESTIN ADDR
0A86 CD9F0A     MMOV2: CALL  MBR         ;IF BR CHAR OR EOL
0A89 CA940A         JZ    SMOV           ;  THEN STOP THE MOVE
0A8C 23             INX   H
0A8D EB             XCHG                 ;HL = DESTIN ADDR
0A8E 70             MOV   M,B            ;    MOVE CHAR
0A8F 23             INX   H
0A90 EB             XCHG                 ;HL = NEXT SOURCE ADDR
0A91 C3860A         JMP   MMOV2          ;CHECK THE NEXT CHAR
0A94 23         SMOV:  INX   H
0A95 EB             XCHG                 ;HL = DESTIN BR CHAR ADDR
0A96 3E0D           MVI   A,0DH          ;    PUT AN ODH THERE
0A98 77             MOV   M,A
0A99 23             INX   H
0A9A EB             XCHG                 ;HL = NEXT M-ITEM ADDR
0A9B 223C01         SHLD  MPTR           ;SAVE M-POINTER
0A9E C9             RET
                ;
                ; M-BREAK CHAR BETWEEN ITEMS
                ;    ENTER:   HL = CHAR ADDR
                ;    RETURNS: A,B = CHAR.  IF BR CHAR THEN Z FLAG TRUE
                ;
0A9F 7E         MBR:   MOV   A,M         ;    GET CHAR
0AA0 47             MOV   B,A            ;    SAVE IT
0AA1 FE0D           CPI   0DH            ; IF EOL
0AA3 C8             RZ                   ;  THEN RETURN
0AA4 3A4701         LDA   MBRCH          ;GET CURRENT BR CHAR
0AA7 B8             CMP   B
```

```
0AA8 C9             RET
                    ;
                    ; PAD ADDS A BLANK TO EACH END OF A STRING
                    ;   ENTER:   HLSAV = STRING ADDRESS
                    ;
0AA9 218B02 PAD:    LXI   H,WORD
0AAC 3E20           MVI   A,' '         ;  SET BLANK AT FRONT OF
0AAE 77             MOV   M,A
0AAF 23             INX   H             ;DE = DESTIN ADDR
0AB0 EB             XCHG
0AB1 2A1A01         LHLD  HLSAV         ;GET SOURCE ADDR
0AB4 CD9D07         CALL  CNTLN         ;C = CHAR COUNT
0AB7 2A1A01         LHLD  HLSAV         ;GET SOURCE ADDR
0ABA CD540C         CALL  BLKTFR        ;MOVE TEXT
0ABD EB             XCHG                ;ADDR NEW TEXT END
0ABE 2B             DCX   H             ;SET BLANK AT
0ABF 3E20           MVI   A,' '         ;     END OF
0AC1 77             MOV   M,A           ;      TEMP STRING
0AC2 23             INX   H             ;SET EOL
0AC3 3E0D           MVI   A,0DH
0AC5 77             MOV   M,A
0AC6 2A1A01         LHLD  HLSAV         ;MOVE NEW STRING
0AC9 EB             XCHG                ;  TO ORIGINAL
0ACA 218B02         LXI   H,WORD        ;     LOCATION
0ACD CD9D07         CALL  CNTLN
0AD0 218B02         LXI   H,WORD
0AD3 CD540C         CALL  BLKTFR
0AD6 C9             RET
                    ;
                    ; SQUEZ REDUCES MULTIPLE BLANKS TO A SINGLE BLANK
                    ;   ENTER:   HLSAV = STRING ADDRESS
                    ;
0AD7 2A1A01 SQUEZ:  LHLD  HLSAV         ;ADDR STRING
0ADA 221C01         SHLD  HLLSAV        ;SAVE POINTER
0ADD 7E             MOV   A,M           ;    GET CHAR
0ADE FE0D   CKEOL:  CPI   0DH           ;  IF EOL
0AE0 C8             RZ                  ;  THEN RETURN
0AE1 FE20           CPI   ' '           ;IF BLANK
0AE3 CAEA0A         JZ    CKNC          ;  THEN CHECK NEXT CHAR
0AE6 23             INX   H             ;ELSE BUMP ADDR
0AE7 C3DA0A         JMP   SQUEZ+3       ;  AND CONTINUE
0AEA 23     CKNC:   INX   H             ;IF NEXT CHAR IS
0AEB 7E     CKNC1:  MOV   A,M           ;     NOT BLANK
0AEC FE20           CPI   ' '           ;  THEN CHECK IF EOL
0AEE C2DE0A         JNZ   CKEOL
0AF1 221C01         SHLD  HLLSAV        ;ELSE SAVE ADDRESS
0AF4 CDFD0A         CALL  SHIFT         ;  REMOVE A BLANK
0AF7 2A1C01         LHLD  HLLSAV        ;  RETRIEVE ADDRESS
0AFA C3EB0A         JMP   CKNC1         ;  AND CONTINUE
                    ;
                    ; SHIFT STRING CHARS LEFT WITH LOSS OF FIRST CHAR
                    ;   ENTER:   HL = ADDR OF STRING
                    ;   RETURNS: HL = ADDR OF 0DH
                    ;
0AFD 23     SHIFT:  INX   H             ;ADDR NEXT CHAR
0AFE 46             MOV   B,M           ;    GET IT IN B
0AFF 2B             DCX   H             ;MOVE IT
0B00 70             MOV   M,B
0B01 7E             MOV   A,M
0B02 FE0D           CPI   0DH           ;  IF IT WAS EOL
0B04 C8             RZ                  ;  THEN RETURN
0B05 23             INX   H             ;  ELSE MOVE ANOTHER
0B06 C3FD0A         JMP   SHIFT
                    ;
                    ; C OPERATION:  COMPUTE WITH TEXT OF STATEMENT
                    ;    LIMITED TO: X = NN  (NN = INTEGER -99 TO +99)
                    ;                OR X = X + NN  OR X = X - NN
                    ;                OR X = X + X   OR X = X - X
                    ;      WHERE X = SINGLE LETTER VARIABLE NAME A-Z
                    ;           FIRST LETTER ONLY OF LONGER NAME GETS USED
                    ;    ENTER:   HLSAV = ADDR OF EXPRESSION FIELD
                    ;
0B09 2A1A01 COP:    LHLD  HLSAV         ;ADDR EXPRESSION FIELD
0B0C 063D           MVI   B,'='         ;  LOOK FOR EQUAL SIGN
0B0E CD3708         CALL  INDX
0B11 79             MOV   A,C
0B12 B7             ORA   A             ; IF NOT PRESENT
0B13 CA770B         JZ    EXMSG         ;  THEN COMPLAIN
0B16 23             INX   H             ;BUMP POINTER
0B17 CD2C07         CALL  GETCH         ;GET THE CHAR
0B1A FE2D           CPI   '-'           ;IF NOT MINUS SIGN
0B1C C22C0B         JNZ   CGVAL         ;  THEN GET THE VALUE
0B1F 221C01         SHLD  HLLSAV        ;SAVE THE POINTER
0B22 AF             XRA   A
0B23 324901         STA   TEMP          ;SET TEMP = 0
0B26 2A1C01         LHLD  HLLSAV        ;RETRIEVE POINTER
0B29 C3600B         JMP   SUBV          ;  AND SUBTRACT
0B2C CD9F0B CGVAL:  CALL  GVALUE        ;GET THE VALUE
0B2F 214901         LXI   H,TEMP        ;  SAVE IT
0B32 73             MOV   M,E
0B33 2A3801         LHLD  CPTR          ;RETRIEVE CHAR POINTER
0B36 23             INX   H             ;BUMP POINTER
0B37 CD2C07         CALL  GETCH         ;GET CHAR
0B3A FE0D           CPI   0DH           ; IF END OF LINE
0B3C CA730B         JZ    AVAL          ;  THEN ASSIGN VALUE
0B3F FE2D           CPI   '-'           ;IF MINUS
0B41 CA600B         JZ    SUBV          ;  THEN SUBTRACT VALUE
0B44 FE2B           CPI   '+'           ;IF PLUS
0B46 CA4F0B         JZ    ADDV          ;  THEN ADD VALUE
0B49 CD9F0B         CALL  GVALUE        ;ELSE GET VALUE
0B4C C3730B         JMP   AVAL          ;ASSIGN VALUE
0B4F 23     ADDV:   INX   H             ;BUMP POINTER
0B50 CD9F0B         CALL  GVALUE        ;GET THE VALUE
0B53 3A4901         LDA   TEMP          ;GET OLD VALUE
0B56 83             ADD   E             ;ADD VALUES
0B57 FE64           CPI   100           ;IF > 99
0B59 F2810B         JP    OVMSG         ;  THEN COMPLAIN
0B5C 5F             MOV   E,A           ;      ELSE FORM NEW VALUE
0B5D C3730B         JMP   AVAL          ;ASSIGN VALUE
0B60 23     SUBV:   INX   H             ;BUMP POINTER
0B61 CD9F0B         CALL  GVALUE        ;GET THE VALUE
0B64 3A4901         LDA   TEMP          ;GET OLD VALUE
0B67 93             SUB   E             ;SUBTRACT VALUES
0B68 FE9D           CPI   9DH           ;IF >= -99
0B6A F2720B         JP    AVAL-1        ;  THEN CONTINUE
0B6D FE64           CPI   100           ;IF > 99
0B6F F2900B         JP    UNMSG         ;  THEN COMPLAIN
0B72 5F             MOV   E,A
0B73 CDC00B AVAL:   CALL  ASSIGN
0B76 C9             RET
                    ;
0B77 CD7908 EXMSG:  CALL  TOP           ;SHOW THE BAD EXPR
0B7A 21080E         LXI   H,EXPMSG
0B7D CDDE0D         CALL  ERROR
0B80 C9             RET
                    ;
0B81 1E63   OVMSG:  MVI   E,99          ;  SET VALUE TO 99
0B83 CDC00B         CALL  ASSIGN
0B86 CD7908         CALL  TOP           ;SHOW THE EXPR
0B89 211C0E         LXI   H,OVFMSG
0B8C CDDE0D         CALL  ERROR
0B8F C9             RET
```

```
                        ;
0B90 1E9D       UNMSG:  MVI   E,-99          ;  SET VALUE TO -99
0B92 CDC00B             CALL  ASSIGN
0B95 CD7908             CALL  TOP
0B98 21290E             LXI   H,UNFMSG
0B9B CDDE0D             CALL  ERROR
0B9E C9                 RET
                        ;
                        ; GVALUE - GETS VALUE OF CONSTANT OR VARIABLE
                        ;     ENTER:   HL = ADDRESS OF CHAR NAMING THE VARIABLE
                        ;              CPTR = ADDRESS OF CHAR
                        ;     RETURNS: E = VALUE
                        ;              CPTR = ADDRESS OF THE CHARACTER
                        ;
0B9F CD2C07     GVALUE: CALL  GETCH          ;GET CHAR
0BA2 5F                 MOV   E,A            ;       SAVE IN E
0BA3 CD6008             CALL  LETTER         ;IF LETTER
0BA6 CAB00B             JZ    LTR            ;  THEN PROCEED
0BA9 CD3106             CALL  GETNM          ;ELSE EXPECT NUMBER
0BAC 223801             SHLD  CPTR           ;SAVE CHAR POINTER
0BAF C9                 RET
0BB0 223801     LTR:    SHLD  CPTR           ;SAVE CHAR POINTER
0BB3 46         CONV:   MOV   B,M            ;   SAVE CHAR IN B
0BB4 CDD804             CALL  VARMCH         ;LOOK IT UP
0BB7 FE01               CPI   1              ;IF END MARKER
0BB9 CCCB04             CZ    BADFRM         ;  THEN COMPLAIN
0BBC C8                 RZ                   ;  AND RETURN
0BBD 23                 INX   H              ;ELSE POINT AT VALUE
0BBE 5E                 MOV   E,M            ;   SAVE VALUE IN E
0BBF C9                 RET
                        ;
                        ; ASSIGN SETS A NEW VALUE TO AN OLD OR NEW VARIABLE
                        ;     ENTER:   HLSAV = ADDR OF EXPRESSION FIELD
                        ;              BINARY VALUE IN E
                        ;
0BC0 2A1A01     ASSIGN: LHLD  HLSAV          ;ADDR EXPRESSION FIELD
0BC3 CD2C07             CALL  GETCH          ;GET FIRST CHAR
0BC6 CD6008             CALL  LETTER         ;IF NOT A LETTER
0BC9 C2770B             JNZ   EXMSG          ;  THEN COMPLAIN
0BCC 46                 MOV   B,M            ;   GET CHAR IN B
0BCD CDD804             CALL  VARMCH         ;LOOK IT UP
0BD0 FE01               CPI   1              ;IF END MARKER
0BD2 CCCB04             CZ    BADFRM         ;  THEN COMPLAIN
0BD5 C8                 RZ                   ;  AND RETURN
0BD6 23                 INX   H              ;  ELSE BUMP TO VALUE ADDR
0BD7 73                 MOV   M,E
0BD8 C9                 RET                  ;  AND RETURN
                        ;
                        ; BASIC INTERPRETATION -
                        ;   IF PROGRAM TEXT IS NOT LEGAL PILOT, THEN
                        ;   AN ALTERNATE INTERPRETER SUCH AS BASIC CAN BE
                        ;   SUPPLIED TO BE TRIED BEFORE PILOT COMPLAINS.
                        ;
0BD9 B4         BASIC:  ORA   H              ;DUMMY ILLEGAL RETURN
0BDA C9                 RET
                        ;
                        ; A OPERATION -
                        ;     ENTER:   HLSAV = A-STATEMENT ADDRESS
                        ;
0BDB C9         AOP:    RET
                        ;
                        ; E OPERATION - RETURNS FROM CALL OR ENDS PROGRAM
                        ;
0BDC 3A4501     EOP:    LDA   LEVEL          ;IF RETURN LEVEL = 0
0BDF B7                 ORA   A              ;   THEN QUIT
0BE0 CA0F03             JZ    RSTRT

0BE3 CD380C             CALL  RESRET         ;  ELSE SET RETURN FROM
0BE6 C9                 RET                  ;  PILOT CALL
                        ;
0BE7 210010     LOAD:   LXI   H,PBUFF
0BEA CDCF0C             CALL  INPUT
0BED C9                 RET
                        ;
                        ; NEW$ DELETES $NAMES BY RESETTING A-POINTER
                        ;
0BEE 2A3401     NEWN:   LHLD  APTR           ;REMOVE STOP CHAR
0BF1 3620               MVI   M,20H
0BF3 2A3E01             LHLD  MEMTP          ;ADDR MEMTP
0BF6 3601               MVI   M,1            ;PLACE STOP CHAR
0BF8 2B                 DCX   H
0BF9 223401             SHLD  APTR           ;STORE MEMTP-1 ADDRESS
0BFC C9                 RET
                        ;
                        ; INITIALIZE NUMERIC VARIABLES
                        ;   SETS A-Z TO ZERO VALUE
                        ;
0BFD 215602     INITV:  LXI   H,NVAR         ;ADDR FRONT OF VAR LIST
0C00 0641               MVI   B,'A'          ;START WITH 'A'
0C02 78                 MOV   A,B
0C03 FE5B       NV:     CPI   'Z'+1          ;IF ALPHABET COMPLETE
0C05 C8                 RZ                   ;THEN RETURN
0C06 77                 MOV   M,A            ;   STORE THE LETTER
0C07 23                 INX   H              ;BUMP ADDRESS
0C08 3E00               MVI   A,0
0C0A 77                 MOV   M,A            ;   STORE ZERO
0C0B 23                 INX   H              ;BUMP THE ADDRESS
0C0C 78                 MOV   A,B            ;       GET LETTER
0C0D C601               ADI   1              ;CHANGE TO NEXT LETTER
0C0F 47                 MOV   B,A            ;   SAVE IN B
0C10 C3030C             JMP   NV             ;NEXT VARIABLE
                        ;
                        ; SAVE RETURN POINTER IN STACK
                        ;     ENTER:   IPTR = START OF NEXT SOURCE LINE
                        ;     RETURNS: LEVEL BUMPED ONE HIGHER
                        ;              IPTR COPIED AT LEVEL POSITION
                        ;
0C13 214501     SAVRET: LXI   H,LEVEL        ;GET CURRENT LEVEL
0C16 7E                 MOV   A,M
0C17 C601               ADI   1              ;BUMP TO NEXT LEVEL
0C19 FE08               CPI   8              ;IF < 8
0C1B FA250C             JM    SAV2           ;  THEN CONTINUE
0C1E 21360E             LXI   H,STMSG        ;ELSE STACK OVERFLOW
0C21 CDDE0D             CALL  ERROR
0C24 C9                 RET
0C25 77         SAV2:   MOV   M,A            ;   STORE IT
0C26 87                 ADD   A              ;DOUBLE IT
0C27 4F                 MOV   C,A            ;       SAVE IN C
0C28 2A3A01             LHLD  IPTR           ;PUT IPTR IN DE
0C2B EB                 XCHG
0C2C 212401             LXI   H,RETSAV       ;  GET BASE ADDR
0C2F 7D                 MOV   A,L
0C30 81                 ADD   C              ;BASE + 2 X LEVEL
0C31 6F                 MOV   L,A            ;       HL = STACK ADDR
0C32 72                 MOV   M,D            ;   SAVE IPTR
0C33 C601               ADI   1
0C35 6F                 MOV   L,A            ;       HL = STACK ADDR+1
0C36 73                 MOV   M,E
0C37 C9                 RET
                        ;
                        ; RESET RETURN POINTER FROM STACK
                        ;     RETURNS: IPTR SET TO LAST SAVED RETURN
                        ;              LEVEL REDUCED BY ONE
                        ;
```

```
0C38 3A4501   RESRET: LDA   LEVEL          ;GET RETURN LEVEL
0C3B 87               ADD   A              ;DOUBLE IT
0C3C 4F               MOV   C,A            ;       SAVE IN C
0C3D 212401           LXI   H,RETSAV       ;  GET BASE ADDR
0C40 7D               MOV   A,L
0C41 81               ADD   C              ;BASE + 2 X LEVEL
0C42 6F               MOV   L,A            ;       HL = STACK ADDR
0C43 56               MOV   D,M            ;    SAVE POINTER IN DE
0C44 C601             ADI   1
0C46 6F               MOV   L,A            ;       HL = STACK ADDR+1
0C47 5E               MOV   E,M
0C48 EB               XCHG
0C49 223A01           SHLD  IPTR           ;RESET IPTR
0C4C 214501           LXI   H,LEVEL        ;REDUCE LEVEL
0C4F 7E               MOV   A,M
0C50 D601             SUI   1
0C52 77               MOV   M,A
0C53 C9               RET
                ;
                ; BLOCK TRANSFER FROM HL TO DE, C CHARACTERS
                ;   RETURNS: HL AND DE AT LAST CHAR+1 ADDR
                ;
0C54 79       BLKTFR: MOV   A,C            ;    GET COUNT
0C55 B7               ORA   A              ; IF COUNT = 0
0C56 C8               RZ                   ;  THEN RETURN
0C57 46               MOV   B,M            ;    GET A SOURCE CHARACTER
0C58 23               INX   H              ;GET NEXT DEST ADDR
0C59 EB               XCHG
0C5A 70               MOV   M,B            ;    PUT IT IN DEST LOCATION
0C5B 23               INX   H              ;GET NEXT SOURCE ADDR
0C5C EB               XCHG
0C5D 79               MOV   A,C            ;    DECREMENT COUNT
0C5E D601             SUI   1
0C60 4F               MOV   C,A
0C61 C2540C           JNZ   BLKTFR         ;IF NONZERO THEN NEXT
0C64 C9               RET
                ;
                ; KEYBOARD INPUT TO EBUFF
                ;   ENTER:  CHMAX= MAXIMUM CHARS ALLOWED IN LINE
                ;   DEL (SHIFT O) OR RUBOUT CANCELS LAST CHAR
                ;   CTL/U CANCELS CURRENT LINE
                ;   USES B FOR CHAR COUNT, C FOR OUTPUT
                ;
0C65 219D01   KEYIN:  LXI   H,EBUFF        ;POINT AT EBUFF
0C68 223601           SHLD  EPTR           ;SAVE POINTER
0C6B 214401           LXI   H,CHMAX        ;GET MAX COUNT
0C6E 46               MOV   B,M
0C6F 2A3601   KIN2:   LHLD  EPTR           ;RETRIEVE POINTER
0C72 CD0001           CALL  CI             ;GET CHAR AND ECHO
0C75 FE5F             CPI   5FH            ;IF DEL
0C77 CA800C           JZ    CANC           ;  THEN CANCEL LAST CHAR
0C7A FE7F             CPI   7FH            ;IF RUBOUT
0C7C CA800C           JZ    CANC           ;  THEN CANCEL LAST CHAR
0C7F FE15             CPI   15H            ;IF CTL/U
0C81 CAA50C           JZ    CANL           ;  THEN CANCEL LINE
0C84 FE61             CPI   61H            ;FORCE UPPER CASE
0C86 FA8B0C           JM    NTR
0C89 EE20             XRI   20H
0C8B 77       NTR:    MOV   M,A            ;STORE THE CHAR
0C8C 23               INX   H              ;INCR POINTER
0C8D 223601           SHLD  EPTR           ;SAVE IT
0C90 FE0D             CPI   0DH            ;IF CR
0C92 CAA10C           JZ    KOUT           ;  THEN STOP ENTRY
0C95 05               DCR   B              ;ELSE DECR CHAR COUNT
0C96 78               MOV   A,B
0C97 B7               ORA   A              ;IF COUNT NOT 0
0C98 C26F0C           JNZ   KIN2           ;  THEN NEXT CHAR
0C9B 0E0D             MVI   C,0DH          ;ELSE END WITH CR
0C9D 71               MOV   M,C
0C9E CD0301           CALL  CO             ;  AND SEND IT
0CA1 CDC90C   KOUT:   CALL  LF             ;SEND LINE FEED
0CA4 C9               RET                  ;  AND RETURN
0CA5 0E3C     CANL:   MVI   C,3CH          ;SEND <
0CA7 CD0301           CALL  CO
0CAA CDC40C           CALL  CRLF           ;SEND CRLF
0CAD C3650C           JMP   KEYIN          ;  START OVER
0CB0 78       CANC:   MOV   A,B            ;INCR CHAR COUNT
0CB1 214401           LXI   H,CHMAX        ;  UNLESS AT BEGINNING
0CB4 4E               MOV   C,M
0CB5 B9               CMP   C
0CB6 CA6F0C           JZ    KIN2
0CB9 04               INR   B
0CBA 2A3601           LHLD  EPTR
0CBD 2B               DCX   H              ;DECR POINTER
0CBE 223601           SHLD  EPTR
0CC1 C3720C           JMP   KIN2+3
                ;
0CC4 0E0D     CRLF:   MVI   C,0DH
0CC6 CD0301           CALL  CO
0CC9 0E0A     LF:     MVI   C,0AH
0CCB CD0301           CALL  CO
0CCE C9               RET
                ;
                ; INPUT PROGRAM TO BUFFER AREA
                ;   DEL (SHIFT O) CANCELS LAST CHAR, CTL/U CANCELS LINE
                ;   TERMINATES WITH CTL/Z (1AH)
                ;
0CCF CD1806   INPUT:  CALL  BLKBF          ;BLANK THE BUFFER
0CD2 210010           LXI   H,PBUFF        ;SET POINTER
0CD5 7E       INPT1:  MOV   A,M            ;GET EXISTING CHAR
0CD6 222201           SHLD  LLSAV          ;SAVE FIRST CHAR ADDR
0CD9 FE01             CPI   1              ;IF END MARK
0CDB CA2F0D           JZ    CHOP           ;  THEN CHOP ENTRY
0CDE CD0601           CALL  RI             ;GET CHAR
0CE1 FE20             CPI   ' '            ;IF NOT BLANK
0CE3 C2F20C           JNZ   INPT3+3        ;  THEN CONTINUE
0CE6 C3D50C           JMP   INPT1          ;ELSE SKIP LEADING BLANK
0CE9 7E       INPT2:  MOV   A,M            ;GET EXISTING CHAR
0CEA FE01             CPI   1              ;IF END MARK
0CEC CA2F0D           JZ    CHOP           ;  THEN CHOP ENTRY
0CEF CD0601   INPT3:  CALL  RI             ;GET CHARACTER
0CF2 FE00             CPI   0              ;IGNORE NULLS
0CF4 CAEF0C           JZ    INPT3
0CF7 FE7F             CPI   7FH            ;IGNORE RUBOUTS
0CF9 CAEF0C           JZ    INPT3
0CFC FE1A             CPI   1AH            ;IF TERM CHAR CTL/Z
0CFE CA3B0D           JZ    INEND          ;  THEN END OF INPUT
0D01 FE15             CPI   15H            ;IF CTL/U
0D03 CA210D           JZ    KLN            ;  THEN KILL THE LINE
0D06 FE5F             CPI   5FH            ;IF DEL
0D08 CA1D0D           JZ    CLC            ;  THEN CANCEL LAST CHAR
0D0B 77               MOV   M,A            ;ELSE STORE THE CHAR
0D0C 23               INX   H              ;AND INCR THE POINTER
0D0D FE0D             CPI   0DH            ;IF NOT CR
0D0F C2E90C           JNZ   INPT2          ;  THEN GET NEXT CHAR
0D12 CDC90C           CALL  LF             ;ELSE SEND LF
0D15 C3D50C           JMP   INPT1          ;AND GET NEXT NEW LINE
0D18 77       INPT4:  MOV   M,A            ;STORE CHAR
0D19 23               INX   H              ;INCR POINTER
0D1A C3E90C           JMP   INPT2
0D1D 2B       CLC:    DCX   H              ;CANCEL LAST CHAR
0D1E C3EF0C           JMP   INPT3
```

```
0D21 0E3C     KLN:    MVI   C,3CH          ;SEND <
0D23 CD0301           CALL  CO
0D26 CDC40C           CALL  CRLF           ;SEND CRLF
0D29 2A2201           LHLD  LLSAV          ;ADDR FRONT OF LINE
0D2C C3EF0C           JMP   INPT3
0D2F 2A2201   CHOP:   LHLD  LLSAV          ;ADDR FRONT OF LINE
0D32 CD490D           CALL  DSPLY
0D35 21F50D           LXI   H,IOVMSG
0D38 CDDE0D           CALL  ERROR
0D3B 3601     INEND:  MVI   M,1            ;STORE END MARK
0D3D 221801           SHLD  TOPP           ;STORE ADDRESS
0D40 CDC40C           CALL  CRLF           ;SEND CRLF
0D43 214501           LXI   H,LEVEL        ;ZERO RETURN LEVEL
0D46 3600             MVI   M,0
0D48 C9               RET
                ;
                ; DISPLAY A CHARACTER STRING TO CR OR 01
                ;   ENTER:  HL = STARTING ADDRESS
                ;   OUTADR CONTAINS ADDRESS OF CO, LO, OR PO
                ;
0D49 7E       DSPLY:  MOV   A,M            ;GET A CHARACTER
0D4A 23               INX   H              ;BUMP POINTER
0D4B 54               MOV   D,H            ;SAVE IT
0D4C 5D               MOV   E,L
0D4D FE01             CPI   1              ;IF 01
0D4F C8               RZ                   ;  THEN RETURN
0D50 4F               MOV   C,A            ;PUT CHAR IN C
0D51 2A4001           LHLD  OUTADR         ;MAKE AN INDIRECT CALL
0D54 CD6D0D           CALL  OVCTR          ;  TO SEND THE CHAR
0D57 FE0D             CPI   0DH            ;IF CR
0D59 CA600D           JZ    ENDOL          ;  THEN EOL
0D5C EB               XCHG                 ;ELSE RETRIEVE POINTER
0D5D C3490D           JMP   DSPLY          ;AND DISPLAY MORE
0D60 0E0A     ENDOL:  MVI   C,0AH          ;SEND LINE FEED
0D62 2A4001           LHLD  OUTADR         ;MAKE AN INDIRECT CALL
0D65 CD6D0D           CALL  OVCTR          ;  TO SEND IT
0D68 EB               XCHG                 ;RETRIEVE POINTER
0D69 CD7A04           CALL  SKLN           ;SKIP ANY LINE NOS.
0D6C C9               RET
0D6D E9       OVCTR:  PCHL
                ;
                ; OUTPUT PROGRAM IN MEMORY TO 01 END MARK
                ;
0D6E 210010   PRGOUT: LXI   H,PBUFF        ;PGM START ADDR
0D71 CD490D           CALL  DSPLY          ;DISPLAY ONE LINE
0D74 FE01             CPI   1              ;IF NOT END MARK
0D76 C2710D           JNZ   PRGOUT+3       ;  THEN MORE
0D79 C9               RET
                ;
0D7A 210301   DPRG:   LXI   H,CO           ;DISPLAY PROGRAM IN MEMORY
0D7D 224001           SHLD  OUTADR
0D80 CD6E0D           CALL  PRGOUT
0D83 C9               RET
                ;
0D84 210901   LPRG:   LXI   H,LO           ;LIST PROGRAM IN MEMORY
0D87 224001           SHLD  OUTADR
0D8A CD6E0D           CALL  PRGOUT
0D8D 210301           LXI   H,CO           ;RESET TO CONSOLE
0D90 224001           SHLD  OUTADR
0D93 C9               RET
                ;
0D94 210C01   SPRG:   LXI   H,PO           ;SAVE PROGRAM
0D97 224001           SHLD  OUTADR
0D9A CD6E0D           CALL  PRGOUT
0D9D 210301           LXI   H,CO           ;RESET TO CONSOLE
0DA0 224001           SHLD  OUTADR
0DA3 C9               RET
                ;
                ; I/O ROUTINES
                ;     USE PORT 0 FOR STATUS, PORT 1 FOR DATA
                ;     STATUS BIT 6 (40H) FOR READ DATA AVAIL (RDA)
                ;     BIT 7 (80H) FOR TRANSMIT BUFFER EMPTY (TBE)
                ;
                ; OUTPUT CHAR FROM C
                ;   LOOKS FOR CTL/Z INPUT FOR PANIC EXIT
                ;
0DA4 DB00     CHO:    IN    0              ;GET STATUS
0DA6 E640             ANI   40H            ;IF NO INPUT
0DA8 CAB30D           JZ    CHO1           ;  THEN CONTINUE
0DAB CD0001           CALL  CI             ;ELSE SEE WHAT IT IS
0DAE FE1A             CPI   1AH            ;IF CTL/Z
0DB0 CABE0D           JZ    INTR           ;  THEN INTERRUPT
0DB3 DB00     CHO1:   IN    0              ;NOW FOR STANDARD OUTPUT
0DB5 E680             ANI   80H
0DB7 CAB30D           JZ    CHO1
0DBA 79               MOV   A,C
0DBB D301             OUT   1
0DBD C9               RET
                ;
0DBE E5       INTR:   PUSH  H              ;SAVE REGISTERS
0DBF D5               PUSH  D
0DC0 C5               PUSH  B
0DC1 21650E           LXI   H,INTMSG       ;INTERRUPT MESSAGE
0DC4 CDDE0D           CALL  ERROR
0DC7 CD8809           CALL  ENTR           ;ALLOW RESTART
0DCA C1               POP   B              ;ELSE CONTINUE
0DCB D1               POP   D
0DCC E1               POP   H
0DCD C9               RET
                ;
                ; INPUT CHAR TO A AND ECHO
                ;
0DCE DB00     CHI:    IN    0              ;NORMAL INPUT
0DD0 E640             ANI   40H
0DD2 CACE0D           JZ    CHI
0DD5 DB01             IN    1
0DD7 E67F             ANI   7FH
0DD9 4F               MOV   C,A
0DDA CD0301           CALL  CO
0DDD C9               RET
                ;
                ; ERROR - DISPLAYS ERROR MESSAGE
                ;    ENTER:  HL = ADDRESS OF MESSAGE
                ;    RETURNS: ZERO FLAG SET
                ;
0DDE CD490D   ERROR:  CALL  DSPLY
0DE1 AF               XRA   A
0DE2 C9               RET
                ;
                ; ERROR MESSAGES
                ;
0DE3 2D204C41 BLMSG:  DB    '- LABEL NOT FOUND',0DH
0DE7 42454C20
0DEB 4E4F5420
0DEF 464F554E
0DF3 440D
                ;
0DF5 2F4F5645 IOVMSG: DB    '/OVERFLOW',0DH
0DF9 52464C4F
0DFD 570D
                ;
0DFF 2A4E4F20 NRMSG:  DB    '*NO ROOM',0DH
```

```
0E03 524F4F4D
0E07 0D
                ;
 0E08 2A494C4C EXPMSG: DB     '*ILLEGAL EXPRESSION',0DH
0E0C 4547414C
0E10 20455850
0E14 52455353
0E18 494F4E0D
                ;
 0E1C 2A56414C OVFMSG: DB     '*VALUE >  99',0DH
0E20 5545203E
0E24 20203939
0E28 0D
                ;
 0E29 2A56414C UNFMSG: DB     '*VALUE < -99',0DH
0E2D 5545203C
0E31 202D3939
0E35 0D
                ;
 0E36 2A555345 STMSG:  DB     '*USE DEPTH EXCEEDED',0DH
0E3A 20444550
0E3E 54482045
0E42 58434545
0E46 4445440D
                ;
 0E4A 2A4E554D NMSG:   DB     '*NUMERIC RESPONSE REQUIRED',0DH
0E4E 45524943
0E52 20524553
0E56 504F4E53
0E5A 45205245
0E5E 51554952
0E62 45440D
                ;
 0E65 2A494E54 INTMSG: DB     '*INTERRUPTED',0DH
0E69 45525255
0E6D 50544544
0E71 0D
                ;
 0E72 543A0D   IBUFF:  DB     'T:',0DH
 0E75 543A5049         DB     'T:PILOT-8080, 1.1',0DH
0E79 4C4F542D
0E7D 38303830
0E81 2C20312E
0E85 310D
                ;
 0E87 3A4C4F41         DB     ':LOAD A NEW PROGRAM?',0DH
0E8B 44204120
0E8F 4E455720
0E93 50524F47
0E97 52414D3F
0E9B 0D
                ;
 0E9C 413A0D           DB     'A:',0DH
                ;
 0E9F 4D3A2059         DB     'M: Y',0DH
0EA3 0D
                ;
 0EA4 4A4E3A2A         DB     'JN:*%',0DH
0EA8 250D
                ;
 0EAA 543A454E         DB     'T:ENTER PILOT PROGRAM',0DH
0EAE 54455220
0EB2 50494C4F
0EB6 54205052
0EBA 4F475241
0EBE 4D0D
                ;
 0EC0 3A544552         DB     ':TERMINATE INPUT WITH CTL/Z',0DH
0EC4 4D494E41
0EC8 54452049
0ECC 4E505554
0ED0 20574954
0ED4 48204354
0ED8 4C2F5A0D
                ;
 0EDC 4C4F4144         DB     'LOAD:',0DH
0EE0 3A0D
                ;
 0EE2 2A252049         DB     '*% IEP:',0DH
0EE6 45503A0D
                ;
                ; ORG HERE CAN SET START OF RAM PROGRAM BUFFER SPACE
                ;
 1000                  ORG    PBUFB
 1000 01       PBUFF:  DB     1
                ;
                ; SOURCE PROGRAM AND $STRING STORAGE HERE TO MEMTP
 0000                  END
```

NEWS FROM COLA

The first meeting of the Computer Organization of Los Angeles (COLA) for the 1976-77 academic year was held on October 14 at the Pacifica Hotel in Culver City. Following a testimonial dinner honoring Mr. Art Freier, Mathematics Supervisor for the Los Angeles School District, Prof. Richard Ballard of the University of California at Irvine gave a fascinating slide presentation on "Computers and the Future of Learning."

An enthusiastic and highly innovatibe proponent of computers in education, Prof. Ballard outlined some of the current uses of computers at UCI and some possible applications of techniques now under development.

If the present trend toward larger classes and smaller budgets continues in American universities, computers may soon become the only cost-effective way of evaluating student performance. For example, at UCI all Physics examinations at the undergraduate level are computerized.

Using a Xerox Sigma 7 timesharing system and other computers, UCI provides highly sophisticated computer services to both students and faculty. High-speed video terminals facilitate extensive use of coordinated text and graphics displays, and hard copy is available to students via coin-operated non-impact printers.

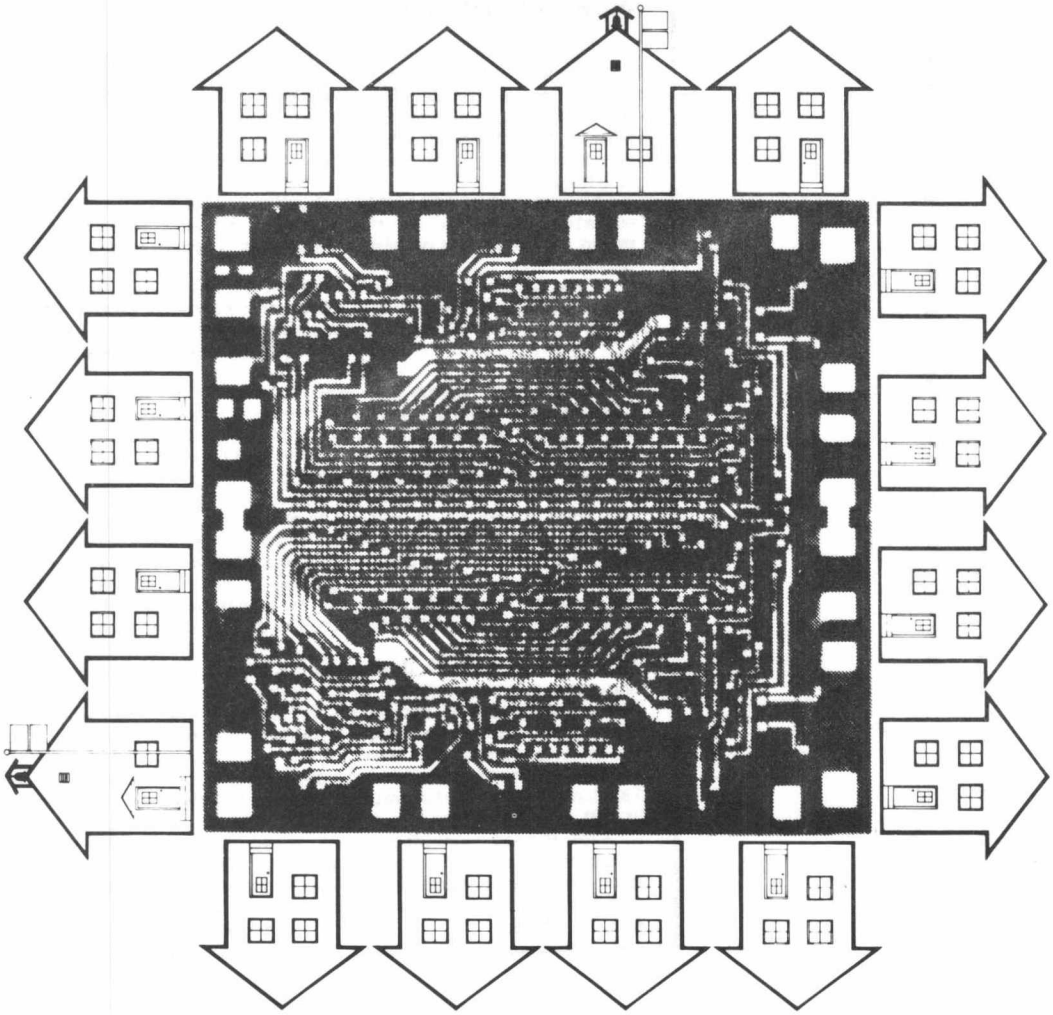
One of the exciting new developments at UCI is an incredibly versatile program that allows students to set up and perform an almost infinite variety of simulated physical experiments. Over 150,000 distinct functions are provided, and the progress of each simulation is monitored on-line by a graphic display of all of the variables requested by the student. Examples cited were the display of electric field intensity plots under varying conditions, and the display of planetary orbits in multiple star systems.

Another remarkable development is a CAI technique giving the student almost total freedom in his interaction with the computer. At any point the student can change the subject within reasonable limits, request clarification of a specific point, or ask the computer to cover the material in greater or lesser depth.

To implement the extensive graphics capabilities of these programs, UCI has added powerful graphics functions to high-level languages such as APL and BASIC.

Prof. Ballard concluded his talk by saying that presently available APL interpreters are rather slow for some CAI applications, but speculated that a highly parallel APL multiprocessor might be several orders of magnitude faster than any machine in use today.

Jim Day
Oct. 15, 1976



The mention of low-cost computers usually evokes one of two images. Some of us see a super calculator; others picture a large data-base processor. The system described here is a more modest machine that could sell for under $500 in the relatively near future. Not much has been written on practical computers of this size. Nevertheless, prototypes of this low-cost, mass-market, free-standing computer system have been constructed, programmed, and operated in a home environment over the past several years.

A PRACTICAL, LOW-COST, HOME/SCHOOL MICROPROCESSOR SYSTEM

Joe Weisbecker, RCA Laboratories

Reprinted from the IEEE Computer Society's **Computer** *magazine, August 1974 issue, with permission of the IEEE and the author.*

Jim-- 8/76

My ideas have changed somewhat since I wrote this article. I think such a system could be in the under-$200 range today, but the philosophy is still sound as far as I'm concerned.

Joe Weisbecker

EDITOR'S NOTE: Joe was the designer of RCA's COSMAC.

Meet Fred

Despite the recreational and educational potential of stored-program computers, the single factor of cost has kept them out of the economic reach of most people. But with the advent of LSI microprocessor and memory chips, this may all change – particularly if we take a more modest applications approach and place reasonable limitations on hardware capability.

Of course, if we need conventional input, output, and bulk storage devices, or if our applications are playing chess, printing pages of data, or accessing large on-line digital/video data bases, the cost will still be prohibitive. However, prototypes of such modest home/school systems have

been constructed and in use for several years.

This system, called FRED (Flexible Recreational and Educational Device), has been developed using the RCA COSMAC microprocessor.

A computer of this type could have major social value. As an interactive, open-ended, adaptive, recreational and educational device, it could stimulate the development of analytical and other intellectual abilities. One can easily imagine the formation of a whole new group of computer hobbyists, complete with user groups and publications for the exchange of programs and ideas. In short, the inexpensive home/school computer could open the door to an entirely new environment that stimulates experimentation, analysis, and creativity.

Application and System Overview

In schools, FRED could provide a powerful educational tool. It could be used to drill and test students from first grade on. It could be used in educational games, simulation exercises, and reading readiness, as well as in teaching programming, as an adjunct to math courses, and as an accessible student tool in almost any subject. FRED could be used to set up stimulating demonstrations and experiments in a wide variety of areas, to help correct learning disabilities, and to stimulate the development of creative abilities. Cost per student hour would be measured in pennies.

In the home, FRED has already functioned as a sophisticated entertainment center for the whole family. It provides a variety of games, simulates a calculator, and even provides a controllable TV puppet for the youngest member of the family. FRED permits a number of creative activities including TV picture drawing, low-fidelity music synthesis, and programming at a variety of skill levels. FRED also provides a shooting gallery, a variety of puzzles, and animated TV greeting cards for holidays.

Since FRED is a stored-program computer, it requires a program to be loaded into memory before use. Program loading is performed with an inexpensive audio cassette player which also gives the computer its voice, music, and sound effect capabilities. Prerecorded program cassettes can be loaded in less than 30 seconds.

After a program cassette is selected and loaded, FRED is operated with a small 16-position keyboard. For a game, the player presses appropriate keys to indicate the moves. Overlay cards are provided so that keyboard labeling can be changed for different programs.

FRED is attached to the antenna terminals of any TV set. This provides an inexpensive, flexible, dynamic output display which is ideally suited for home/school use. Numbers, words, or simple pictures can be displayed on the TV screen in the form of dot patterns.

The basic FRED system comprises the RCA COSMAC microprocessor, 1024 bytes of RAM, a simple hex keyboard, an inexpensive audio cassette player, and the user's own TV set. One would be hard-pressed to imagine a less expensive free-standing computer system. This system is supported by a library of cassette programs in the same way that a phonograph is supported by a record library. A continuing supply of new programs could be provided by the manufacturer of the system together with a selection of optional hardware attachments.

Adding a $25 punched card reader and $10 manual punch to the basic system increases its usefulness and provides more sophisticated users with the ability to prepare and save short parameter lists or programs. Adding a module for recording the contents of memory on cassettes turns the basic FRED system into a user-programmable computer for serious hobbyists. Other possible attachments include light guns, extra memory (RAM), pre-stored programs or tables (ROM), and output relays for control uses.

Design Considerations

Two different approaches can be taken toward developing an under-$500 home/school computer. One approach involves specifying a desired set of system characteristics and then attempting to achieve cost objectives. The danger in this approach is the tendency to overspecify the hardware so that price targets can't be met. The approach we took in developing FRED was to define a minimum cost, nontrivial hardware system that could easily meet our low price goal, and then testing its usefulness. This approach ensures that applications development effort won't be wasted. Applications developed for our minimum system are easily transferred to a larger system.

Any free-standing computer must include a CPU, main memory, input and output device(s), and bulk storage device(s).

The FRED system required the development of a system philosophy that was consistent with utilizing minimum cost hardware. This system philosophy included asking what we could do with cheap devices instead of asking what types of devices we might ideally want. Because of our low price goal, the implications of each of our choices were magnified in importance.

For our purposes, competitive cost-performance ratios were largely ignored. Reliability was also sacrificed to some extent to achieve low cost. Of course, permanent system failures cannot be tolerated, but occasional transient errors, provided they are not catastrophic, may be permitted during program loading. The need for memory and processor parity checking was also felt to be an unaffordable luxury.

Ease of use is a primary requirement. So that the system would not appear overly complex to the novice user, a turn-key system philosophy was adopted. The user simply loads a program from a library to obtain a desired function. He is not expected to program the machine. This approach eliminates the need for program debugging and maximizes ease of use. The need for an expensive control and diagnostic panel is also eliminated. In fact, only two switches are required for basic use: LOAD and RUN. Utility routines and optional hardware are provided for the sophisticated user who wants to develop his own machine language programs. The minimum system also provides for small user-generated programs via a variety of simulation languages.

A block diagram of the basic system is shown in Figure 1. System considerations will be included in the discus-

sion of individual elements.

Central Processing Unit and Memory

As mentioned earlier, the advent of single chip LSI microprocessors has made the under-$500 system possible. Suitable microprocessor chips should be available for less than $25 within the next several years. The choice of a microprocessor has a large influence on total system chip count and system cost. This influence is an important consideration since the microprocessor itself is only a small part of total system cost. The COSMAC architecture immediately eliminates the need for a read only memory (ROM) in the minimum system. Only one supply voltage is required. COS/MOS circuits further reduce system power supply costs. A self-contained direct memory access (DMA) channel facilities initial program loading and display refresh. A single-phase clock is another minimum cost feature. High output drive capability eliminates external buffer circuits.

The 8-bit COSMAC architecture is compatible with the intended uses of the system. The short instruction format permits compact programs with small memory requirements. Since the average user will never see the processor micro-instruction set, ease of programming is secondary to efficient memory utilization.

A complete description of the COSMAC microprocessor has appeared previously[1,2] and will not be repeated here. This architecture has demonstrated its advantages in prototypes of the low cost home/school system.

Due to the nature of our application, RAM is required for both program and data storage. It's well known that programs tend to expand to fill available memory space. Providing a 4096 byte memory only insures that no program will be written requiring a smaller memory. Even projecting a cost of 2¢ per byte would yield a cost of $82 for a 4096 byte memory. This size memory would add $200 or more to the selling price of the system. Instead of asking how much RAM we could use, we provide 1024 bytes in the minimum system. This is consistent with keeping memory cost equal to projected microprocessor chip cost. Should LSI memory costs drop below 2¢ per byte we can increase minimum system capacity to 2048 bytes or lower the price of the 1024-byte system. Based on current trends, we can safely predict one microsecond LSI RAM costs of 2 to 3¢ per byte. Dynamic RAM chips are at this cost level now, while static, single voltage RAM chips are currently available at 7 to 8¢ per byte.

The challenge of a 1024-byte memory seems to stimulate cleverness in programming and makes a future 2048-byte memory seem large by comparison. If we had initially provided a 4096-byte memory, subsequent size reduction to meet cost targets would have been extremely difficult. Assuming 4x1024 bit RAM chips are available within the next several years, a minimum system would require only two chips for memory.

Limiting the minimum system memory to 1024 bytes also provides several system cost advantages. Power supply cost is reduced, memory address drivers are eliminated, and printed circuit board space is saved. A less obvious system implication is the effect of memory size on program loading costs.

In general, the user should be able to load a program in half a minute or so. This coincides with observed user-patience factors.

An occasional error requiring reload can be tolerated for short load times, so that lower reliability loading devices may be used. To load a 1024-byte memory in 30 seconds only requires a serial transfer rate of 300 bits/second. This assumes a parity bit for each byte. For a 4096-byte memory, the required rate jumps to 1200 bits/second. The required transfer rate influences the choice of a program loading technique. Lower rates can generally be translated into lower costs and better reliability.

It's in the area of input, output, and bulk storage that we encounter the major cost problems. The choice of I/O and bulk storage techniques also has a major effect on the

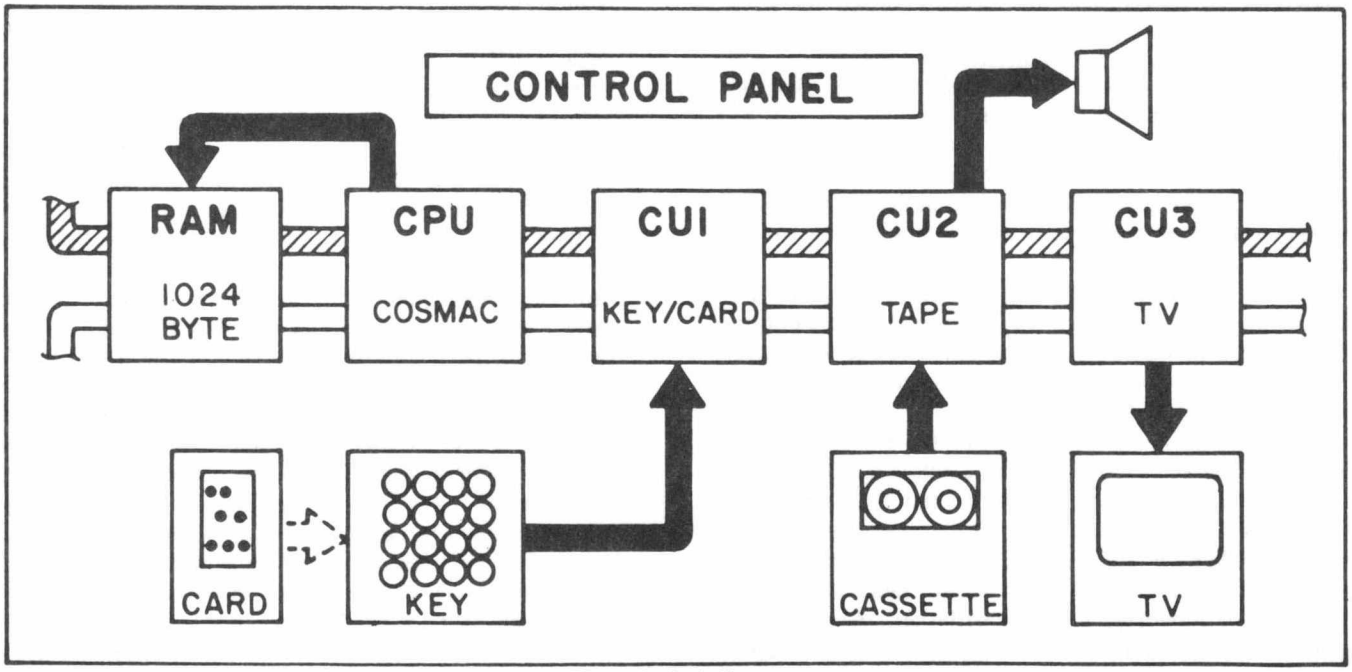


Figure 1. Basic System

range of possible system applications. Obviously, a single switch input and single light output would achieve minimum cost but would also result in a trivial system relative to use.

Output Display

Fortunately, an ideal, low-cost output device for home/school applications already exists: a standard TV set provides a flexible, dynamic output display device which most users already own.

The choice of a TV display format involves a number of system considerations. These include types of applications, display refresh memory requirements, and complexity of control circuits. A low resolution, black and white dot matrix was chosen for maximum flexibility at minimum cost. An array of white dots is displayed on a black background. The black background avoids potential picture noise problems. Arrays of 32x32, 16x64, and 32x64 dots are provided. Figure 2 illustrates the flexibility of this format for displaying small game boards, simple pictures, words, numbers, or symbols. Each dot represents the state of a main memory bit. If the bit is "1" the dot is on; if the bit is "Ø" the dot is off.

Changing the memory bit pattern immediately changes the TV picture accordingly. Bit patterns are readily moved in and out of the displayed memory area by normal programming procedures. Simple animation can be achieved by modifying memory bit patterns at appropriate time intervals. Any contiguous 128 or 256-byte section of memory can be selected for display by setting a microprocessor address pointer. This display pointer can be modified at any point in a program, thus allowing the user to step through various memory display areas at any desired rate. It is easy to flash selected portions of a picture by alternating between two display areas in memory.

For 32x32 and 16x64 displays only 1024 bits (128 bytes) of memory are required for display refresh. This is only 12.5 percent of the minimum 1024 byte memory. The 32x64 display option utilizes 25 percent of the minimum system memory but provides a larger area picture when required. It is also useful in expanded memory systems. It should be emphasized that no ROM is required for TV display in the minimum system and that frame refresh storage is provided via main memory.

The TV control unit (CU3 in Figure 1) contains the circuits for generating TV sync signals and for requesting memory bytes via the COSMAC DMA channel as required for display refresh. The individual bits of each byte are used to generate a video signal. The composite sync and video signal modulates the output of a simple RF oscillator. This modulated RF can be applied to the antenna terminals of any standard TV set.

Figure 3 illustrates the detailed timing for displaying dots on the TV screen. A magnified view of four dots is shown. Each dot is two horizontal TV lines high with a two-line space between dots. An 8-byte row buffer is provided in the TV control unit. Each TV line time is 65 microseconds. During the two blank line times, between rows of dots, up to 8 bytes (64 bits) are retrieved from main memory and stored in the row buffer. During the next two TV line times the bits in the row buffer modulate the TV beam to display the proper dot row pattern. By spreading the dots as shown,

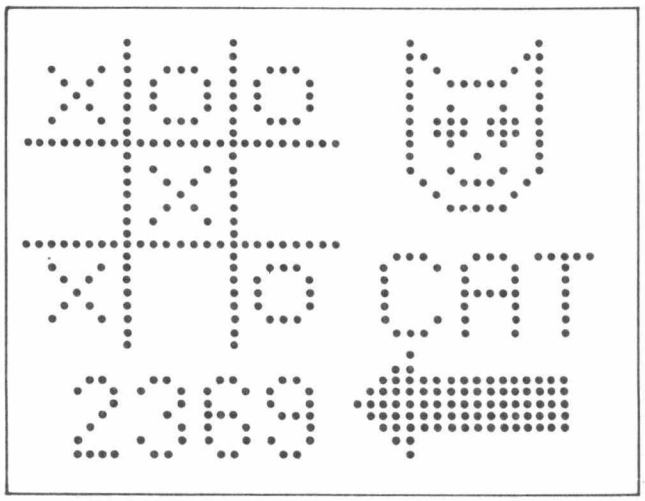


Figure 2. Display Flexibility

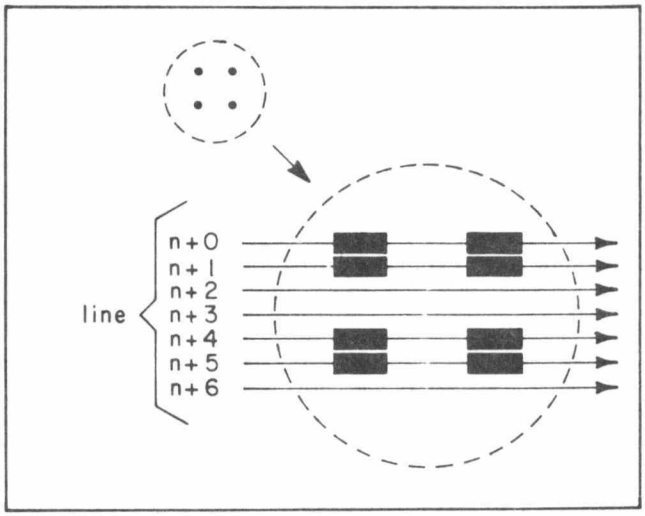


Figure 3. TV Dot Detail

the low resolution display fills up the TV screen without requiring a high refresh rate.

The TV control unit also generates a program interrupt signal at the beginning of each TV frame. This interrupt permits the program to initialize the microprocessor display address pointer at the appropriate time. Since TV program interrupt occurs 60 times/second, a free real-time clock exists when needed. This clock capability is useful for timing purposes in a number of applications.

Bulk Storage

Program library storage and loading presents another major problem area in a low cost system. The high cost of existing computer devices such as floppy discs and digital tape units immediately rule out their use. Paper tape is awkward and still fairly expensive. Conventional punched card readers are expensive and inconvenient.

This problem was solved by using another existing, inexpensive consumer device – the audio cassette recorder. Suitable portable units sell for under forty dollars. A built-in unit could be provided for less than $20. Several methods

for storing bit serial digital data on audio cassettes have been described[3,4] and others are possible. We developed a proprietary, pulse counting technique that yields a 50 byte per second transfer rate, tolerates missing or extra pulses, and permits tape speed variations of 30 percent. This system works well even for cheap portable audio units. Only single track capability is required in the system.

Since errors can be expected on occasion, a parity bit is added to each byte on tape. The cassette control unit checks the parity of input data read from tape and turns on an error light for incorrect parity. Reloading a program when an error occurs is a simple and quick procedure.

Figure 4 shows the single track, cassette tape format which was used. Digital or audio blocks are always framed by 4 kHz stop tones (T). The stop tone detection circuit is designed to respond only to long (.5 sec) continuous tones so that voice or music frames will not cause false triggers.

Figure 5 shows how a standard cassette player is used in the system. Most cassette recorders provide an external speaker or earphone output jack. This output is connected to the control unit as shown. Stop tones and digital data are detected via this cassette output line. A relay is also provided which permits the cassette output to be connected to a speaker under program control. This permits selected tape frames to be passed inaudibly.

The majority of inexpensive cassette recorders have a remote start-stop control jack. This is designed for use with a microphone or foot switch. For use in our system the cassette remote jack is connected to a program controlled relay. This gives the computer the ability to start and stop tape, providing the user has previously placed the cassette recorder in its PLAY mode.

The primary system operating controls comprise two toggle switches – LOAD and RUN. The LOAD switch activates the cassette control unit (CU2 in Fig. 1). The desired program cassette is selected by the user, rewound, and the recorder set to PLAY. When the first stop tone is encountered the data reading circuits are automatically turned on. Waiting for this stop tone eliminates possible noise problems at the beginning of tape. The digital data representing the program is loaded sequentially into memory at 50 bytes/second. The second stop tone automatically stops the tape via the tape control relay. Turning off the LOAD switch resets the computer. The RUN switch initiates execution of the program which was just loaded.

During program execution the tape can be automatically restarted so that the user will hear audio frame #1 at a

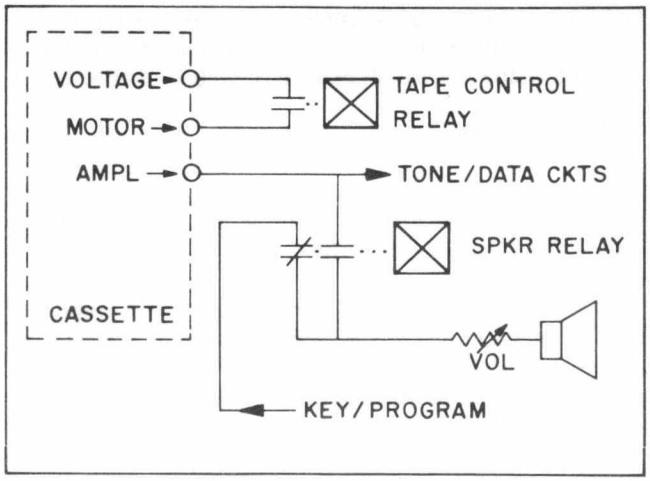


Figure 5. Cassette Attachment

desired time. The stop tone following audio frame #1 will automatically stop the tape. The program can monitor the state of the control relay to determine when the end of data/audio frames occur. This permits synchronizing audio material on cassettes with a program.

The provision for program controlled audio segments has an important system implication. The ability to provide instructions, questions, or other data in the form of voice frames on tape minimizes the need for a high resolution, alphanumeric TV display with its attendant requirement for large refresh and back-up digital storage capacity.

The speaker provided for use with the cassette recorder provides a useful output device. A flip-flop which can be set and reset by program drives the speaker when it's disconnected from the cassette output. Programs can, therefore, create many audible sequences of tones.

Input Devices

The primary input device for our system is a 16 position keyboard. A number of $5 to $10 keyboards of this type have been developed for use in pocket calculators. A flat, printed circuit type was chosen to facilitate an overlay feature. A slight modification of the keyboard permits insertion of a printed card above the switch array. Various cards are provided to relabel the switch array for different programs. For educational programs keys can be labeled

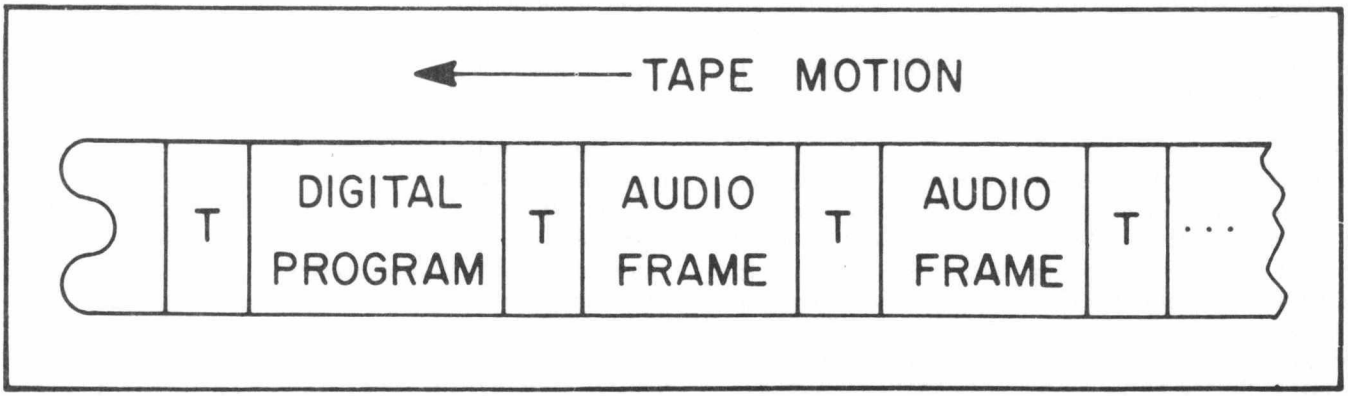


Figure 4. Cassette Tape Format

with colors, pictures, words, or possible answers to questions. For other programs, the keys might be labeled with direction arrows for manipulation of the TV display. The variable label keyboard is fundamental to meeting the ease-of-use criterion for this type of system.

Unfortunately, the flat keyboard which is ideal for variable labeling has no tactile feel. This objection was overcome by taking a systems approach. Since a speaker already exists, switch depressions need only be coupled into this speaker to provide an audible "click." This has proven to be an adequate substitute for tactile feel. The scanning approach used to decode the switch panel minimizes the cost of this approach. Specific programs can also generate various tones for switch depressions which again substitute for tactile feel.

The 16-position keyboard normally causes an 8-bit byte to be stored in memory for each key depression. The most significant four bits (digit) are normally ØØØØ. A shift switch pressed in conjunction with a hex key causes the most significant four bits to be ØØØ1. The least significant four bits of a stored byte represent the code for one of the 16 possible hex digits shown in Figure 6. The hex keyboard in conjunction with a shift switch permits entry of 32 different codes.

An alternate mode of keyboard entry is also provided. In this mode two key depressions per byte are required. The first key specifies the most significant hex digit of the byte to be entered. The second key provides the least significant hex digit of the byte. This mode provides the sophisticated user with a convenient way to manually load his own machine language programs. It's also a useful mode for certain turnkey programs.

The hex keyboard control unit (CU1 in Figure 1) also supports the addition of an inexpensive card reader to the minimum system. This unique device uses 3-inch x 5-inch punched cards. Data is punched in the form of rows of holes. Figure 7 shows four such rows (A,B,C,D) punched on one side of a card (both sides can be punched). Each row represents the 4-bit code for one hex digit. A fifth hole is added and encoded with odd parity. At least one hole will be punched for each of the possible hex digit codes. Cards are read by dropping them into a 3-inch slot. They fall past a light source and six photodiodes. One photodiode senses the presence of the card and conditions the control unit circuits accordingly. The other five photodiodes read the self clocking hex digit codes into the system. Hex digits are paired to form bytes before storage in memory.

By limiting the information content of a card to 16 hex digits per side, the mechanical tolerances of the reader can be considerably relaxed. The reader has no moving parts, and photodiodes can drive the COS/MOS control unit circuits directly. These factors combine to provide a very low cost input device ($25 or less). A simple, manual card punch can also be provided permitting users to punch their own cards.

The low cost card reader can be used to enter short lists of parameters or short user prepared programs. In a classroom, the teacher might use parameter cards to set up test/drill programs. Picture cards used by the student could contain the spelling of the word pictured for checking by the computer. The cards also facilitate certain simulation languages and permit users to save simulation language programs that they develop.

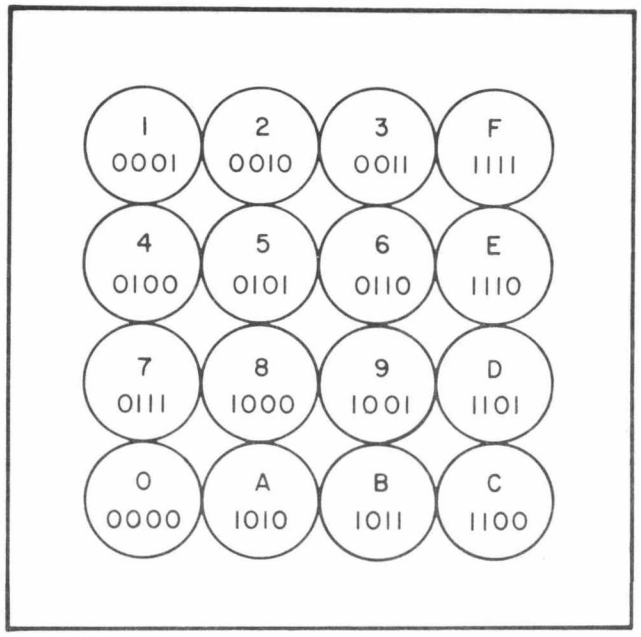


Figure 6. Keyboard Control

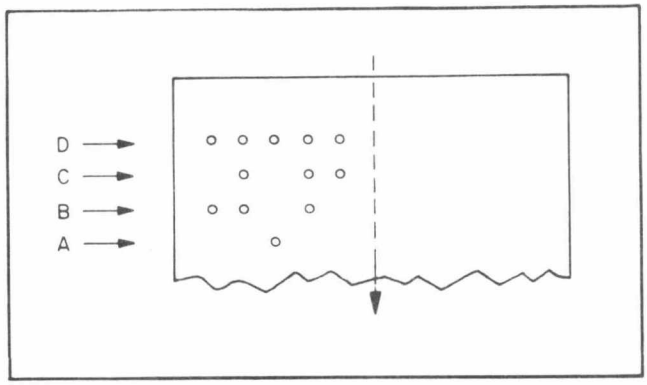


Figure 7. Punched Card

Another low cost, optional input device is a simple light gun. This contains a lens system and a photodiode. The computer detects when the gun is pointed at any lighted area of the TV screen. The light gun facilitates various computerized target shooting games. By alternately flashing portions of the TV display a program can determine the area at which the light gun is pointed. This permits the user to indicate various types of choices by pointing the gun at appropriate portions of the display.

Applications Philosophy

The open ended aspect of a stored program computer differentiates it from other types of recreational and edu-

cational devices. Any number of special purpose devices such as TV games, shuffleboard tables, electric football games, and educational toys are ideally suited to their intended function. None of these, however, will change their characteristics as user moods or interests change. Many of these special purpose devices are seldom used after their initial novelty expires. The stored program computer is a general purpose device. New programs can adapt it to changing moods and interests without the expense of new hardware. It can satisfy the needs of young and old and can grow with individual abilities.

The real value of the home/school system lies in its ability to stimulate and develop human capabilities that are often ignored or discouraged by conventional recreational and educational devices. The computer system provides an environment that stimulates experimentation, analysis, and creativity. For example, contemporary TV encourages passive viewing. However, the computer attached to a TV set enables the user to interact and play a game with the TV set. As the games played increase in sophistication, the user is encouraged to improve his analytical abilities. The user can subsequently be encouraged to experiment via specific programs or eventually to write his own programs.

For a child, the computer may initially provide arithmetic or spelling drills. Even this kind of memory development can be made more interesting via interaction with the computer. However, the child will eventually begin to wonder about the computer. Programs are made available which stimulate this curiosity and let him experiment with changing game rules. He can even begin to formulate and develop his own simple programs in a variety of simulation languages. While the initial use of the computer involves memory skills, it eventually encourages experimentation and the development of analytical and other capabilities.

The creation of programs that stimulate the user to develop mentally is a challenging task with a high payoff in terms of satisfaction. We have only begun to explore this area of use for very small, inexpensive, practical computers of the type described here. Even so, the number and richness of uses for this type of system are surprising. Those of us who are experienced with 64,000 byte main memories and large disc files may be inclined to dismiss a 1024 byte memory system as unuseable. But, in fact, such a system can be adapted to a wide range of uses. Over 80 specific applications of the inexpensive home/school system will be listed in the following sections. Many represent classes of programs which could be developed.

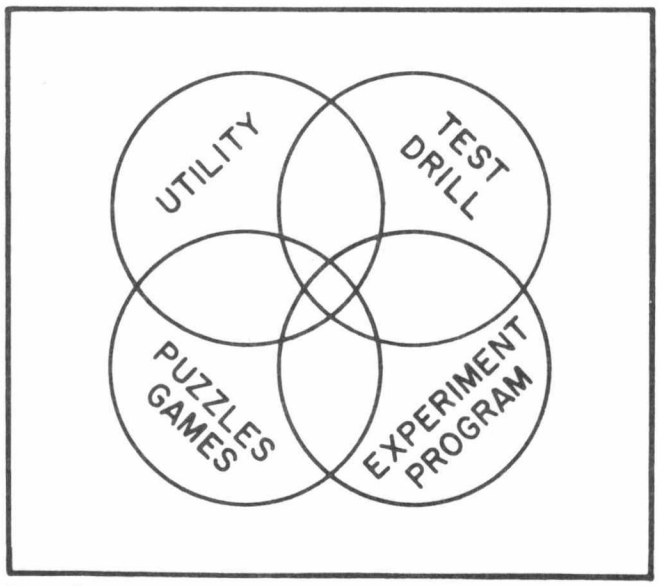


Figure 8. Areas of Use

Four general areas of use are identified in Figure 8. These areas will be discussed individually although there is a high degree of overlap between them. Most of the listed uses only require the basic system. Reference 18 also describes a number of uses (mostly games) that have been programmed on larger computers with hard copy output. Many of these are readily adapted to the low-cost computer.

Utility Applications

This category of applications involves use of the computer to achieve specialized functions such as those listed in Table 1.

*Four Function Decimal Calculator
Hex Binary Calculator
Game Score Keeper
*Number Base Converter
Weight/Measure Converter (Metric)
Secret Code Computer
Logic Machine[5]
Classification Computer
Gambling Strategy Computer
Other Specialized Calculators
(temperature conversion, interest, etc.)
Electronic Dice
Random Number Generator
Simulation Game Computer
Bar Graph
Interactive Audio-Visual Toy
*TV Greeting Card
*Electronic "Etch a Sketch"
TV Puppet
*Audio-Visual Demonstrator
Mind Reading Computer
Party Compatibility Computer
Programmed Timer/Controller
Stop Watch/Game Timer
Simple Electronic Organ
Metronome
Advertising Display

***Already developed for the COSMAC miniprocessor.**

Table 1. Utility Applications

A simulated four-function decimal calculator has been implemented on the basic 1024 byte memory system. This includes display refresh, digit pattern tables, and decimal arithmetic algorithms with 20-digit operand and result capability. A 2048 byte memory would permit development of a programmable calculator with multi-line display. Optional ROM chips could provide a permanently resident calculator capability if desired.

A variety of specialized calculators can be implemented on the basic system. Programs to provide scorekeeping for card, war, or commercial games could be provided. Children could have their own secret code computer. For several years a plastic toy rock identification computer has been on the market. Certain tests are performed (color, hardness, etc.) on a mineral sample. The plastic computer and a

set of cards is then used to identify the sample. The basic home/school system could readily be programmed as a classification computer of this type.

Logic machines have held a certain fascination for years.[5] The computer readily simulates a variety of machines of this type. It can also be programmed to simulate gambling algorithms. A pair of dice is easily simulated for use in a number of games. Random number generating machines find use in various school courses and experiments. Serious war game fans can use computer generated battle results and score keeping to advantage. The leading magazine in the field, "Strategy & Tactics," has over 20,000 subscribers indicating a wide interest in this type of activity.

For very young children the computer simulates a variety of interactive, audio-visual toys that make sounds and change TV pictures in response to key depressions. Customized, animated, TV Greeting Cards/Decorations for Birthdays, Christmas, or Halloween can be provided. Simple, key operated TV puppets are possible. Stepping a spot around the screen permits drawing TV pictures.

The ability to synchronize audio tape frames with programs permits programmed audio-visual tutorials for home and school or eye-catching advertising displays. The basic system real-time clock facilitates key operated game timer or stop watch capability. The program controlled speaker turns the computer into a simple electronic organ or metronome. TV display can be included with the sound generation.

Test and Drill Applications

These are probably the first types of uses that come to mind when education is mentioned. Drills involve the development of memory or conditioned reflexes. Testing can involve the development of other skills, as well. The infinite patience of a computer makes it ideal for drills. Interactive capability adds interest and motivation. Some specific examples are included in Table 2.

*TV Arithmetic Drill
*Word Spelling Drill
*Word Recognition Test
*Pattern Recognition (Superimposed, Complex)
Electronic Flash Cards
Classroom Group Games
Preschool Shape/Color Recognition
Up-Down, Left-Right Discrimination
Sound-Picture Matching
Reading Readiness Skill Drills
Logical Aptitude Test[6]
*Number Base Conversion Drill
Flap Board Simulator[7]
Morse Code Drill
Reflex Testing
*Logical Deduction Test (21 Questions)
Logidex[8]
Memory Training (Sobriety Test)
Individual Testing & Scoring Aid
Change Making Drill
X-Y Curve Plotting Drill
Time Sense Development

***Already developed for the COSMAC miniprocessor.**

Table 2. Test and Drill Applications

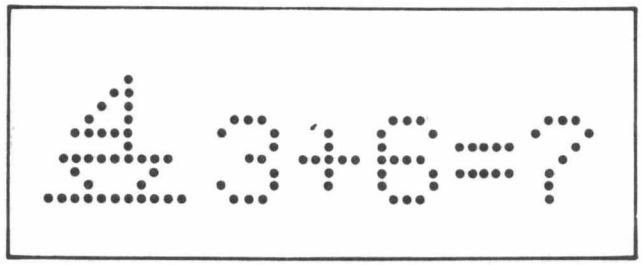


Figure 9. Add Drill Display

The number of different programs possible in this area is virtually unlimited. Programs of this type are ideal for individual use to overcome specific weaknesses. Unlike the teacher or parent, the computer does not make value judgements about the child during a drill. The drills can also be made to appear as games with the computer providing added motivation.

A simple arithmetic drill might appear as shown in Figure 9. Addition problems are randomly generated on the TV screen. The child must enter correct answers via the keyboard in time to prevent the boat from completely sinking. The rate at which the boat sinks can be preset by the teacher and changed from session to session to maintain challenge as the child's speed and accuracy improves. The computer displays the child's score when the teacher enters a special code (key/card).

Spelling drills can be implemented in several ways. A cassette voice could ask for the spelling of a word which the student then spells via the keyboard. The TV display and/or audio tone responds if he is right. The computer again keeps score and times the answers. A simple word recognition drill involves displaying a word on TV and asking for the corresponding picture via keyboard or card input. Patterns can be superimposed on the TV screen and the student asked to identify the components of the picture. Simple preschool shape, color, or sound recognition programs are possible. Up-down, left-right concepts are readily presented via taped voice and animated TV displays. Most test and drill programs become classroom games. Team members take turns answering computer questions and the computer announces the winner.

The computer can be programmed to momentarily flash a picture, word, pattern, or group of symbols on the TV screen to develop perception skills. Reflexes or time sense can be developed by requiring a specific keyboard response following programmed sounds or TV displays. Scoring adds a motivating game element to these types of drills. Morse code is taught by requiring the translation of tape voice passages into key depressions. The computer checks accuracy and gradually increases speed.

Reading readiness skills include simple shape recognition, word configuration recognition, and ability to maintain fixation on a moving object. The latter could involve having a child press direction-changing keys to prevent a moving TV spot from hitting obstacles. While not explicitly designed for this purpose, many of the games played with the system also develop reading readiness skills.

The computer can easily simulate logical aptitude testing devices[6], existing simple educational aids[7], or games[8]. The dot array TV display format is ideal for X-Y plotting practice. The computer can be used for individual testing

and scoring in any subject area and at any grade level. The test questions are provided in printed page or booklet form. The computer specifies which questions are to be answered via taped voice or TV display. Answers can be in the form of multiple choice, numbers, or words which the computer can check against a prestored table of correct answers.

The ability to skip audio frames on tape via the program controlled speaker relay provides added flexibility for test and drill applications. Two sequential voice frames could be provided per question. One frame would "tell" the student that his answer was right. The other frame would "tell" him that his answer was wrong, and why. For each student response the computer "plays" the appropriate frame and "skips" the other.

Games and Puzzles

Games and puzzles are normally associated with recreation. We have already seen that a number of utility programs have recreational aspects. The educational as well as recreational aspects of games and puzzles will be discussed here. Some of the possible uses of the computer in this area appear in Table 3.

*TIC TAC TOE
*Hexapawn[9]
*Sliding Block Puzzles
*State Change Games/Puzzles[10]
*Bowling
Football[11]
*Minikreig
*Target Shoot (Optional Gun)
*One Armed Bandit
*Network Games
*Twenty One
*Cell Matching Games
*Maze Tracing (Invisible, Changing)
*Race Games (Against Time)
*Space War
Bombs Away
Combinational/Sequential Puzzles[12]
Dodge Games (Space Ship & Asteroids)
Fish Card Game
Moon Landing
*NIM Games (Static/Dynamic)
Invisible Counter Board Games
Simulation Games[13]
Game Forms of Utility/Test/Drill Programs

***Already developed for the COSMAC miniprocessor.**

Table 3. Games and Puzzles

One of the most obvious aspects of this list is that there was no problem in motivating people to write game programs. TIC TAC TOE, Hexapawn, Twenty One, and Space War are played against the computer. Two player versions are, of course, possible. Hexapawn[9] was implemented as a learning program. The computer learns to play perfectly only after a number of games have been played. This type of learning program provides the basis for experiments where the user plots games played versus games lost by the computer to establish empirical learning curves. Heuristic versions of other games such as TIC TAC TOE are also possible.

A less familiar class of game involving interactive cell state changes is illustrated by a small plastic device called think-a-dot which has been sold for a number of years.[10] These types of games involve deducing the rules experimentally before any skill can be developed. The computer permits variations which would be otherwise impractical.

Bowling displays the pins on the TV screen. A ball spot randomly moves up and down at the opposite end of the alley. Pressing a key at the proper time rolls the ball. Sound effects, scorekeeping, and some random factors are incorporated in this two-player game. A variety of football games are possible. The simplest of these involves simulating commercial varieties of electric football[11]. Minikreig is a simplified war game. With the optional light gun a variety of computer controlled target shooting games can be devised.

Sliding block puzzles are easily simulated via the TV display. A move counter keeps the puzzle solver honest. Cell matching games involve momentarily flashing an array of cells on the TV screen at the beginning of the game. Each cell contains a symbol. Two players take turns trying to find cell pairs with matching symbols. The player with the most matches wins the game. Network games involve completing a path from one point to another before your opponent. A number of published combinational/sequential puzzles can be easily simulated[12].

Manipulating a moving spot through a racecourse or maze in the minimum time has proved to be a popular pastime. Spot acceleration and deceleration maximizes the challenge. The computer also permits invisible and changing mazes to be easily implemented. NIM type games can have a dynamic aspect included by using the computer. Board games can incorporate invisible counters whose positions must be deduced. Moon landing involves selecting fuel burning rates so as to avoid crashing. This type of game lends itself to experimentation, since it represents a simple simulation situation.

The whole area of using small computers in simulation games requires further investigation. Reference 13 contains brief descriptions of over 500 such games currently available for educational purposes. The availability of an inexpensive computer for referee, controller, and randomizing functions should be welcomed in this area.

Many of the utility, test, and drill uses previously discussed also provide the basis for games or puzzles. A widely available, mass market computer will undoubtedly stimulate the invention of many new games and puzzles for recreational and educational use.

Experimentation and Programming Uses

This area might be thought of as primarily educational. There is, however, a recreational aspect as well. The unique characteristics of a computer that make it ideal for experimentation also provide the basis of a fascinating hobby. Developing programs for your own computer embodies both educational and recreational aspects. Some specific experimentation and programming possibilities are listed in Table 4.

A classic example of experimentation via computer is provided by Conway's game of LIFE described in Reference 14. The hours of bootlegged computer time devoted to this program at computer centers all over the country are a testimonial to its recreational value. LIFE simulates

*LIFE[14]
Penny Matching Computer[15]
Turing Machine[16]
*Tutorial Computer
Picture Computer
Sound Computer
Machine Code Programming
Simulations
Variable Rule Games
Logic Simulator
Learning Machines
Probability & Monte Carlo Experiments
Heuristic Program Design

***Already developed for the COSMAC miniprocessor.**

Table 4. Experimentation and Programming Uses

a succession of generations for a colony of cells. Cell birth, survival, and death are controlled by algorithms in the program. Watching the patterns of cell change for each generation on the TV screen is addicting. The program is extremely rich in experimental possibilities. New starting patterns that yield interesting and sometimes surprising life histories are constantly being discovered. The availability of an inexpensive computer would permit even unsophisticated users to experiment via programs of this type.

Heuristic programs for simple games such as Hexapawn and Penny Matching[9,15] let the user develop experimental learning curves. This approach has been used to add interest to grade school math even without the availability of a computer.[17] Letting the user modify program behavior via keyboard parameters stimulates more creative and sophisticated experimentation; even TIC TAC TOE becomes a fascinating educational device when the user is allowed to modify the rules that the computer uses.

The computer can provide a variety of simple simulations that encourage experimentation. A simple moon landing game or racecourse game with acceleration and deceleration controls are examples. A logic simulator would permit experimenting with arrays of logic elements. Commercial hardware logic trainers are quite expensive. A random number generating program facilitates experimental development and understanding of probability curves. Game theory experiments are a natural application. None of these uses requires programming ability on the part of the user.

The area of programming provides the richest and most valuable recreational and educational experiences. Programming capability can be provided at several levels. A simple set of simulated instructions to move a spot around the TV screen could be provided via card or key symbols. Programming this picture-drawing computer could be introduced as early as second or third grade. The ability to program sequences of audible tones (or music) via a simple simulation language is also easily provided.

At a slightly higher level of sophistication, various tutorial computers can be simulated. A simple fixed word, decimal computer was simulated on the basic 1024 byte system. This included ten instructions, 100 words of user memory, and a simulated control/debug panel. Teenage children were able to write and debug their own programs with as little as one hour of instruction. Simple, simulated, tutorial computers open the door to a variety of interesting educational projects. What better way is there for a student to learn than by teaching his computer.

The construction of a hardware Turing machine model for educational purposes is described in Reference 16. The authors list several disadvantages inherent in the alternative approach of computer simulation:

A. Computer time too expensive.
B. Students have to learn how to operate the computer which has nothing to do with the simulation.
C. Graphic output display is too expensive and printed output is too slow and inconvenient.

It is interesting to note that the home/school system readily overcomes all three objections. How many other valuable educational tools might be easily provided via simulation if this type of inexpensive computer was made widely available?

For the sophisticated hobbyist the area of machine code programming will be of major interest. All that is required is to add circuitry for writing cassette tapes. This is an inexpensive option. The use of a small set of programming conventions together with specialized subroutines permits the sophisticated user to develop, debug, and save his own machine code programs.

Conclusions

A practical, inexpensive, free standing computer system based on the RCA COSMAC microprocessor has been described. The fact that over 30 programs are already running on prototypes demonstrates its viability. Over 80 uses are listed to dispel any notion that limited capability machines of this class are necessarily trivial. Several basic system enhancements were mentioned and many others are possible.

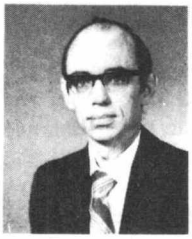
A system of this type would, for the first time, permit widespread access to computers. Much of the public awe and confusion relative to computers would be dispelled. The creation of a group of home computer hobbyists will stimulate invention and development of new computer devices and applications. Educational benefits are unlimited.

Computers are considered to be useful tools with which to achieve a specific end result such as processing a payroll or calculating a trajectory. This view of computers has often carried over into educational applications with the computer cast in the role of teacher/tutor. The low-cost home/school system described here is intended as a flexible plaything which encourages experimentation and stimulates a desire to learn. This approach may be more significant than the improvement of teaching methods for unmotivated students.

Acknowledgements

A. R. Marcantonio, C. T. Wu, and B. J. Call have made a number of contributions to the development of this low-cost computer system. A. D. Robbi and P. Russo have provided continuing moral support, ideas, and major assistance in developing and evaluating the system. Most of all, R. O. Winder deserves credit for making the development

of the system possible. ■



Joe Weisbecker has been developing new computers, communications terminals, I/O devices, and computer related commercial games for almost 20 years. He held engineering, advanced development and product planning positions within RCA's Computer Systems Division prior to joining the Lab in 1970. Mr. Weisbecker is a graduate of Drexel University, has written several articles, received two RCA Laboratories Achievement Awards, and holds 21 patents, with others pending. He is a Senior Member of the *IEEE* and a member of the Computer Society. The computer described in this article has been accepted as a member of the family by Jean (his wife), their two children (Joyce and Jean) and even their dog who punches random number cards with her teeth.

References

1. J. Weisbecker, "A Simplified Microprocessor Architecture," *Computer* (March 1974).
2. N. Swales and J. Weisbecker, "COSMAC – A Microprocessor for Minimum Cost Systems," 1974 INTERCON, Session 17/2.
3. "Putting Data on an Ordinary Audio Recorder," *The Electronic Engineer* (May 1972) p. DC-9.
4. E. Wolf, "Ratio Recording for Lower Cassette Recorder Cost," Computer Design (December 1972) p. 76.
5. M. Gardner, *Logic Machines and Diagrams*, McGraw-Hill, 1958.
6. A. Opler, "Testing Programming Aptitude," *Datamation* (October 1963) pp. 28-31.
7. J. Jones, "The Flap Board: A Simple Diagnostic and Remedial Tool," *Educational Technology* (January 1974) pp. 59-61.
8. H. Nurse, "Logidex," *Popular Electronics* (November 1973) pp. 63-66.
9. J. L. Hughes and K. J. Engvold, "Hexapawn: A Learning Demonstration," *Datamation* (March 1968) pp. 67-73.
10. B. L. Schwartz, "Mathematical Theory of Think-A-Dot," *Mathematics Magazine* (September-October 1967) pp. 187-193.
11. R. F. Graf and G. J. Whalen, "Electronic Football Lets You Play Like the Pros," *Popular Mechanics* (October 1967) pp. 147-149, 228.
12. J. W. Cuccia, "The Princeps Puzzle," *Popular Electronics* (May 1971) pp. 27-32.
13. D. W. Zuckerman and R. E. Horn, *The Guide to Simulations/Games for Education and Training*, Information Resources, Inc., 1973.
14. M. Gardner, "John Conway's New Solitaire Game – LIFE," *Scientific American* (October 1970), pp. 120-123.
15. D. W. Hagelbarger, "Seer, A Sequence Extrapolating Robot," IRE Transactions on Electronic Computers (March 1956), pp. 1-7.
16. I. Gilbert and J. Coh, "A Simple Hardware Model of a Turing Machine: Its Educational Use," Proceedings of the ACM Annual Conference (August 1972) pp. 324-329.
17. J. Ackerman, "Computers Teach Math," *The Arithmetic Teacher* (May 1968), pp. 467-468.
18. D. H. Ahl, Ed., *101 Basic Computer Games*, Digital Equipment Corp., 1973.

SIGNETICS INTRODUCES KIT FOR FAST BIPOLAR EMULATION OF THE 8080A MICROCOMPUTER

77 MAR 30, SUNNYVALE, Calif.–Bipolar emulation of the popular Intel 8080A microcomputer system, with speed improved by as much as 9 times, is now possible with a new 8080 Emulator Kit from Signetics.

The kit, designated the 3000KT8080SK, is a microprogrammed microprocessor using Schottky LSI components to implement the 8080A emulation at speeds that allow expansion of existing 8080A-based systems with no change in software.

Dr. John Nemec, Product Planning Manager in Signetics' Logic Division, said the emulator provides a 150ns microinstruction cycle time and a 150ns RAM access time.

The kit contains all parts necessary to construct the emulator, including the preprogrammed microcontrol store, printed circuit board, and an instruction manual with all schematic and microcode listings.

The 3000KT8080SK is fully equivalent in function to systems containing the Intel 8080A, 8228, 8224, and the 8212 and will execute all 8080A instructions, with performance improvements on individual instructions ranging from 2 to 9 times that of the 8080A system, Dr. Nemec said.

Features of the kit include: fully static operation (DC to 150ns), a single 5 volt power supply, hardware multiply and divide, and full vectored interrupt to any location within 64K of memory.

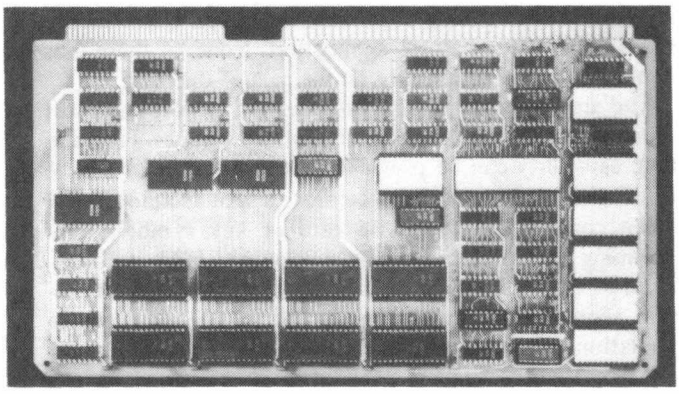
In addition, the emulator is microprogram expandable, enabling the user to utilize 12 unused instruction locations in the 8080 for expanding his instruction set. Over 150 spare locations are available in the control store for this expansion.

Since the kit uses bipolar LSI elements, it utilizes a single phase clock instead of the two-phase non-overlapping clock of the 8080A, according to Dr. Nemec. In addition, the system does not provide the signals which emanate from the 8080 during SYNC time but instead generates the status signals normally provided by the 8228 system controller.

The kit provides the standard address, data, status and control buses as defined in the Intel 8080 Microcomputer System Manual.

The 3000KT8080SK is now available form Signetics and its authorized distributors at a unit price of $299.

For further information contact Signetics, 811 East Arques Avenue, Sunnyvale, California, 94086; phone (408) 739-7700.



Signetics bipolar kit for emulating the 8080A microprocessor improves instruction execution speed nine times. The emulator is microprogram expandable, permitting 12 unused instruction locations in the 8080 set to be implemented as the user requires.

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 16 **June/July 1977** **Volume 2, Issue 6**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

EDITORIALS

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

Wayne Watch: A New *DDJ* Featurette for the Naive Novice 5

Which Manufacturer Offers the Longest Delivery Delay? *a psuedo-contest* 6

WARNING

Data Sync Leader Charged with Fraud 6

SUPPORT FOR SOFTWARE VENDORS

Response to Tom Pittman's Editorial, *Dave Allen* 7

More Support for Software Vendors, *Brent Longtin* 8

ALGORITHMS & SOFTWARE

A Story of Traffic Lights, or: A BASIC Program to Play the Game of LIFE, *R. Broucke* 12

Conversions Between Octal and Hexadecimal, *R. Broucke* 13

SCHDL: A BASIC-coded 'Daily Reminder,' *Bob Moody & Steve Williams* 15

Curve Plotting on a TV Screen with the Polymorphics Video Interface and MITS BASIC, *R. Broucke* 17

6800 Circle Maker, *David Hudson* 19

Dompier Plus: A "Musical Keyboard," *Bruce Hicks* 21

Permanent Solution to Missing SP and PSW Definitions in PTC Sys, *Bruce R. Kendall* 22

Tape LOAD and SAVE Routines for P.T.'s BASIC, *Ron Santore* 23

KIM-1 Breakpoint Routines: Plain & Fancy, *Willi Kushe* 25

A 'Turtle' for a VDM, *Marvin R. Winzenread* 28

HARDWARE

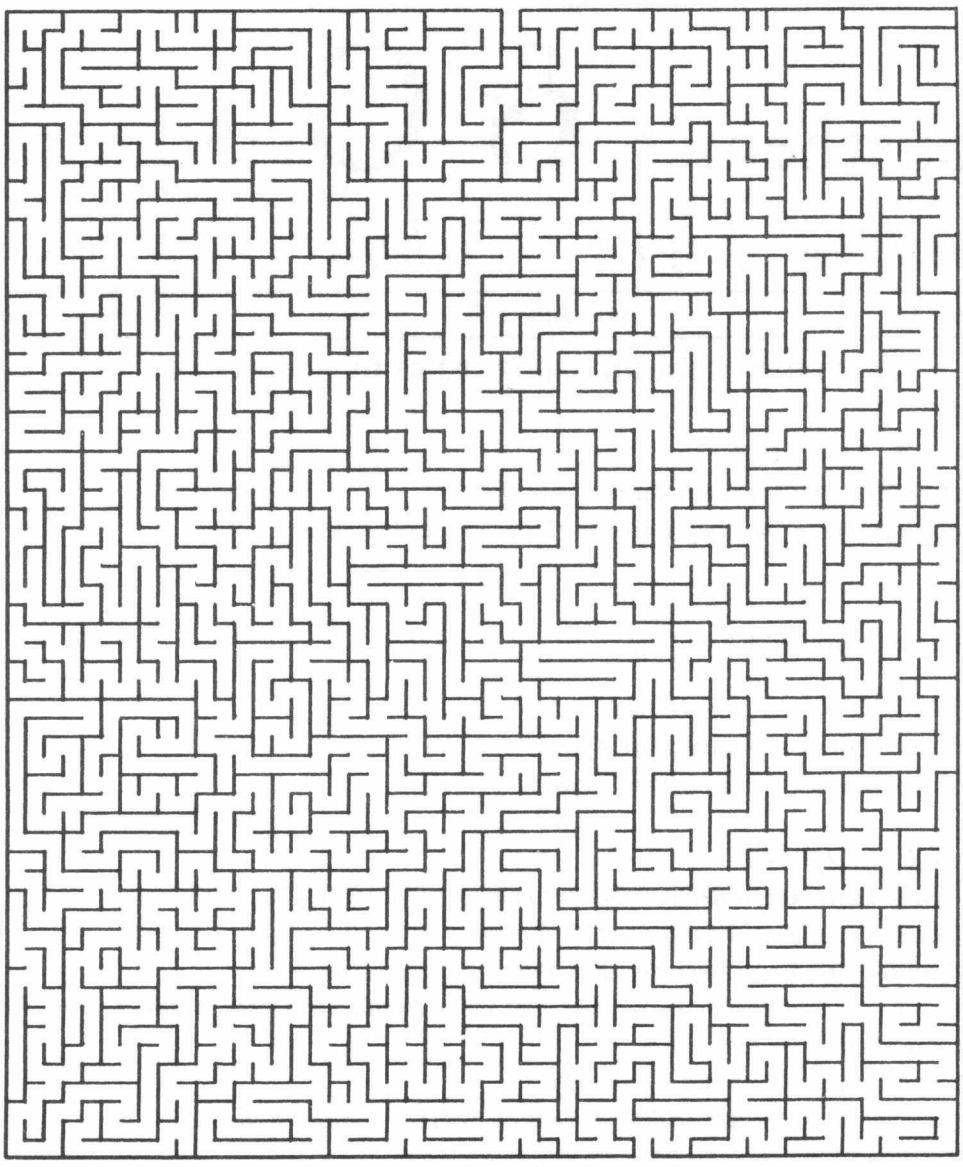
An Example of an M6800-Based GPIB Interface, *S. C. Baunach* 31

Complete Hardware & Software Details for an IEEE 488 Interface

Errata: TVT-II Mods had Pin Numbers Permuted, *Christopher Terry* 56

Lower Case Mod for SWTPC's CT-1024, *R. Lynn Smith* 57

. . . .and the usual collection of other tidbits. . . .



© David Dameron 1979

WAYNE WATCH: A NEW DDJ FEATURETTE FOR THE NAIVE NOVICE

Jim C. Warren, Jr., Editor, DDJ

For those who tend to assume the validity of anything that they read, simply because it is written – and that's almost all of us – I have decided that it is time to offer a regular and careful dissection and analysis of many of the comments written by Wayne Green. Wayne is the Publisher of *Kilobaud* and *73* and two promotional newsletters – the *Kilobaud Hobby Computer Newsletter* for potential advertisers, and *Computermania*[tm] for potential exhibitors.

Those "done unto" have access to DDJ

I am also offering the pages of *Dr. Dobb's Journal* to everyone who feels they have been maligned – explicitly, or as is more often the case, via exquisitely phrased innuendo – by Wayne. Most of those whom Wayne attacks have, essentially, no access to the potent weapon Wayne uses so freely – the media.

Of course, two of his most regular targets, *Byte* and the ARRL, involve publications with far larger circulation than those Wayne controls; however, they exercise great editorial restraint and diplomacy in responding to Wayne's ongoing snidery. This is probably partially due to their concern for the ethical standards that *should* be exercised by every publisher, and partially due to their recognition of the ad man's hack, "There ain't no such thing as *bad* publicity." (Nixon might not agree with the ad man.)

Why bother?

Recognizing the above, why am I bothering to do what *Byte* has consistently refused to do, and what the ARRL and its *QST* have done in only a very restrained manner? After all, I have repeatedly been told, "everyone" knows how Wayne does things. Why give him the publicity?

First of all, I *don't* believe that "everyone knows." They may know in the ham community, where they have watched Wayne for some years. It is not clear, however, that the community of computer hobbyists are aware of the care they must exercise in reading anything Wayne writes.

Secondly, I want to do what I can to make sure people know what kind of a person they are supporting every time they buy a copy of *KB* or attend Wayne's Boston exhibit.

Further, part of the ethical responsibility incumbent upon those of us who control media is to speak the truth, and particularly to counteract untruth. Most publishers are hesitant to do this when it involves making disparaging remarks concerning another publisher – perhaps for fear of reprisal; more likely because it looks bad to overtly attack a competitor.

Well, I don't consider *KB* a competitor to *DDJ,* and my suspicion is that Wayne feels the same way. We are radically different in format, style, and – to a large extent – content. As far as reprisal is concerned, I figure we can "bear up under the attack" and we'll be in such good company.

Preliminary Note

This article is purposely phrased in the first person – "I" – rather than the more usual "editorial 'we'," to emphasize that it concerns myself and *Dr. Dobb's Journal,* and does not necessarily concern the several other pubications of People's Computer Company. (If Wayne's gonna attack, I don't want him shot-gunning everybody in sight.)

Boundary conditions

Very explicitly, this is *not* meant to negatively reflect on John Craig, the Editor of *Kilobaud* – who has *no* control over what Wayne puts in *KB.* I have consistently found John to be an average nice guy, trying to do a good job, and have repeatedly noted him going out of his way to maintain diplomatic relations with people who objected to Wayne's style of doing things. This article is not meant to negatively reflect on any of the *KB* or *73* personnel.

In fact, it's not even meant to be a blanket indictment of Wayne; only of the style and content of much of his writings. It is my understanding that Wayne can be very helpful and friendly in a one-to-one relationship with those with whom he feels no competition (this is hearsay; not personal knowledge). I also understand from some hams that Wayne had a significant hand in encouraging the development of ham use of FM.

This is also not meant to imply that I think *KB* is a poor magazine – except for the fact that it carries Wayne's comments phrased the way Wayne phrases them. On the contrary, I find *KB* to be an excellent magazine equivalent to *Byte* and *Interface Age.* I encourage you to xerograph the portions of it that interest you. (Why, *of course* you can also do that with *DDJ.* We explicitly say so in the front of every issue.) But, if you *buy* a copy of *KB* or attend Computermania, be aware that you are giving support to Wayne to do things the way he does 'em.

And, that brings us to the meat of this article: examples of what Wayne does.

Example 1:
ComputerMania conflict with Personal Computing '77

In the June issue of *KB,* Wayne announced the "New England Computerfest" to be held in the Boston area, August 25-27 . . . address, "c/o *Kilobaud* Magazine." He stated, " It looks as if there may be a conflict between the recently announced Boston Computerfest . . . and the PC-77" in Atlantic City. Further along he sez, "Frankly, I think that there should not be two computerfests set up on the same weekend, and while I'm enthusiastic about NEC-77 . . . I think the organizers should have made a better effort to avoid the conflict."

In the July issue of *KB,* Wayne was pushing "ComputerMania" very heavily. Different convention name, but same mailing address, same dates, same location (Boston area) . . . and no further mention of NEC-77, about which he was so enthusiastic in the previous issue. It's reasonable to assume NEC-77 became ComputerMania.

In the June announcement, note how his phraseology *implied* that he, Wayne Green, was merely an outside observer of "the organizers" whom he was chastizing for the conflicting schedule. By July, he was accepting responsibility for the Boston dates.

In his May 16th ComputerMania newsletter, he said, "Most of the Fall dates at the major Boston convention facilities are taken a year or so in advance, so the choice of show dates for ComputerMania were determined by availability more than preference." In his May 23rd *Kilobaud* newsletter, he sed, "It turns out that if one wants any of the larger facilities in September or October (in Boston), one has to sign up a year or two in advance, so the late August date was a most fortunate choice."

This morning, I phoned the management of the Commonwealth Pier where Wayne is holding his show. Speaking to Mr. Ed Phillips of that office, I found out that (1) "August is not a good month here. Everybody's on vacation"; and (2) even at this late date – when the Pier would be far more likely to be scheduled for use than months ago when "the organizers" first reserved it for August 25-27 – the Pier was still "available every weekend in August, except the 25th to 27th." Furthermore, it was also available Labor Day weekend in September, as well as September 17-18, and 23-25. *You* judge whether or not ComputerMania was explicitly and consciously scheduled to be in direct and preemptive conflict with Personal Computer '77 in Atlantic City on August 26-27.

There is an entertaining supplement to this: by the Pier management's own statement, August is a dead month for such shows in Boston (Mr. Phillips expanded on the above quote in our phone conversation, pointing out that even one of his regular and very successful art shows refused to schedule in August, saying it wasn't worth it). On the other hand, even a provincial west coaster knows that many, many more people could be drawn to a personal computing show in Boston in September or October than can be drawn in August . . . if for no other reason than the fact that all the college and university students will be back from vacation and in the area. I would say that "the organizers" of the Boston show were either totally unaware of their potential market, or else they very badly wanted to do what they could to impair the success of Personal Computing '77 in Atlantic City. In either case, they will certainly have a much smaller attendance than they could have expected after people – particularly students – returned to Boston from their vacations. And, of course, that means much less income.

Example 2:
Maligning Personal Computing '77

In the June *KB*, Wayne writes, ". . . choose between grubby Atlantic City, which was like a sauna last year, or Boston." In his May 23rd *KB NL,* he mentions, "If hot, humid, tacky Atlantic City (is) your bag . . . " The *implication* is that Boston *isn't* hot and humid like a sauna in August. In both cases, the convention areas will be air conditioned.

In his latest ComputerMania newsletter, he may have really put his foot in it. He states, concerning Atlantic City: "If you are going to brave the crime rate . . . " and ". . . most of the current moving action reported to be the Mafia and their control of parking lots, linen supply, etc." It would seem to me that Atlantic City must either nail Wayne for slanderous implications (notice he didn't come right out and *say* the Mafia was active in Atlantic City; he only implied it), or sit back and allow several thousand potential computer exhibitors to assume Wayne's implications were true, which might impact their future computer convention business.

Example 3:
The Numbers Game

This is where Wayne absolutely shines. For example:

In the June 13 ComputerMania Newsletter promoting exhibitor participation in ComputerMania, he says, "Dealers will be looking for from $25,000 to $100,000 in sales from their booths" No doubt they will! But I doubt that they will be expecting it. Furthermore, in his *Kilobaud Newsletter,* last Dec. 9th, he states that the average exhibitor

DATA SYNC LEADER CHARGED WITH FRAUD

On July 7th, we received a call from John Craig, editor of *Kilobaud,* asking us to spread the following information as widely as possible:

John – who lives not too far from Santa Maria, California – said he had just found that a Norman Henry Hunt, Jr., alias Col. David Winthrop, USAF, Ret., alias William D. Winthrop, the Chairman of the Board of Data Sync in Santa Maria, had been arrested by the Santa Maria Police Department in cooperation with the FBI on a fraud charge. John was asking that I spread the word to help hobbyists avoid possibly being ripped off by Data Sync.

We contacted Detective Ernie Kapthahn of the SMPD (805-922-7811) to verify the story and obtain the rest of the details. Det. Kapthahn stated that Hunt (etc.) had initially pled guilty to three counts of False Pretense Theft, a felony, but had just withdrawn his plea. He was currently being held by the SMPD in lieu of $100,000 bail.

Furthermore, Det. Kapthahn stated that Hunt had a long history of such adventures. He said Hunt operated out of Weatherford, TX, in 1976 with a company name of Computer Devices, Inc., and an alias of William Northrop. Prior to that, he worked from College Park, GA, under the name of Digi-Tone Communications Corp. and the alias of Robert L. Renfro. Before that, it was the National Digital Computer Co. in Scotsdale, AZ, under the name of William Andrew Scoville, as well as Scoville Automation out of Las Vegas and Reno.

Consider the word to have been spread. Contact Det. Kapthahn (pronounced "Ka-pan") for more details.

---

sales reported to him from the Personal Computing '76 show in Atlantic City "was a bit over $3,000."

In another *ComputerMania Newsletter,* he says, "We're hoping for a minimum of 25,000 . . . and two to four times that could not surprise us." Again, I say, no doubt, but hardly probable.

He says, "There are only 250 booths available in the air-conditioned area," not bothering to mention that he has only 262 in all. Incidentally, *Atlantic City* has 146 booths and reported last night that 136 have already been taken, giving me the names of 87 exhibitors. In comparison, the most recent *ComputerMania Newsletter* gave a booth chart that *showed* 144 sold. This morning, after making two phone calls to *Kilobaud,* I could get the names of only 65 exhibitors. Gary Dozier, the very pleasant person who is handling exhibitor arrangements for *ComputerMania,* did say that he had "about 20 more" that he hadn't processed . . . but earlier in the morning when I first spoke to him (before he talked to Wayne), he said he thought there might be three or four in the mail that he hadn't yet entered on his master list. I explicitly asked for the booth assignments or at least booth-counts of the named exhibitors, and Gary said that he felt it "wouldn't be fair" to give me that information. He could offer no justification for that position and was obviously embarassed by the position he was in.

About the ComputerMania exhibitors

Once Gary gave me the exhibitor names – with my telling him that I was going to spot check them (to which he said that they had signed contracts or money from each named exhibitor), I phoned three of them that sounded unlikely to me. Anderson-Jacobson was the first one. Pam Lebut, the person who handles convention participation for Anderson-Jacobson, said they were *not* going to be exhibiting at *ComputerMania*. The other two firms requested that I not publish their names, but said that they were *not* going to be exhibitors at CM, and had not sent money or a contract. (I'll be happy to furnish anyone who calls with, say, eight names from Gary's list including these two, and they can check the validity of *my* statements). One of them, in asking me not publish their name in this context, commented, "I hate to be a chicken, but that's a powerful dude," referring to Wayne. All three of these companies were explicitly specified to me by Wayne's ComputerMania exhibitor manager as being officially committed ComputerMania exhibitors.

There's another interesting name on Dozier's list: Data Sync. Please see the article above.

Enough of this for today. As implied by my opening lines, I plan for this to be a continuing column . . . for at least as long as Wayne continues to publish the kind of material he writes. Whenever he writes only material that is fair and evenhanded, then this column will disappear. I hope that happens soon.

Until then, in future issues I expect to discuss such tidbits as Wayne's claimed circulation figures, his *KB* promotional literature, various aspects of his interaction with *Byte*, and particularly some of his published statements concerning companies that "may not advertise" in *Kilobaud*. I suspect that I may also be discussing some of his comments about me and/or *DDJ* (it'll take about three months for them to appear in *KB*, but Wayne has a much faster turnaround in publishing his two newsletters).

I encourage you to check all information I have given here. And, I encourage Wayne to exercise editorial restraint and fairplay. If and when he does, I will be happy to support him in his activities.

a pseudo-contest

WHICH MANUFACTURER OFFERS THE LONGEST DELIVERY DELAY?

We have received such a volume of complaints regarding tardy product deliveries – and received them on such a continuous and growing basis – that we have decided to initiate a non-contest:

If you ordered a product and failed to receive it within a reasonable amount of time (at least a month), please send us the details *for publication*. We are particulatly interested in such orders where you have "fronted" money – partial or full payment. Please send the following:

You must include your name, mailing address, and phone number. If you prefer that we not publish it, we will honor such a request, but we must have the information for our own files. State the date on which you ordered the product, and, if you fronted any money, include a copy of the cancelled check (both sides). Give the name of the company with which you placed the order, a list of what you ordered, and – if you have it – a copy of the original order document (letter, order form, whatever). State the total cost of the order. If possible, indicate where and when you first saw an ad for or heard about the product you ordered, and the source of information you used in placing the order (e.g., an ad, a brochure, phone call, etc.).

The following statement must also be included, with your signature: "At the time I placed this order, I had no indication that there would be more than a one-month delay in my receiving it. To date, I have not received a notification of shipping delay." (If you advanced some money with the order:) "To date, I have not received an offer of an immediate refund of the advance payment I made with the order."

We will then publish at least the essentials of the information you furnish in the next issue of *DDJ*, in particular: the company name, a description of what was ordered, the value of the order, the date ordered, and the date of your complaint to us. Unless you request that we not do so, we will also publish your name and phone number.

It won't necessarily get you "justice," but it may provide a warning to others who are thinking of doing business with the same company for the same product.

Of course, we will continue to publish letters *praising* unusually responsive vendors, as we have done in the past. The reason for this special activity regarding delivery problems, is that it appears that the situation is getting completely out of hand, and a number of naive consumers are becoming very disillusioned with the personal computing market.

Electronics professionals have come to expect tardy delivery of electronics components by their suppliers, and have come to assume that the fact that a product is being advertised and orders solicited does not mean that the product is available or can be delivered. The professionals *know* they must be very cautious in ordering electronic goodies. Consumers and lay people don't. We feel that manufacturers marketing to the general public – to a consumer market – should follow the traditions (and the trade regulations) of the consumer marketplace.

"PET" ARTICLE PROPOSAL

Greg Yob has proposed that we publish an article about Commodore's soon-to-be PET system: "The Care and Feeding of Your Pet & How to Train It."

RESPONSE TO TOM PITTMAN'S EDITORIAL

Dear Jim: March 29, 1977

Bouquets to Tom Pittman, both for his Tiny Basic and for his recent letter which outlines much of the current software problem. I am in agreement with him in almost everything he says, *especially his advice to manufacturers on how to solve some of these problems through standardization.* I regret that even my one experience with a manufacturer resulted in the manufacturer offering "un-standardized" software to support my hardware design. C'est la guerre.

I feel, however, that Tom has omitted reference to a fundamental fact and reason for the existance of most of the "hobbyist" market. These "toys", hardware and software alike, are being used as tools for self-education. This market should not be compared to model railroading and other hobbies of a similar ilk.

Neither the software *nor the hardware* is "a piece of cake" (contrary to Tom's understanding). Tom is apparently a software man who has done a little work with hardware. I am just the opposite. I see remarkable similarities in hardware and software problems. Given a little time, patience, and tinkering the computer "hobbyist" can indeed concoct some simple hardware, almost like Tom's "piece of cake". Similarly, with a little forethought, the same hobbyist can write game programs for himself, perhaps even a Tiny Basic (though he would be foolish not to use Tom's Tiny Basic). Two years ago, after I built my first floppy-disc interface, I wrote a simple D.O.S. (in machine-code no less). I found the hardware and software tasks roughly comparable. Neither was a piece of cake, and both were within the grasp of talented "hobbyists".

Regrettably, almost none of the "hobbyists" want to do it themselves. The handful who do have the ability to build hardware do so, but don't stop at building just for themselves. They have flooded the market with poor-quality, poorly-supported, un-standardized hobbyist equipment. The same personalities are present at the software end of the spectrum, and similar quality software is available to match the disreputable hardware.

Where I take issue with Tom is my conception of the solution to this problem: Vendors of hardware and software should assist "hobbyists" in their attempts to do-it-themselves! We don't need any more turnkey systems. We don't even need the number we now have. Many of them don't even work very well, and only add fuel to the "non-standard" fire.

If these vendors, of hardware and software alike, would quit behaving like gurus and start providing assistance to people who try to upgrade their own abilities, the whole community would benefit. The genuinely competent vendors (as I am sure Tom is) would prove their worth by being out in front with new ideas and by having the experience and ability to implement their ideas more efficiently and more reliably than the "hobbyists" can.

My pet peeve is: vendors who supply hardware without schematics and software without source listing. If a vendor really has a technique which deserves to be considered proprietary, then he is wasting his time in the hobbyist market.**[Although I tend to agree with you, I do not get upset when the manufacturer of my automobile fails to provide me with the drawings for the engine. In fact, I didn't even get upset at not having access to the source listing of OS360 when I had occasion to use such a dinosaur. The difference is hacker vs. user. –JW]**

As I see it, Tom is guilty on this score. Anybody who subscribed to *DDJ* from its start is aware of how Tom's Tiny Basic works. As I understand it, nothing revolutionary is going on. By withholding source listings Tom is shackling the hobbyist's curiosity and retarding the growth of competence within the "hobbyist" community. Tom is also complicating his own life, since only he can modify his software to support the diversity of hardware configurations. Why doesn't he shift that responsibility over to the user? No wonder there is such a software vacuum. At this rate, the software market will be in its infancy forever. By offering Tiny Basic for $5 without a source listing, Tom has essentially put a "Pong" machine in an arcade. If he really expects to make $25,000, then why is he wasting his time collecting quarters? **[Why not? Anyway, 5000 x $5 = $25,000]**

Sincerely,

Dave Allen
Electronics Consultant
1317 Central Avenue
Kansas City, KS 66102
(913) 371-6136

SEPTEMBER ISSUE OF *INTERFACE AGE* TO CARRY AN 8080 GENERAL LEDGER "FLOPPY ROM" RECORD

The May issue of *Interface Age* magazine made computer history by including a "Floppy ROM" – a flexible phonograph record containing a complete 6800 4K BASIC – along with complete instructions on how to interface a record player to your computer and read the program. (Shortly thereafter, several people at a Homebrew Computer Club meeting mentioned that they had successfully built the interface and read the record – including one person who did so with an 8080-based system. He reported that he could read the program, even though he couldn't execute it. He figured that some 8080 software would be forthcoming and wanted to be ready for it. (He was right.)

Now, the September issue of *IA* will contain "Floppy ROM #2", a complete general ledger accounting program for 8080-based systems. The adjacent human-readable pages will contain complete user documentation for the software.

In April, when Bob Jones "published" his "Floppy ROM #1", we were swamped with preparations for the First West Coast Computer Faire, and were remiss in not making editorial note of it at the time. We are delighted to make note of it now – belated though it may be – and wish to congratulate Bob on an innovative job well done. Producing and inserting those records is a costly venture, and "publishing" a computer program via such an untried, risky, and expensive medium is an experiment well worthy of our praise and support.

Interface Age is located at 13913 Artesia Blvd., Cerritos, CA 90701, (213) 926-6620. U.S. subscriptions are $10/year. $1.75 for a single issue.

CORRECT CREDIT FOR PEEK, POKE & CALL

Dear Jim Warren, Received: 77-Jul-21

Jef Raskin's review of MITS, Poly, & NIBL BASIC needs some clarification. The PEEK & POKE functions were not invented by Micro-Soft. DEC (Digital Equip. Corp.) has PEEK & POKE in its BASIC PLUS as well as CALL when implemented with their RT-11 system & has had it since before MITS ever dreamed up their Altair 8800. PEEK & POKE were first put in a hobbyist BASIC by Micro-Soft, but the young people reading your excellent mag should be aware that the world of software did not begin with the advent of MITS Altair. Jef should be aware of *all* the facts before he uses such words as "invented" or "useful innovations".

Ray Atnip
Owner, The Bit Barn
7331 Harwin, Suite 117
Houston, TX 77036

P.S. You will probably get a lot of responses in relation to Jef's article.

****

MORE SUPPORT FOR SOFTWARE VENDORS

Brent Longtin
Algorithmics Inc.
Box 56
Newton Upper Falls, MA 02164 77 June 6

Dear Jim,

Bravo for Tom Pittman's letter on support your local software vendor that appeared in the February issue. The whole issue of software development and support costs is one that deserves a good public airing. One reason why I suspect that it has not received wider attention to date is the dirth of independent software companies that are actively supplying software for the hobbyiest marketplace. Lest there be any confusion, I do not consider one individual working for Amalgamated Behemouth Industries who cranks out software in his weekends and evenings frequently in violation of his employee contract to be an example of an independent software firm. The purpose of this letter therefore is threefold; to provide support for Tom Pittman's position by describing our experience with supplying software for hobbyists, to suggest that a major element of the software problem - support costs - is yet to be recognized and finally to suggest some very positive steps that can be taken to help solve the problem.

OUR EXPERIENCE

Algorithmics' primary business is the production of commercial applications software. When we started to plan our products over a year ago we immediately recognized that to produce any amount of significant software we needed more powerful development tools than were then available on the market. Also, because our products would be based on the new Z-80 we needed a Z-80 assembler which judging from past performance we did not feel would be forthcomming from the hardware vendors. Consequently, we sank over $40,000 and five months of effort into building ZAPS - our Z-80 cassette operating system. We were very proud of the results. We had succeded into cramming into an incredibly small 14K bytes of memory including variable storage, buffers and 1K bytes of symbol table space:

.a full Z-80 assembler

.a line-oriented context editor very similar in function to the one you published last year

.labeled cassette tape subsystem

.a variety of useful system utilities (debug aids and number conversion routines for example)

The power of any tool can be measured by what you can build with it. By this measure ZAPS is a winner. First, we had ZAPS available for our use immediately when the Z-80 was introduced thereby getting a jump on all our competitors who had to wait for the manufacturers to introduce their software. Secondly, we used it to produce in record time our first major commercial product - word processing - that consits of over 24K bytes of application code. Dr. Dobbs can judge ZAPS merits for itself as we have sent you a copy for evaluation.

In an attempt to recoup some of our time and money invested in ZAPS we decided to sell it into the computer hobby market. Before doing this however, we had to turn ZAPS into a true product by writing a user's manual. Our initial cost to do the manual was slightly under $1,000. This investment has subsequently risen to roughly $3,000 because of a major rewrite. At first we approached hardware manufacturers who were not offering any assembler level development systems and offered to sell them ZAPS. A few manufacturers who seem to recognize the development costs and end-user value of software jumped on the ZAPS bandwagon and became O.E.M.'s for the product. Some of the negative responses we received are quite interesting:

(a) One manufacturer refused to carry the Z-80 version because it confliced with his own product which was under development. The product he has subsequently announced requires almost as much memory space for its assembler alone as all of ZAPS.

(b) Another, tried to beat us down to a 20% of $10.00 royalty deal for the 8080 version. Never mind recouping my development costs, a few phone calls from customers per month asking ZAPS questions and my 20% royalty is gone.

(c) Several muttered that they had vague plans about building their own software in the future. That was months ago and I still have not seen any results. One can only wonder what the hobbyists who bought their machines are using.

****

In a few cases I suspect that ZAPS features conflicted with the hardware manufacturer's primary interests in moving more raw iron. Three features of ZAPS that may have hurt us in this area were:

(a) ZAPS is super memory efficient. Not only does it require only 14K bytes, but no line numbers are used in files. Manufacturers who plan on selling more memory boards through memory-inefficent systems software have not learned that one of the primary rules of software is that it quickly expands to consume all available machine resources. The customer is their best memory eater.

(b) ZAPS runs on both 8080 and Z-80. In several cases we saw hardware vendors who were moving their 8080 customers up to Z-80's so that they could get decent software support - assemblers and the like - not available on their 8080.

(c) ZAPS did extensive cyclic redundancy checking of memory and cassette tape interfaces thereby making any hardware problems blatantly visible. Manufacturer supplied diagnostics for their systems have for the most part been totally lacking or grossely inadequate. In many cases customers are running systems with memory and mass-storage flaws that every once in a while show up as a mysterious action by a Basic program. I suspect that if every hardware vendor were to provide first-rate diagnostics, the majority of them would go bankrupt when large numbers of their systems came rolling back from the field for repair.

I feel however that very few hardware manufacturers provide poor software support for the reasons given above. The major reason is inexperience on the part of the founders of these companies regarding what constitutes good software. Most hardware companies not surprisingly are founded by individuals with a primarily hardware background who heave and grunt for six months to a year to get their first piece of iron out the door. Only then do they think, 'Well, I guess we had better toss in some software'. Can you really expect an afterthought to result in a first rate product? Furthermore, in the case of Tom Pittman and ourselves, can you expect them to recognize quality software when it walks in the door?

As an aside to any hardware vendors who may be reading this letter, you might be interested to learn that one independent builder of hobby software, Microsoft, the people who brought you MITS Basic, is now developing Basic for the new Commodore machine. This chip builder seems to have the money to fund good software, and the sense to consider independent suppliers.

Our latest approach in selling ZAPS has been to sell through computer stores and in areas where we do not have a store, direct. To the extent that ZAPS now pays for its own support costs, this marketing technique has been successful. A distributor setup for software has two unique problems that are probably not apparent unless you have tried it:

(a) In the area of systems software where you are competing with the hardware manufacturers offering, you must overcome the automatic aura that surrounds it. Many people seem to feel that because the software was produced by the hardware vendor it is automatically superior and guarantees him/her of easy upward migration to new hardware devices. I have yet to see any evidence that either of these two points are true. Perhaps Dr. Dobbs' readers could provide some.

(b) Establishing dealerships can be very expensive for the independent software vendor. It requires either an expensive advertising campaign or one or more sales calls on each store that you wish to carry your product. I have considerable sympathy for the stores when it comes to selling software and can therefore understand their reluctance to stock it as readily as hardware. Software does not sell itself. It must be demonstrated and in most cases this implies that stores must use their already scarce time to train their staff. Hardware, on the other hand, has many built-in self-selling features. It can be handled. The quality of the circuit board can be inspected. And the customer can count the number of IC's providing him with some measure of its worth.

THE SOFTWARE SUPPORT PROBLEM

The cost of software development that Tom documented represents only the tip of the iceberg. Based on past experience from the minicomputer and large-machine markets, development costs represents only 20 to 40 percent of the total life-cycle cost. Separate versions of software for different hardware machines (Tom has 6 Basics we have 3 ZAPS) is one element of support. Other major contributors to support costs include the following.

(a) Hardware evolution. New mainframes, larger video displays, novel keyboards, and a variety of mass storage devices must be supported. In the case of hobby applications where your customer is generally technically sophisticated it is generally sufficent to provide listings of the software I/O drivers for a few common devices and let him/her modify to suit the environment. But what about applications software for the non-computer oriented person? As an example of the problems here, we are writing a users manual for a word processing system that must include photographs of CPU front panels to show the user how start and load the machine, descriptions and photographs of keyboards to explain editing functions, and photographs of video displays keyed to descriptions in the text. Simply because it is absurd to attempt to do a new manual for every possible hardware configuration we are forced to specify that our word-processing software will only run on a few selected brands of machinery.

(b) Manuals. It is probably as difficult to do a good manual that is properly oriented to your audience as it is to write the software itself. Two approaches to technical manuals that now pervade the hobby field must stop. The first is the idea that a technical manual consists of a listing of the program with whatever comments the program author saw fit to insert. This sort of documentation might be fine if you are a software guru but it is an insult to the novice programmer who wants to use it. The second approach is to provide next to no documentation. Incredible as it may sound I have actually seen a Basic offered by one manufacturer with an accompanying user manual exactly two pages long that consisted for the most part of a listing of the language verbs. Of course, if you do not provide documentation or if the documentation is in the form of source listings no customer can ever complain that the software fails to meet its specifications.

(c) Dealer training & sales support. To sell software you must provide demonstration programs, literature hand-outs, and some form of training for your dealers. One of the reasons why Basic has been so successful in the hobby market is the availability of a an excellent selection of games that serve as program demonstration aids. Need I mention that many of these games originated from the book '101 Computer Games' that was developed by the minicomputer industry? Hardware vendors sell their software for the most part by tossing it in with the hardware (sometimes quite literally - you open your CPU kit and there is an operating system tucked away in the corner).

(d) Bugs. Even in the case of Basic which represents a well understood technology and simple assembler-level operating systems, we are already seeing versions 1, 2 ,3 and so on to fix reported bugs. With complex disk operating systems and powerful high-level languages the bug problem is going to become even more severe. Eventually, maintenance and enhancement of existing software is going to set the pace with which new hobby products can be produced because a good percentage of software talent will be devoted to these activities.

(e) Enhancements. I do not know of any way to produce the exact right functionality in any product on the first try. User feedback and competitive pressures demand that you continue to enhance it.

(f) User Support. If you are really committed to serving your customers, someone must answer their written and telephone questions.

WHAT CAN BE DONE

From our own experience, I think that Tom's list of nine standardization steps that hardware manufacturers should follow would go a long way towards making our life easier by reducing software support costs. Three other parties heavily comitted to the supply and distribution of hobbyiest software - Dr. Dobbs, the independent software manufacturer, and the computer stores - can also help.

1. Dr. Dobbs

(a) Product Evaluation. You have already taken the first major step toward helping the independent software vendor through your product rating service. If, as both Tom Pittman and I have suggested, the products of the independent manufacturer are really superior, a 'no punches pulled' evaluation on your part could provide invaluable support for the independent. Eventually, as the list of software products grows. standardized checklists and tests could evolve with the results published in booklet form in much the same style as Consumer Reports buying guides. The only problem with the evaluation process is the huge amounts of time and equipment that it will require. How for example is the first ANSI standard Cobol for an 8080 (or whatever machine) to be tested to insure it conforms to the standard and is bug-free.

(b) Standards Publication. High level language compilers and interpreters should be an implementation of either a complete or well-defined subset of a standard definition for the language. Dr. Dobbs should either publish such language definitions as you are doing in the case of CASUAL or give reference sources where they can be found. Come now all you software builders, how many of you read the ANSI standard definition of Basic before doing your version. How many of you know where you can obtain one. Dr. Dobbs could make the category of 'adherence to language definition' an important part of its software product review. With a little enthusiasm for standards we might even be able to define a standard set of mnemonics for all microprocessor assemblers - it certainly should be possible for the 8080 and Z-80 .

(c) Copyright Education. One of the most aggravating problems the software manufacturer comes up against is people who make bootleg copies of his software and then proceed to distribute them. I don't believe that this is so much the result of deliberately larcenous intent on the part of such individuals as it is ignorance as to what the true costs of software production are. For the short term (next few years) a strong editorial stand on the part of Dr. Dobbs denouncing this practice would be most helpful. Such an editorial should be reprinted periodically to educate newcommers to the computer hobby field. Ultimately I think the problem of software theft will disappear when there are several million hobby computers and because of this large market for his product the software manufacturer can sell even the most sophisticated product for little more than the cost of reproduction. We are still a long way from this goal.

2. The Independent Software Manufacturer

(a) Design For Hardware Variation. At the very least it should be possible to customize your software for different keyboards, displays (hard copy or video), and mass storage devices. The problem conflicts between ROM location and application code area is best solved by total code relocatability. However, the trend in hardware design does seem to favor Tom's suggestion of locating ROM's in high address space, so software that runs from 0 up is a good compromise. User manuals should include customization tips to help customers adapt the code to their hardware.

(b) Fund Hobby Products From Commercial Software. With the present size of the market, it is completely unreasonable to expect hobbyists to fund major software products via their purchases. Industrial customers will pay for the true costs of software. Micro-processor software companies should spin-off useful pieces of commercial software into the hobby marketplace at low cost thereby creating a snowball effect - more software leads to more hobby machines that leads to a demand for more software. Eventually a critical market size threshold will be passed for some machine whereby production of software for it can be justified in terms of hobby sales alone.

3. The Computer Stores

(a) Stock Software. Computer stores should carry as wide a variety of software as they do hardware. Considering the fact that it is the software that turns the hardware from a lump of silicon and copper into a useful device that can provide hours of enjoyment for the customer, computer stores should give software top billing. For most customers, the software determines the characteristics of the system far more than the hardware. Stores should insure that software is prominently displayed and their staff should learn how to demonstrate and sell it.

(b) Fund Software Development. Stores are in an excellent position to understand what software is needed by the hobbyiest and commercial customer. Local software houses can be an excellent way to meet this need. Funding of customized software can prove a good long term survival strategy for stores when they must compete with machines produced by chip builders such as Commodore and sold through mass retailing chains. To a prospective customer there is very little difference between a $500 or less machine that runs Basic and a machine at twice the cost running the same software. But when the stores can show the customer a dozen different software application packages, a score of high-level languages, and customization at a moderate cost, they have a definite sales edge.

SMOOTH SCROLLING AHEAD?

All of the video displays I've seen do scrolling the same way as a Teletype, everything jumps up one display line at a time. Actually this should be called "ratcheting" instead of scrolling, as the motion is abrupt rather than linear. Ever try to read a listing on a rapidly scrolling monitor? If so, I'm sure you've noticed how jerky the display is and how hard it is to read anything as the text jumps from line to line.

In the early days of personal computing (i.e., last year) hobbyists were happy to be able to display *anything*, jerky or otherwise. Today, with lots of video displays available, byters can afford to be choosy. One of the features I'm looking for in my next video display is linear scrolling, where the whole page moves up one *raster line* at a time. Has anybody seen one?

Jim Day 77-Jul-15

HOW 'BOUT IT?

[Post Card] Received: 77 Jun 15

How about a series on software security, i.e. how to protect your system from unscrupulous programmers, etc. This would, of course, be more relevant to the business use of computers, but an in-depth introduction, with references would familiarize *us* with what to look for in hardware/software should we shift into the business environment.

J. Sikorski
Box 574 NUMS
303 E. Chicago
Chicago, IL 60611

****

(c)Learn How To Sell Software. Software is one of the hardest products to sell properly. The first sales hurdle is to explain to customers why an audio cassette that might contain a complete 8080 Fortran compiler for example might cost several hundred dollars rather than $2.98. If the product is applications software, the store owner must learn how to convince a customer that the 90% match between what the package does and the customers need is close enough to make it worth purchasing. Systems software --- compilers, assemblers, and the like --- is so esoteric that I am afraid that it can be sold to only the most sophisticated customer. The software vendor can help this situation by supplying sales guides for his products.

You have now heard from three sources on the topic of hobby software - Tom Pittman, Bill Gates (creator of MITS Basic), and Algorithmics. The message is loud and consistent. Hobby software is a financial bust! Be you store owner, hardware builder or hobbyist it is in your own best interest to help the independent software vendor. As for Algorithmics, we definitely intend to have future products for the hobbyist just as soon as we can find someone to pick up the tab.

Yours truly,
Brent Longtin
Algorithmics Inc.

****

SUGGESTIONS FOR MUELLER'S "APL"

Dear Sirs: 77-Jul-25

I read with interest the article by Erik T. Mueller in your April issue on a version of APL for 8080s. APL is a fine language which I believe will eventually overcome some of the BASIC myopia which thus far pervades the micro-world. To have implemented it at all in 5.2K is heroic.

However, I am critical of some of Mr. Mueller's design decisions regarding the implementation of the APL functions using standard ASCII characters. As most people who have had any exposure to APL know, the language uses a special character set to represent primitive operators. The problem of finding a suitable representation of these primitives in the standard character set available on most terminals is not easy to solve. Mr. Mueller's solution seems to be to try to select the ASCII character which is most like the normal APL character where possible. This works fine for things like $\underline{+}$, $\underline{-}$, and a few others, but there just aren't enough characters to go around. So he ends up using things like $\underline{!}$ for absolute value (where APL uses $\underline{|}$) and $\underline{\backslash}$ for the iota function (where APL uses $\underline{\iota}$). Now the problem with doing this is that APL (full APL) uses $\underline{!}$ and $\underline{\backslash}$ for other primitives (e.g. $\underline{!}$ for the factorial function) which introduces a serious imcompatability between his APL and the more standard versions of the language.

This approach is poor human factors and needlessly confusing. Granted that potential APL users in the micro-world should not have to wait for special hardware to draw all those funny Greek characters in order to reap the benefits of APL. But there is a better approach that is available now without introducing confusion; Mr. Mueller uses it for one of his functions (the *RHO* function). That approach is to use the *names* of the APL primitives to call them where they cannot be faithfully represented in the standard ASCII character set. Thus use *iota* for the APL $\underline{\iota}$ function, *abs* for the APL $\underline{|}$ function, etc. This approach has the advantage of not causing an APL symbol to mean one thing on one system and another on a different system. Admittedly, some of the economy of syntax unique to APL is lost (though none of the power of expression . . .), but a side benefit of legibility to the novice exists since an APL program is readable without memorizing a long list of funny symbols. Indeed many novice APL programmers begin by writing their own named functions using APL primitives so that they can write, for example, *times* when they mean to multiply instead of APL's $\underline{\times}$.

So I suggest that Mr. Mueller re-evaluate his design and do us all a favor by helping to make APL available to the micro-world without confusing those who already have used (or will in the future use) APL on a system that supports the special characters.

Yours Truly,
Dr. John O. Aikin — The Evergreen State College
Dir. of Computer Services — Olympia, WA 98505

2311 DISC DRIVES FOR $350

Paul Perez (415-961-9146), a San Francisco Bay Area dealer in used computer equipment, announced at the July 20th Homebrew Computer Club meeting that he has dozens of properly functioning 2311-type hard-disc drives available for $350 (presumably plus shipping). Several people in both northern and southern California (and presumably elsewhere) are currently completing a controller for these 7¼ *megabyte* drives that will plug into a S-100 bus.

A STORY OF TRAFFIC LIGHTS, or: A BASIC-PROGRAM TO PLAY THE GAME OF LIFE

R. Broucke

The game of life is a solitaire game that consists in watching the evaluation of a population of cells on a checker board or better yet on the screen of a T.V. set connected to your computer. The game was invented in 1970 by Cambridge Mathematician John Conway and popularized by Martin Gardner in *Scientific American.* The whole game is ruled by three very simple genetic laws that define the survival, death and birth of the cells at each generation.

1. Cells with two or three neighbors survive.
2. Cells with less than two neighbors die because of loneliness. Cells with more than three neighbors die because of overpopulation.
3. A new cell is born in each empty space with exactly three live neighbors.

The diagonals as well as the horizontal and vertical rows are used to count the number of neighbors.

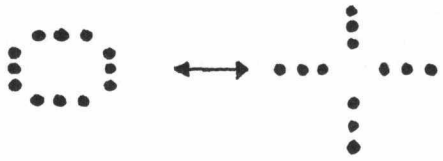
We will here describe very briefly our program written in MITS-BASIC for the ALTAIR-computer. It uses a Video output system to display the cells on a T.V. screen, (black = a cell and white = an empty space). It could be used with several of the existing video-systems, but we make extensive usage of the PEEK and POKE instructions of MITS-BASIC. All the graphic displays are performed in BASIC and no machine language software is used at all.

Our program is an adaptation of another program by Whipple and Arnold that was published in *Dr. Dobb's Journal* in January 1976. The listing is given below. The reader will have no difficulty adapting it to his own need and taste. Our program uses a board of 16 by 32. It starts with random initial conditions using the BASIC function RND to generate a random number between 0 and 1. We use an input number NI (see statement 105) to change the density of the initial population. NI should be anywhere from 0 to 1 and can be considered as the initial percentage of occupation of the 16x32 spaces. A one in the matrix G(or H) of the program corresponds to the presence of a cell while a zero is an empty space. The function INT(RND(1)+NI) determines the 0 or 1 to be stored in G, (statement 120). The statements 126 to 129 make the inital population matrix symmetric with respect to both the norizontal and vertical axes. The four statements 130 to 152 display the complete population on the screen. The statements 155 to 300 compute the new population (matrix H) according to the genetic laws. Finally, the last statements 310 to 330 move the matrix H back to the matrix G. The number 31744 (=31*2^{10}) that is seen at several places in the program is the initial address of our video board. This is the last 1K of memory bloc that can be accessed with PEEK and POKE in the 8K-BASIC, (31K to 32K). Our machine has 12K of RAM-memory.

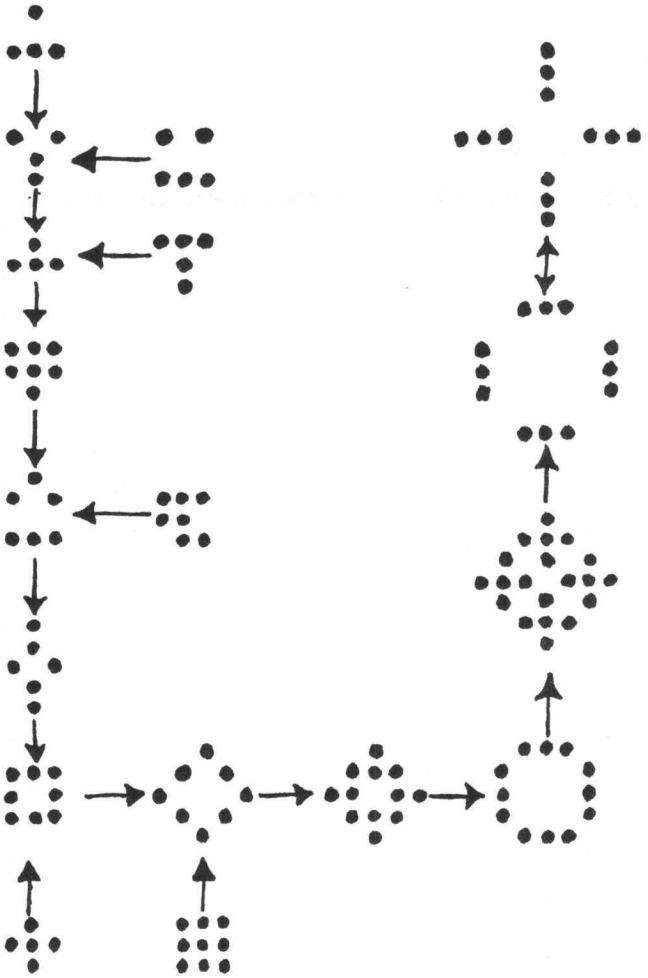
The above program has the only pupose of producing a pleasant-looking display on the T.V. screen. The user should also modify it to play a game which starts with a given simple input population. As an example, let us mention that the following seven initial populations will survive for several hundred's of generations before coming to some end:

It is known that several populations end with a stable pattern such as the bloc, the tub, the beehive, the pond, the boat, the ship, the barge, the snake, and so on. Some other populations end with an alternating pattern such as the blinker:

or the traffic lights which are composed of four blinkers:



The traffic lights are a very familiar terminal configuration in the game of life. In the following figure we show 16 populations which are all connected to each other and which end with the traffic lights, showing that computers can be and should be fun!



continued on page 19

CONVERSIONS BETWEEN OCTAL AND HEXADECIMAL

by R. Broucke

Some confusion exists among the small computer users because of the different systems of notations in programming. Not only is there a serious confusion concerning Mnemonics in assembly language programming but also in machine language programming where two systems are in wide usage: octal and hexadecimal. For this reason we want to show here a few formulas and tables to convert between octal and hexadecimal. Of course, if we examine the address lights on an Altair computer, for instance, it is easy to convert mentally between hex and octal, by just associating the lights in groups of four or in groups of three. Also, it is not difficult to convert first from octal to decimal, for instance and then from decimal to hex. Many simple rules are well known to convert to and from decimal. However, we will show here a few simple formulas that can be easily programmed in BASIC or used in the form of Tables for the direct conversion between hex and octal.

In the octal system the digits are 0, 1, 2, 3, 4, 5, 6 and 7 while in the hexadecimal system the best known notations represents the 16 digits by the 10 decimal digits, 0 to 9 followed by the first six letters of the alphabet, A, B, C, D, E, F stand for 10, 11, 12, 13, 14 and 15. Both the octal and hexadecimal systems are well suited for microcomputers such as ALTAIR and IMSAI or any computer with a one-byte word (8-bits) and a two-byte address (16 bits). An octal digit is a group of three binary bits (0 to 7) while a hexadecimal digit is a group of four binary bits (0 to 15). We represent a byte (0 to 255) in octal by three digits, $a_2\ a_1\ a_0$, and in hexadecimal by two hex digits $h_1\ h_0$.

Example: 255 (decimal) = 377 (octal) = FF (hex).

In general we use the formulas:

$$a_2\ a_1\ a_0 = a_2 \times 64 + a_1 \times 8 + a_0\ ;\ (a_i < 7),$$

$$h_1\ h_0 = h_1 \times 16 + h_0\ ;\ (h_i < 15).$$

Example: The address lights on my Altair show the octal number 132472. What is the Hex equivalent? We divide this number in two groups: 13 and 2472. The octal 13 is a Hex B. We will learn later that 2472 (octal) can be converted directly to 53A hex. The correct hex address displayed on the lights is thus B53A.

As was said above we show two methods of converting without actually writing out the binary representation of the numbers, the first with some small formulas and the second with Tables. Let the number be represented in octal by $a_3\ a_2\ a_1\ a_0$ and in hex by $h_2\ h_1\ h_0$, with the definition,

$$a_3\ a_2\ a_1\ a_0 = a_3 \times 512 + a_2 \times 64 + a_1 \times 8 + a_0\ ;\ (a_i < 7),$$

$$h_2\ h_1\ h_0 = h_2 \times 256 + h_1 \times 16 + h_0;\ (h_i < 15)$$

When the four octal digits $a_3\ a_2\ a_1\ a_0$ are given, the three hex digits can be computed with the formulas:

$$h_0 = a_0 + 8\ \text{REM}\ (a_1/2),$$

$$h_1 = \text{INT}\ (a_1/2) + 4\ \text{REM}\ (a_2/4),$$

$$h_2 = \text{INT}\ (a_2/4) + 2\ a_3.$$

In these formulas, INT indicates a standard function in BASIC: the integer division or quotient (for instance INT (19/4) = 4). Also REM represents a simple function: the remainder of the integer division, such as REM (19/4) = 3. The function REM (N/M) can be coded in BASIC by a statement such as REM = N - M * INT(N/M).

Similar formulas can be written for computing $a_3\ a_2\ a_1\ a_0$ when $h_2\ h_1\ h_0$ are given. They are-

$$a_0 = \text{REM}\ (h_0/8),$$

$$a_1 = \text{INT}\ (h_0/8) + 2\ \text{REM}\ (h_1/4),$$

$$a_2 = \text{INT}\ (h_1/4) + 4\ \text{REM}\ (h_2/2),$$

$$a_3 = \text{INT}\ (h_2/2).$$

As an example we can see that FAC (Hex) = 7654 (octal), because h_2 = F, h_1 = A, h_0 = C and also $a_3 = 7$, $a_2 = 6$, $a_1 = 5$ and $a_0 = 4$. The same results can be obtained with the use of the seven little tables below. There is one table for each of the four octal digits and each of the three hex digits.

To find a_0 :

h_0 =	0 8	1 9	2 A	3 B	4 C	5 D	6 E	7 F
a_0 =	0	1	2	3	4	5	6	7

To find a_1 :

h_1 ↓ \ h_0 →	0, 1, 2, 3 4, 5, 6, 7	8, 9, A, B, C, D, E, F
0, 4, 8, C	0	1
1, 5, 9, D	2	3
2, 6, A, E	4	5
3, 7, B, F	6	7

To find a_2 :

h_1 ↓ \ h_2 →	0, 2, 4, 6 8, A, C, E	1, 3, 5, 7 9, B, D, F
0, 1, 2, 3	0	4
4, 5, 6, 7	1	5
8, 9, A, B	2	6
C, D, E, F	3	7

To find a_3 :

h_2 =	0 1	2 3	4 5	6 7	8 9	A B	C D	E F
a_3 =	0	1	2	3	4	5	6	7

MORE

TDL SOFTWARE FOR DIGITAL GROUP Z-80's

News Release *Received: 77-Jul-20*

MICRO COM, a micro software specialty company, has announced the acquisition of a software licensing agreement from TECHNICAL DESIGN LABS (TDL). The software, available now, will be released in a format compatible with DIGITAL GROUP Z-80 hardware only. Software implementation has been achieved with user-oriented I/O drivers implemented on Digital Group hardware.

There are presently six software packages which are being offered. These packages are:

1) ZAPPLE BASIC – 8K
2) SUPER BASIC – 12K
3) TEXT EDITOR – 3K
4) WORD PROCESSOR – 3K
5) MACRO ASSEMBLER – 8K
6) SYSTEM MONITOR – 2K

Future TDL software releases, including the soon to be announced FORTRAN IV compiler, will also be offered. The software will be similar in price to that offered by TDL. For further details and pricing information write:

MICRO COM Inc.
P.O.B. 4069
Pompano Beach, FL 33063

To find h_0 :

a_1 \ a_0	0	1	2	3	4	5	6	7
0, 2 4, 6	0	1	2	3	4	5	6	7
1, 3 5, 7	8	9	A	B	C	D	E	F

To find h_1 :

a_1 \ a_2	0 4	1 5	2 6	3 7
0, 1	0	4	8	C
2, 3	1	5	9	D
4, 5	2	6	A	E
6, 7	3	7	B	F

To find h_2 :

a_2 \ a_3	0	1	2	3	4	5	6	7
0, 1 2, 3	0	2	4	6	8	A	C	E
4, 5 6, 7	1	3	5	7	9	B	D	F

FOR FREE: REPLACE YOUR BAUDOT TTY WITH AN ASCII TTY

For no more than the cost of shipping, you can probably replace your functioning Baudot Teletype with an ASCII machine – no hidden gotchas. It seems that there is a nationwide network set up for communication among the deaf, which has standardized on usage of the old (cheap) Baudot TTY's. Now that the hobbyists are grabbing up those old machines, they have come to be in short supply – a sad circumstance, indeed, since those using the DEAFNET *want and need* those Baudot machines.

A solution is at hand, however, that should please everyone involved. Some people involved with the DEAFNET have obtained a number of ASCII terminals – in fully finctional condition – that they will trade for Baudot machines.

For more details, contact Dave Fylstra, 415-326-6200x2816. (Dave is a researcher for Stanford Research Institute in Menlo Park, CA, is an active hobbyist, well known among S.F. Bay Area computer hackers, and is assisting in arranging these trades strictly as a service for users of the DEAFNET. No money is involved and no financial profit accrues to anyone. Considerable profit, however, in the form of good will and good feelings, is available to everyone involved.)

OSI OFFERS *BASIC* COMPUTER FOR $298 (PLUS TERMINAL & POWER SUPPLY)

News Release

Ohio Scientific's new Model 500 CPU board can be used as a stand-alone computer or as the CPU in a larger system. The board accepts 8K of ROM, 4K of RAM, 750 bytes of PROM, an ACIA based serial port, a 6502 processor, and full buffering for expansion!

The Model 500 is available completely assembled with 8K BASIC in ROM for $298.00. By adding a terminal and power supply, the user has a complete system which will accept up to 200 lines of BASIC program without expansion.

The Model 500 is software and hardware compatible with Ohio Scientific's 400 Series kits and Challenger products, thus, allowing expansion to a large system.

The board is available, enclosed with power supply, as the Model 500-1, and is available in an 8-slot Challenger case as the Model 500-8.

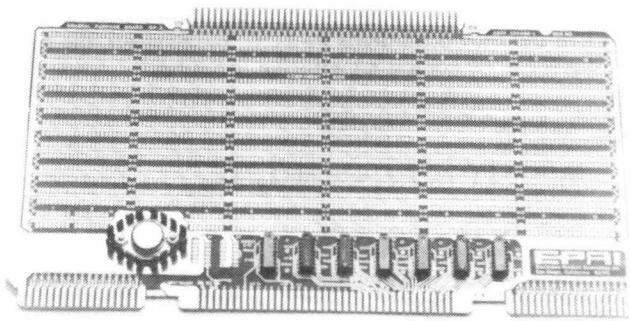
GENERAL PURPOSE PROTOTYPING BOARD FOR $170

News Release

Electronic Product Associates, Inc., announces the availability of a new Genreal Purpose Prototyping board for use in the Micro-68 Microprocessor systems. The 8" x 14.8" (20.32cm x 37.59cm) GP-2 board is exorcisor bus compatible and has complete bus buffering already established on-board with 8833 driver/receivers. The GP-2 board contains Vcc and ground busing, 3000 holes worth of blank dip patterns for thirty-five 24, 40 or 42 pin dip packages, or 107 14 or 16 pin dip packages. Price is $170.00 and they are available from stock.

For additional details contact:

Patti Neumann, Vice President
Electronic Product Associates, Inc.
1157 Vega Street
San Diego, CA 92110



SCHDL: A BASIC-coded 'DAILY REMINDER' FOR HOME AND OFFICE USE

Bob Moody and Steve Williams
Byte of Palo Alto - A Computer Store
2233 El Camino Real
Palo Alto, CA 94306
(415) 327-8080

77 Jun 8

Over the course of a year owning a computer store, I have come across a tremendous amount of software, some good and some not so good. The question that is asked of me a great deal is - now that I have this computer, what practical use can I put it to. So we at Byte of Palo Alto will be submitting a program a month to try and satisfy the need for useful applications for the home and office.

```
580 W=72
650 K$="" \ L$=CHR$(10)
RUN

1:INITIALIZE A FILE, 2:DISPLAY MONTH, 3:EXIT 1
FILE NAME?DATA
NAME OF MONTH?JULY
YEAR?1977
NUMBER OF DAYS?31
DAY OF WEEK OF FIRST DAY?6

1:INITIALIZE A FILE, 2:DISPLAY MONTH, 3:EXIT 2
ENTER FILE NAME: DATA
                        JULY 1977
     S         M         T         W         T         F         S
-------------------------------------------------------------------
!31       !         !         !         !         ! 1       ! 2       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
! 3       ! 4       ! 5       ! 6       ! 7       ! 8       ! 9       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
!10       !11       !12       !13       !14       !15       !16       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
!17       !18       !19       !20       !21       !22       !23       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
!24       !25       !26       !27       !28       !29       !30       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
ENTER DAY: 5   MESSAGE: DINNER JAN L-7:40
ENTER DAY: 8   MESSAGE: DEN-  TIST 3PM
ENTER DAY: 14   MESSAGE: JUNE  DR DOBB'S COMES
ENTER DAY: 21   MESSAGE: MEET  SALLY FOR LUNCH
ENTER DAY: 24   MESSAGE: MEET  SUSIE FOR LUNCH
ENTER DAY: 27   MESSAGE: MEET  SALLY'S  LAWYER
ENTER DAY: 0
1:INITIALIZE A FILE, 2:DISPLAY MONTH, 3:EXIT 2
ENTER FILE NAME: DATA
                        JULY 1977
     S         M         T         W         T         F         S
-------------------------------------------------------------------
!31       !         !         !         !         ! 1       ! 2       !
!         !         !         !         !         !         !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
! 3       ! 4       ! 5 DINNER! 6       ! 7       ! 8 DEN-  ! 9       !
!         !         ! JAN 7:40!         !         !TIST 3PM !         !
!         !         !         !         !         !         !         !
-------------------------------------------------------------------
!10       !11       !12       !13       !14 JUNE  !15       !16       !
!         !         !         !         !DR DOBB'S!         !         !
!         !         !         !         ! COMES   !         !         !
-------------------------------------------------------------------
!17       !18       !19       !20       !21 MEET  !22       !23       !
!         !         !         !         !SALLY FOR!         !         !
!         !         !         !         ! LUNCH   !         !         !
-------------------------------------------------------------------
!24 MEET  !25       !26       !27 MEET  !28       !29       !30       !
!SUSIE FOR!         !         !SALLY'S  !         !         !         !
! LUNCH   !         !         !LAWYER   !         !         !         !
-------------------------------------------------------------------
ENTER DAY: 0
1:INITIALIZE A FILE, 2:DISPLAY MONTH, 3:EXIT 3
STOP IN LINE 870
READY
```

```
?
*GO BASIC

READY
LOAD SCHDL

READY
NULL 1
READY
LIST

100 REM*******************************************************
110 REM                      SCHEDULE                         *
120 REM                   STEVE WILLIAMS                      *
130 REM                    MAY  27,1977                       *
140 REM*******************************************************
150 REM                          THIS PROGRAM WAS DESIGNED BY
160 REM BOB MOODY, AND WRITTEN BY STEVE WILLIAMS OF THE BYTE
170 REM SHOP OF PALO ALTO. IT WAS WRITTEN ON AN IMSAI 24K
180 REM SYSTEM WITH A NORTH STAR MINI-FLOPPY DISK, IN NORTH
190 REM STAR'S EXTENDED DISK BASIC. IN WRITING IT, I HAVE,
200 REM HOWEVER, DECIDED TO NOT USE SEVERAL OF NORTH STAR'S
210 REM BEST FEATURES, IN ORDER TO PROVIDE FLEXIBILITY FOR
220 REM HOBBYISTS NOT BLESSED WITH A DISK. IT CAN BE MODIFIED
230 REM TO RUN WITH ANY TERMINAL WIDTH, AND TO USE ANY CONTROL
240 REM CHARACTER (CLEAR, CLEAR LINE) COMBINATION, OR NONE.
250 REM                            SCHEDULE PROVIDES A
260 REM MEANS OF STORING AN ENTIRE MONTH'S SET OF MESSAGES
270 REM FOR RECALL ONTO A TERMINAL OR TELETYPE.
280 REM                          SEVERAL FUNNY THINGS HAVE
290 REM BEEN DONE TO ACCOMODATE THE NORTH-STAR DISK BASIC.
300 REM THESE ARE CONCERNED WITH THE LARGE STRING ARRAY
310 REM THAT CONTAINS ALL OF THE MESSAGES FOR THE MONTH
320 REM THAT IS CURRENTLY BEING DISPLAYED, THE FORMATTED
330 REM OUTPUT, AND DISK I/O.
340 REM                            CONCERNING  THE FORMER, THE
350 REM EXPRESSION 'A$(I,J)' IS EQUIVALENT TO THE I'TH THROUGH
360 REM J'TH CHARACTERS OF A$. THEREFORE, THE SUBSCRIPT
370 REM '((I-1)*MO+IO,(I-1)*MO+JO)' INDICATES THE IO'TH THROUGH
380 REM JO'TH CHARACTERS OF THE MESSAGE FOR DAY I . HENCE,
390 REM THE EXPRESSIONS IN LINES 1160 THROUGH 1230 ACCESS THE
400 REM PORTIONS OF THE MESSAGE STRING THAT PERTAIN TO
410 REM EACH DAY.
420 REM                              THE ONLY FORMATTED OUTPUT
430 REM THAT I HAVE USED IS FOR THE DAY NUMBERS THAT APPEAR
440 REM IN THE UPPER-LEFT-HAND CORNER OF THE DAY SQUARE. WHEN
450 REM ' ! %2I,---- ' OCCURS, ALL OF THE NUMERICAL PARAMETERS
460 REM IN THE PRINT STATEMENT WILL BE TRUNCATED AND PRINTED,
470 REM RIGHT-JUSTIFIED IN A 2-CHARACTER FIELD. (THE EXCLAMATION
480 REM POINT IS 'SHORTHAND' FOR 'PRINT' IN NORTHSTAR BASIC.)
490 REM                               I HAVE MADE THE ENTIRE SET
500 REM OF MESSAGES FOR ONE MONTH RESIDENT IN ON-LINE MEMORY
510 REM AT THE TIME THAT THE CALENDAR IS PRINTING. THIS IS SO
520 REM THAT PERSONS USING CASSETTE TAPE FOR OFF-LINE STORAGE
530 REM MAY EASILY LOAD THE DATA IN ONE OPERATION, AND SAVE IT
540 REM AS EASILY. IF THE EXTENSIVE RANDOM-ACCESS DISK I/O
550 REM CAPABILITIES OF NORTH STAR BASIC WERE USED, IT WOULD
560 REM PRECLUDE THE USE OF CASSETTE STORAGE.
570 REM W=MAX LENGTH OF LINE
580 W=72
590 CO=INT((W-8)/7) \ MO=3*CO-3
600 DIM D$(31*MO),B$(MO),B1$(MO),W$(7)
610 B1$="                                                  "
620 W$="SMTWTFS"
630 REM INITIALIZE THE FOLLOWING STRINGS FOR CLEAR AND
640 REM CLEAR LINE FOR YOUR HARDWARE CONFIGURATION.
650 K$="CLR SCR" \ L$="CLR LIN"
660 !K$,CHR$(0)
670 INPUT "1:INITIALIZE A FILE, 2:DISPLAY MONTH, 3:EXIT ",F
680 ON F GOTO 710,780,860
690 REM "M$:NAME OF MONTH, Y:YEAR, F:DAY OF WEEK OF FIRST DAY"
700 REM "D: # OF DAYS"
710 REM INITIALIZE A FILE TO DISPLAY ALL BLANKS
720 INPUT "FILE NAME?",A$ \ OPEN #0,A$ \ D$=""
730 FOR I=1 TO 31 \ D$=D$+B1$ \ NEXT
740 INPUT "NAME OF MONTH?",M$ \ INPUT "YEAR?",Y
750 INPUT "NUMBER OF DAYS?",D
760 INPUT "DAY OF WEEK OF FIRST DAY?",F
770 WRITE #0,M$,Y,F,D,D$ \ CLOSE #0 \ GOTO 660
780 REM DISPLAY MESSAGES FROM A FILE
790 INPUT "ENTER FILE NAME: ",A$ \ OPEN #0,A$
800 READ #0,M$,Y,F,D,D$ \ CLOSE #0
810 F9=0
820 GOSUB 880
830 IF F9=0 THEN 660
840 OPEN #0,A$ \ WRITE #0,M$,Y,F,D,D$ \ CLOSE #0
850 GOTO 660
860 STOP
870 REM SUBROUTINE REFRESHES MONTH SCREEN
880 !K$,TAB((W-LEN(M$)-5)/2),M$,Y
```

More

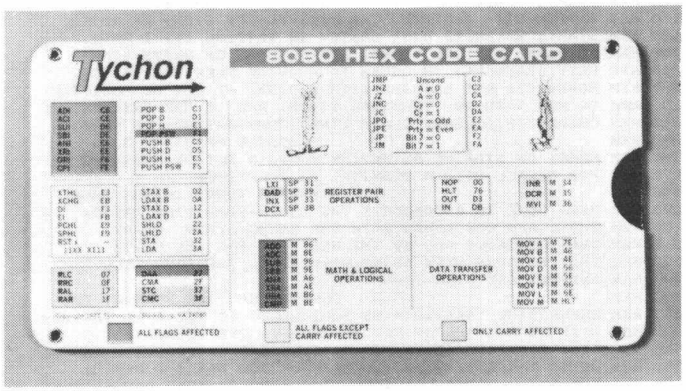
CLASSY 8080 CODE CARD FROM TYCHON

Tychon Incorporated is pleased to announce its 8080 Hex Code Card. The code card is a slide rule-like aid for programming and debugging 8080 software. It contains all the mnemonics and their corresponding hexadecimal codes. The instructions are all color coded to indicate which flags are affected during execution. The pocket sized card measures 6.5 by 3 inches and it provides the instructions in a neat, logical format for quick reference.

The back side of the card is printed with an ASCII code chart for all 128 characters plus the 8080 status word and register pair codes.

Delivery of the 8080 Hex Code Card is immediate and the price is $2.95 postpaid. Quantity discounts start at ten units and custom printing is also available.

For further information write or call C.A. Titus at Tychon, Inc., P.O. Box 242, Blacksburg, VA 24060, (703) 951-9030.




```
890 FOR I=1 TO 7 \ !TAB((CO+1)*I-CO/2),W$(I,I), \ NEXT \ !
900 GOSUB 1160
910 REM PRINT FIRST WEEK
920 FOR K=1 TO 3 \ !"!", \ IF D+F<36 THEN 940 \ FOR I=37-F TO D
930 GOSUB 1140 \ NEXT
940 GOSUB 1240 \ FOR I=1 TO 8-F \ GOSUB 1140 \ NEXT \ ! \ NEXT
950 GOSUB 1160
960 REM PRINT SECOND, THIRD, AND FOURTH WEEKS
970 FOR J=9-F TO 23-F STEP 7 \ FOR K=1 TO 3
980 !"!", \ FOR I=J TO J+6
990 GOSUB 1140 \ NEXT \ ! \ NEXT \ GOSUB 1160 \ NEXT
1000 REM PRINT FIFTH AND LAST WEEK
1010 FOR K=1 TO 3 \ !"!",
1020 IF D+F<36 THEN J=D ELSE J=36-F \ FOR I=30-F TO J
1030 GOSUB 1140 \ NEXT \ FOR I=D+F TO 35
1040 !B1$(1,CO),"!", \ NEXT \ ! \ NEXT
1050 GOSUB 1160
1060 REM GET MESSAGES TO PUT IN THE CALENDER
1070 INPUT1 "ENTER DAY: ",I \ IF I<1 OR I>D THEN RETURN \ F9=1
1080 INPUT1 "   MESSAGE: ",B$ \ !CHR$(13)+L$,
1090 B$=B$+B1$
1100 D$((I-1)*MO+1,I*MO)=B$
1110 GOTO 1070
1120 RETURN
1130 REM SUBROUTINE JUMPS TO CORRECT ROUTINE
1140 ON K GOTO 1180,1200,1220
1150 REM SUBROUTINE DOES A LINE OF -'S
1160 FOR I=1 TO 7*CO+8 \ !"-", \ NEXT \ ! \ RETURN
1170 REM SUBROUTINE PRINTS FIRST LINE OF DAY I
1180 ! %2I,I," ",D$((I-1)*MO+1,(I-1)*MO+CO-3),"!", \ RETURN
1190 REM SUBROUTINE PRINTS SECOND LINE OF DAY I
1200 ! D$((I-1)*MO+CO-2,(I-1)*MO+2*CO-3),"!", \ RETURN
1210 REM SUBROUTINE PRINTS THIRD LINE OF DAY I
1220 ! D$((I-1)*MO+2*CO-2,I*MO),"!", \ RETURN
1230 REM SUBROUTINE PRINTS THE PROPER NUMBER OF BLANK SQUARES
1240 IF D+F-35<0 THEN J=F-1 ELSE J=35-D
1250 FOR I=1 TO J \ !B1$(1,CO),"!", \ NEXT
1260 RETURN
READY
```

$500 LABEL-MAKER SOFTWARE RUNS ON NORTH STAR MICROFLOPPYS, INCLUDES SORT PKG

News Release from Tylog *Received: 77-Jul-14*

MIAMI, FL.: Tylog Systems is proud to announce the introduction of its first microcomputer software package, the Micro-Label System.

The Micro-Label System is a general purpose mailing label system for the creation and maintenance of mailing label lists and other similar label applications (such as inventory bin labels, etc.). The system is designed to be run by non-technical personnel. Extensive documentation and procedures are provided to facilitate operation of the system.

The software is designed to run on an 8080 processor using dual North Star Computers, Inc. minifloppyTM disk drives and a pin feed or tractor feed impact printer. Additions, changes and deletions are applied to the master file via CRT. The Micro-Label System permits the label files to be printed in more than one sequence. Error editing and restart capabilities are also provided. Each master file can hold over 540 records with multiple files being used for longer lists.

Tylog warranties the software for 90 days. The warranty includes new release updates at no extra charge. Maintenance contracts are available for post warranty support. The Micro-Label System is also available as a turnkey package, complete with all hardware.

Tylog Systems has inventory, accounting, word processing and lawyer's systems now under development. For details, prices and your nearest Tylog dealer write:

Tylog Systems, Inc.
9805 SW 152 Terrace
Miami, FL 33157

Description:
General purpose label print system.
Programs:
7 modules:
Update Module — Add, change and delete to an existing master file or create a new file.
Extract Module — Create a key file by name, zip code or zip and name.
Sort Module — Sort key files.
Reorganization Module — Reorganize master file and remove deleted records.
Print Module — Print mailing labels. Restart at any record.
Copy Module — General purpose file copy.
Read Module — General purpose file read and print.
Standard File Organization:
Master File:
* Fixed length records, randomly placed.
* Length 120 bytes.
Last name 15 bytes
First name and MI 14 bytes
Address lines 1 & 2, each line 30 bytes
City 20 bytes
State 2 bytes
Zip code 6 bytes

Key file:
* Fixed length records, sequentially placed
* Length normally 35 bytes (dependent on extract type).

Standard Sort Keys:
* Last name, first name
* Zip code
* Zip code, last name, first name

Standard Label Format:
Any length print line up to 132 columns, 1 to 20 labels across. Labels must be spaced one inch from the top of one label to the top of the next. Other formats are optional.

Hardware Required:
* Microcomputer — 8080 or Z80 with 20 K of memory.
* Video terminal — 16 lines, 64 char. min.; w/screen clear.
* Pin or tractor feed printer.
* Dual North Star Computers, Inc. minifloppyTM disk drives.

Program Run Times:
* Update response time — 8 seconds max.
* Sort time — 30 minutes max.
* Extract time — 5 minutes max.
* Reorganization time — 12 minutes max.
* Print time — a function of printer speed.

Options:
* Custom record formats.
* Custom sorts.
* Custom label height.

Other modifications by separate negotiations.
Software Medium:
North Star compatible minifloppyTM disk

Prices:

Standard Micro-Label System software	$500
Options — each	$40

CURVE PLOTTING ON A T.V. SCREEN WITH THE POLYMORPHICS VIDEO INTERFACE AND MITS BASIC

by R. Broucke, Associate Professor of Aerospace Engineering, University of Texas, Austin, Texas

1. The Polymorphics Video Interface

Early in 1976, Polymorphic Systems produced the Video Terminal Interface, (VTI), allowing a microcomputer to display the content of a kilobyte of memory on a regular Video Monitor or even on a regular television set. The VTI is a single board which is directly compatible with the 8080-based Altair and Imsai Computers as well as the POLY-8080 marketed by Polymorphics. It contains a read-only-memory character generator IC (MCM6571AL) capable of generating the 7x9 dot matrix of 192 different characters. Among these characters are the regular lower case alphabet, the capitals, the greek alphabet, many special signs such as $\surd$, $\uparrow$, $\Leftarrow$, $\Rightarrow$ and finally 64 special graphic signs. It is our intention here to describe the usage of these special signs for displaying curves on the T.V. screen. It is not possible with this system to display smooth continuous curves as are seen on an oscilloscope, due to the fact that only finite size square black-and-white blocs are available with the VTI system. Let us thus first say a few words about these questions.

The VTI board is normally available in two versions. The so-called minimum version displays 512 bytes of memory (four 2111's or 9111's) on 16 lines of 32 characters. Addition of four more 2111's allows us to display 16 lines of 64 characters each. The 64 special graphics signs are composed of six black or white blocs, ($2^6 = 64$) in a configuration of three blocs high and two blocs wide. With a minimum board the best resolution that can be obtained is thus 32 x 2 = 64 blocs horizontally by 16 x 3 = 48 blocs vertically. Let us also mention that in terms of television resolution each one of these blocs is five dots by five dots.

The 64 graphic signs are normally numbered from 0 to 63 (decimal) or from 0 to 77 (octal). For instance, the sign in Figure 1 is normally number 25 (octal) or 21 (decimal). The sign 0 is all white while 63 is black. In using these signs for curve-plotting we will use only six of the 64 available signs. They are the six characters that have only a single black bloc. Their identification numbers are 1, 2, 4, 10, 20 and 40 (octal). Let us note that more complicated graphics signs can be constructed by logically OR-ing the above six signs with each other. This will turn out to be important in our software.

Figure 1

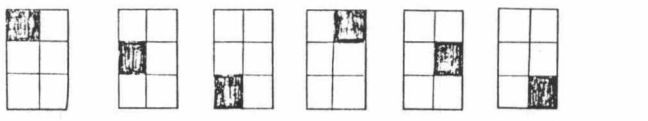


Figure 2

2. The Screen Coordinates

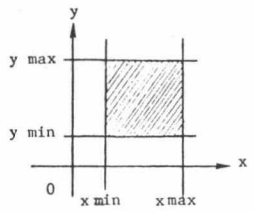


Figure 3

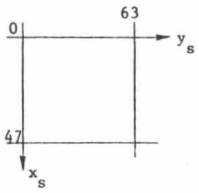


Figure 4

Before a curve can be plotted for a function such as, for instance, y = sin(x), it is necessary for the user to define the upper and lower limits for both the abscissa x and the ordinate y. In other words, we assume the four numbers x_{min}, and x_{max} and y_{min}, y_{max} to be given by the user. This defines the user's plotting surface and coordinate system as shown in Figure 3.

It will be convenient to introduce a new coordinate system of screen coordinates (x_s, y_s) as shown in Figure 4. As we see, x_s is an integer increasing from 0 to 47 when we move down on the screen, while y_s is a integer increasing from 0 to 63, left to right on the screen. This system is particularly easy to use for plotting with the VTI board. We assume here the minimum version with 32 characters on a line. The reader will have no difficulty adapting our formulas to the 64-character line (y_s from 0 to 127).

The conversion from the user coordinates (x,y) to the screen coordinates (x_s, y_s) can be easily performed with the formulas

$$x_s = \text{INT}\left[47\left(\frac{y_{max} - y}{y_{max} - y_{min}}\right)\right]; \quad y_s = \text{INT}\left[63\left(\frac{x - x_{min}}{x_{max} - x_{min}}\right)\right].$$

where INT means that we take the integer part of this number. It should be noted that x_s and y_s are the coordinates of a particular black block on the screen, corresponding to a point (x,y). Next we transform x_s to the line number x_s' and y_s to the character number y_s' by the formulas

$$x_s' = \text{INT}\left(\frac{x_s}{3}\right); \quad y_s' = \text{INT}\left(\frac{y_s}{2}\right),$$

in consequence of the facts that we have three blocs per line and two blocs per character. Finally, we have to determine the address in the screen memory of the byte containing the representation of the point (x,y). Let us assume that A_0 is the address of the first location of the video memory. This corresponds to the point $x_s = y_s = 0$ and is in the upper left corner on the screen. In our present system we use A_0 = 31744 (decimal) = 76000 (octal). It is known that A_0 should always be a multiple of $2^{10} = 1024$. In our case $A_0 = 31 * 2^{10}$. This gives us the last one K of memory that can be used with PEEK and POKE in the MITS 8K-BASIC language. For any given point (x,y), the memory address can then be computed with the formula

$$A = A_0 + 64x_s' + y_s'.$$

Note that the constant 64 in the above formula is the same constant whether the system has 32 or 64 characters per line.

3. Selecting the Appropriate Bloc

Let us now consider the question of choosing the appropriate bloc, among the six shown in Figure 2, corresponding to a given point (x,y). First we will determine if a left bloc (octal 40, 20 or 10) has to be used or a right bloc (octal, 4, 2 or 1), according to the ID's given in Figure 2. We can verify that a right or left code *a* can be computered as follows

$$a = y_S - 2*(y_S/2) = y_S(\text{MOD } 2).$$

When a=0 we have a left bloc, while when a=1 we will have a right bloc. Next we have to make the selection between an upper, middle or lower bloc according to the code

$$\beta = x_S - 3*(x_S/3) = x_S(\text{MOD } 3).$$

The values 0, 1 and 2 respectively correspond to the upper, middle and lower blocs. The ID number 1,2,4,8,16 or 32 can thus be determined as a function of a(=AL) and B(=BE) with the use of three lines of BASIC programming:

```
130 ID=4: IF BE=1 THEN ID=2
140 IF BE=2 THEN ID=1
150 IF AL=0 THEN ID=INT(8*ID + 0.5) .
```

The constant 0.5 is added in the last statement in order to force the integer value (INT) of a number such as 7.99999 to be 8 rather than 7. Let us also note that the above three statements can be replaced by a single statement (which may however result in slower execution times) such as:

```
130 ID=INT (8-7*AL)*2↑(2-BE)+0.5)
```

or also the following statement

```
130 ID=INT(2↑(5-3*AL-BE)+0.5) .
```

As was said above, this single statement may be slightly slower due to the presence of the exponent. We again add a quantity 0.5 before taking the integer part because of the possibility of a negative round-off error in the exponent subroutine.

To put the appropriate block ID on the screen at the address A(=AD) we could use a statement such as

```
170 POKE AD, ID.
```

However, this destroys the five neighboring blocs and is is thus better to "OR" the new bloc with whatever is already present:

```
170 POKE AD, PEEK(AD) OR ID.
```

To do this, one should initially erase the whole screen. This can be done with a single line of coding, of the following type:

```
100 FOR I=31744 TO 32767: POKE I,O:NEXT I.
```

4. Some Examples of Plotting Programs

To finish this text we display below two short program which illustrate all the principles which have been layed out in the previous sections. The sample program plots the function y=sin(x), (statement 117). The statements 102 and 103 give the upper and lower limits of x and y: X1, X2, Y1 and Y2. The statement 104 determines another input number NP which is the number of points to be plotted or the vertical resolution on the screen. All the other statements have been previously explained.

```
100 FOR I=31744 TO 32767
101 POKE I,0 : NEXT I
102 X1=-6:X2=6
103 Y1=-1.2:Y2=+1.2
104 NP=100
105 DE=(X2-X1)/NP
106 FOR X=X1 TO X2 STEP DE
117 Y=SIN(X)
120 XS=INT(47*(Y2-Y)/(Y2-Y1))
122 YS=INT(63*(X-X1)/(X2-X1))
124 AL=INT(YS-2*INT(YS/2))
126 BE=INT(XS-3*INT(XS/3))
130 ID=INT(2↑(5-3*AL-BE)+.5)
140 AD=INT(31744+64*INT(XS/3)+YS/2)
150 POKE AD,PEEK(AD) OR ID
160 NEXT X
170 END
```

In the second program we show how to produce a curve from a table of points, rather than a mathematical function given by a formula. The points could have been obtained for instance as results of experimental measurements, or in any other way such as the step-by-step integration of differential equations. In the program listing we see that XT is the table with abscissa while YT is the table with ordinates. The number of points is NP. This is the only input needed for this program. It is seen that this program automatically determines the lower and upper limits (X1, X2, Y1, Y2) of the data.

```
50 DIM XT(100),YT(100)
55 NP=100
60 FOR I=1 TO NP
65 XT(I)=I
70 YT(I)=SQR(I)+10*SIN(I/10)
75 NEXT I
100 FOR I=31744 TO 32767
105 POKE I,0 : NEXT I
110 X1=XT(1):X2=XT(1)
111 Y1=YT(1):Y2=YT(1)
112 FOR I=2 TO NP
113 IF XT(I) < X1 THEN X1=XT(I)
114 IF XT(I) > X2 THEN X2=XT(I)
115 IF YT(I) < Y1 THEN Y1=YT(I)
116 IF YT(I) > Y2 THEN Y2=YT(I)
117 NEXT I
118 FOR I=1 TO NP
119 X=XT(I):Y=YT(I)
120 XS=INT(47*(Y2-Y)/(Y2-Y1))
122 YS=INT(63*(X-X1)/(X2-X1))
124 AL=INT(YS-2*INT(YS/2))
126 BE=INT(XS-3*INT(XS/3))
130 ID=INT((8-7*AL)*2↑(2-BE)+.5)
140 AD=INT(31744+64*INT(XS/3)+YS/2)
150 POKE AD,PEEK(AD) OR ID
160 NEXT I
170 END
```

5. Conclusions

We hope having succeeded in showing two points in this short text. First of all that the Polymorphic System Video Interface is a powerful tool for the display of curves in a large number of varied applications. Secondly that the MITS-BASIC language is an extremely flexible programming language for the solution of engineering and scientific problems and their graphic display with a microcomputer.

continued from page 12

```
100 DIM G(15,31),H(15,31)
105 NI=0.42
110 FORI=1 TO 14:FOR J=1 TO 30
120 G(I,J)=INT(RND(1)+NI)
125 NEXT J:NEXT I
126 FOR I=0 TO 7:FOR J=0 TO 15:G(I,31-J)=G(I,J)
127 NEXT J:NEXT I
128 FOR J=0 TO   31:FOR I=0TO 7:G(15-I,J)=G(I,J)
129 NEXT I:NEXT J
130 FOR I=0 TO 15:FOR J=0 TO 31
140 AD=INT(31744+64*I+J+0.5)
150 POKE AD,INT(63*G(I,J)+0.5)
152 NEXT J:NEXT I
155 FOR I= 1 TO 14:FOR J= 1 TO 30
160 N=G(I-1,J-1)+G(I,J-1)+G(I+1,J-1)+G(I-1,J)+G(I+1,J)
170 N=G(I-1,J+1)+G(I,J+1)+G(I+1,J+1)+N
172 N=INT(N+0.5)
180 IF G(I,J)=0 THEN 250
190 IF N=2 OR N=3 THEN 230
200 H(I,J)=0
210 POKE INT(31744+64*I+J+0.5),0
220 GOTO 300
230 H(I,J)=1
240 GOTO 300
250 IF N<>3 THEN 200
260 H(I,J)=1
270 POKE INT(31744+64*I+J+0.5),63
280 GOTO 300
290 H(I,J)=0
300 NEXT J:NEXT I
310 FOR I= 1 TO 14: FOR J=1 TO 30
320 G(I,J)=H(I,J)
330 NEXT J:NEXT I
340 GOTO 155
350 END
```

6800 CIRCLE MAKER

Dear Dr. Dobb, October 25, 1976

The enclosed subroutine for an M6800 will generate a circle of given center and radius on an x-y display (e.g. scope). It uses two 8-bit D/A's connected to a PIA. Points on the "circle" are computed by the following algorithm attributed to Marvin Minsky in MIT AI Memo No. 239:

$$X(N+1) = X(N) + Y(N)/K$$
$$Y(N+1) = Y(N) - X(N+1)/K$$

An interesting discussion of this and a few other nifty graphing algorithms can be found in an article by B.K.P. Horn, "Circle Generators for Display Devices", *Computer Graphics and Image Processing* 5, 280-288 (1976). If one were to call this subroutine repeatedly, each time specifying a new center, the circle would appear to move. Then, with only a minimal amount of cleverness, one might invent the game of Pong.

Also, for anyone interested, I have written a PDP-11 cross assembler for the M6800. It's written in MACRO-11 which produces octal listings; however, I have a FORTRAN program which will convert LST files into hex format (see enclosed listing). Copies will be free (please pay postage) unless demand is too great.

David Hudson
3513 Wabash Ave.
Granite City, IL 62040

P.S. Things I would like to see:
LISP for M6800
Two dimensional FFT for M6800
Articles on machine vision and robotics
Information on computer control of model railroads
Video to digital converter for image processing

```
                        ; CIRCLE DRAWING SUBROUTINE
                        ; CALL WITH CENTER IN CX,CY
                        ; AND RADIUS IN R:
                        ;  0<=CX<=255
                        ;  0<=CY<=255
                        ;  0<=R<=127
                PIAD    = 100010
0000 00         CX:     .BYTE 0
0001 00         CY:     .BYTE 0
0002 00         R:      .BYTE 0
0003 0000       XX:     .WORD 0
0005 0000       YY:     .WORD 0
0007 CE 8008    CINIT:  LDX PIAD,I      ;INITIALIZE PIA
000A 6F 08              CLR 1,X         ;DATA REGISTERS AS
000C 6F 03              CLR 3,X         ;OUTPUTS
000E 86 FF              LDA A,377,I
0010 A7 00              STA A,0,X
0012 A7 02              STA A,2,X
0014 86 04              LDA A,4,I
0016 A7 01              STA A,1,X
0018 A7 03              STA A,3,X
001A 39                 RTS
001B 96 02      CIRCLE: LDA A,R,D       ;INITIALIZE X AND Y
001D 97 05              STA A,YY,D      ;OFFSET VECTORS
001F 4F                 CLR A
0020 97 06              STA A,YY+1,D
0022 97 03              STA A,XX,D
0024 97 04              STA A,XX+1,D
0026 96 05      CLP:    LDA A,YY,D      ;A <- Y/256
0028 2B 08              BMI CYNEG       ;A<0 ?
002A 9B 04              ADD A,XX+1,D    ;NO.  EXTEND SIGN
002C D6 03              LDA B,XX,D      ;B,A <- X + Y/256
002E C9 00              ADC B,0,I
0030 20 06              BRA XSAV
0032 9B 04      CYNEG:  ADD A,XX+1,D    ;YES  EXTEND SIGN
0034 D6 03              LDA B,XX,D      ;B,A <- X + Y/256
```

```
 43 0036                                ADC B,377,I
        C9  FF
 44 0038                        XSAV:   STA B,XX,D      ;SAVE NEW X
        D7  03
 45 003A                                STA A,XX+1,D
        97  04
 46 003C                                ADD B,CX,D      ; B  <-  CX + X(HI)
        DB  00
 47 003E                                STA B,PIAD      ;OUTPUT B TO X D/A
        F7  80  08
 48 0041                                LDA A,YY+1,D
        96  06
 49 0043                                LDA B,XX,D      ; B  <-  X/256.
        D6  03
 50 0045                                BMI CXNEG       ; B<0 ?
        2B  07
 51 0047                                SBA             ;NO.  EXTEND SIGN
        10
 52 0048                                LDA B,YY,D      ; B,A  <-  Y - X/256.
        D6  05
 53 004A                                SBC B,0,I
        C2  00
 54 004C                                BRA YSAV
        20  05
 55 004E                        CXNEG:  SBA             ;YES.  EXTEND SIGN
        10
 56 004F                                LDA B,YY,D      ; B,A  <-  Y - X/256.
        D6  05
 57 0051                                SBC B,377,I
        C2  FF
 58 0053                        YSAV:   STA B,YY,D      ;SAVE NEW Y
        D7  05
 59 0055                                STA A,YY+1,D
        97  06
 60 0057                                ADD B,CY,D      ; B  <-  CY + Y(HI)
        DB  01
 61 0059                                STA B,PIAD+2    ;OUTPUT B TO Y D/A
        F7  80  0A
 62 005C                                LDA A,XX,D      ;ALGORITHM IS PERIODIC
        96  03
 63 005E                                BNE CLP         ;SO CHECK TO SEE
        26  C6
 64 0060                                LDA A,R,D       ;WHEN WE'RE CLOSE
        96  02
 65 0062                                CMP A,YY,D      ;ENOUGH TO WHERE
        91  05
 66 0064                                BNE CLP         ;WE STARTED
        26  C0
 67 0066                                LSR A
        44
 68 0067                                CMP A,XX+1,D
        91  04
 69 0069                                BLO CLP
        25  BB
 70 006B                                CMP A,YY+1,D
        91  06
 71 006D                                BLO CLP
        25  B7
 72 006F                                RTS             ;RETURN AFTER ONE
        39
 73                                                     ;COMPLETE CYCLE
 74
 75          0001                        END
```

```
.MAIN.  RT-11 MACRO VM02-12   25-OCT-76 06:02:03 PAGE 9+
SYMBOL TABLE

A      =%0001  B      =%0002  CINIT   0007  CIRCLE  001B  CLP     0026
CX       0000  CXNEG    004E  CY      0001  CYNEG   0032  D     = 0002
I      = 0001  PIAD   = 8008  R       0002  SYM   = 0001  X     = 0000
XSAV     0038  XX       0003  YSAV    0053  YY      0005  . . A = 0000
. . AD1 = FFB7  . . B  = 0000  . . C = 0000  . . MM = 0010  . . OP = 0025

  ABS.    0070   000
          000000     001
ERRORS DETECTED:  0
FREE CORE:  851.  WORDS

.CIRCLE<SY3:M68.ASM,CIRCLE
```

EPA FLOPPY DISK SYSTEM FEATURES STATUS PANEL AND WRITE PROTECT SWITCHES

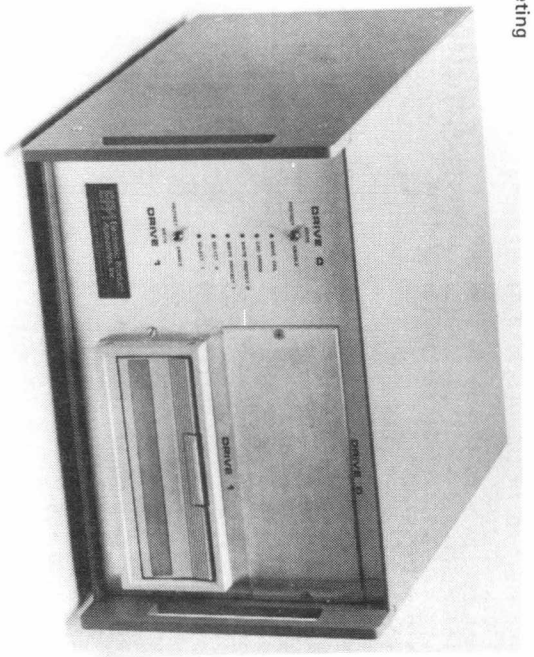
News Release

Received: 77 Jun 13

The EPA Micro-68b Floppy Disk System has been updated to include a disk status monitoring panel and write protect switches. Available in either single ($2495.00) or dual ($3295.00) configurations, the Micro-68b disk is complete and includes the interface necessary to plug in to either the 6800 exorcisor or the 8080 S-100 bus. The write protect switches prevent accidental destruction of valuable files. The status panel contains indicators for drive selection, write protection, CRC error and disk status. In many applications, it is important to visually monitor these functions so that disk activity (or lack thereof) is apparent. Housed in a ruggedized, light blue, aluminum cabinet to match the Micro-68b computer. Available from stock to two weeks.

For further information please contact:

Patti Neumann, Vice President Marketing
Electronic Product Associates, Inc.
1157 Vega Street
San Diego, CA 92110
(714) 276-8911



DOMPIER PLUS: A "MUSICAL KEYBOARD"

Bruce Hicks
Department of Secondary Education
University of Illinois
Urbana, Illinois 61801

77 June 20

Part of my interest in art and music on microcomputers is in finding educational uses that require only simple software and few peripherals. I was therefore particularly interested in Steve Dompier's music program and modifications to it. (Dr. Dobbs, v. 1, (1976): no. 2, pp. 6-7; no. 3, p.6; no. 8, p. 32.) his program plays a melody that has been entered by hand into memory, using a radio to pick up the music-modulated rf emanating from the computer.

I describe here a simple extension of Dompier's program which makes a computer act like a piano keyboard plus recording device, so that melodies can be played, recorded in the memory and played back. We are finding that this is a valuable educational tool in our bag of educational applications of microcomputers that we discuss with students and teachers.

Since a melody is stored as bytes in memory it is easy also to store on cassette tape. (Note the great compactness of this (primitive) type of audio recording.) It would be possible, using many unused keycodes, to add editing, recording of more than one melody and easier control of rest and note duration. But in view of the many synthesizers and software packages available for music production I do not think it is profitable to extend Dompier's program much further.

Operation

1. Start program.
2. Play a melody, using the keyboard keys in the same arrangement as the piano keyboard. The melody is recorded automatically.
3. At the end of the melody type / which is recorded as an end-of-melody marker.
4. Hit any key to play back the melody. To play back again, hit any key but RETURN.
5. Hit RETURN to return to (2) to play a new melody (which will overwrite the old one in memory).

Comments

The duration of each note or rest (space bar) is fixed by the constant at NOTE in the program. If I push a key before a note or rest is finished I get a "hold" or legato effect - not a convenient way to play complex rhythms but rather fun to do, and with some potential for pitch and rhythm training.

I used the keys

```
Q W E R T Y U I
 A S D F G H J K
  Z X C V B N M ,
```

for three octaves of white keys (key of C). The data for one row of keys is shown in Table I.

Each of the key codes is a relative address in the count table and thus points to the proper Dompier note period or count in this table in memory. I would be able to play two full octaves with all of the white *and* black keys by making the appropriate entries in the count table for two rows of keys (a row of eight "white" keys, then one of five "black" keys) positioned like the white and black keys on a piano board, and then similar entries for a second octave using two rows of keys.

The program is written in IMS 8080 Assembler.

```
LIST
0040  ORG 2200H
0097  JMP L1
0100 WAIT  IN 03          key input?
0200  ANI 02
0300  JZ WAIT             no
0400  IN 02               key code
0450  ANI 7FH
0500  MOV B,A             save in B
0600  RET
0800 L1 LXI H,XXX         tune table
0900 MM2 CALL WAIT
0910  XCHG                save tune table pointer
0920  LXI H,YYY
0922  MOV C,B
092A  MVI B,00H
0930  DAD B               count table address
0940  MOV B,M             count
0950  XCHG
0960  MOV M,B             count to tune table
0970  SUI 02FH            /
0980  JZ L2               end?
0990  CALL NOTE           play note
1000  INR L               step tune table address
1005  JMP MM2             next key
1010 L2 CALL WAIT         any key to play tune once
1020 L3 CALL TUNE
1030  CALL WAIT
1040  MOV A,B
1050  CPI 0DH             CR?
1060  JZ L1               play next tune
1070  JMP L3              any other key, repeat tune
2000 NOTE MVI D,40H       the core of Dompier's program
2010 L11 DCR B
2020  JNZ L12
2030  MOV B,M
2040 L12 DCR C
2050  JNZ L11
2060  DCR D
2070  JNZ L11
2080  RET
3000 TUNE LXI H,XXX       start of tune
3010 L21 MOV A,M          get "note" to test
3020  CPI 0FFH            end?
3030  RZ                  out
3040  CALL NOTE           play it
3050  INR L               step tune table address
3060  JMP L21             next note
3070  END
```

Bruce Hicks
Department of Secondary Education
University of Illinois
Urbana, Illinois 61801

keyboard keys	Z	X	C	V	B	N	M	,	space	/	
ASCII keycodes	5A	58	43	56	42	4E	4D	2C	2Ø	2F	H
notes (low octave)	C	D	E	F	G	A	B	C			
counts* (proportional to note periods)	AA	98	89	8Ø	72	66	5A	55	Ø2	FF	H

* Entries in count table

PERMANENT SOLUTION TO MISSING SP AND PSW DEFINITIONS IN PTC SYS

Bruce R. Kendall
334A Camillo Ct.
Mountain View CA 94040

77 June 8

I was reading thru Vol. 2, No. 3 when I ran across Raymond Jurgen's comment (about Jef Raskin's comment) on how to implement POP/PUSH PSW in Processor Tech's (Microtec) Sys8 resident editor-assembler. A nicer way to fix this problem is to define PSW and SP with two equates:

```
SP      EQU     6
PSW     EQU     6
```

This fixes all references to SP and PSW. I learned of this from examples in the Sys8 documentation that I obtained from the Homebrew Computer Club library.

I recently made some modifications to my copy of Sys8 in which I implemented a permanent fix to this problem. I first thought that all I had to do was to add SP and PSW to the internal table that defines the registers letters (A, B, . . . E, M). This route is too messy since the table is designed only for single character labels. Thus I chose to fool the system by preloading the main (RAM) symbol table with the above definitions of SP and PSW at the start of the assembly process. Sys8 contains a memory block moving routine (LMOV) that easily implements this function.

The symbol count is initialized near the beginning of the assembly process, so the first patch must be made at this point. In my copy of Sys8 this occurrs near F671 (Hex):

```
F671:       ASM5    STA     AERR    *  set error switch
                    XRA     A       *  get a zero
                    STA     NOLA    *  initialize label count
            ASM3    STA     PASI    *  set pass indicator
```

I patched this to read:

```
F671:       ASM5    JMP     FIXT    *  jump to patch routine
                    XRA     A
```

and added my fix routine at the end of the Sys8 (say at FD30). There are several hundred bytes left at the end (in the same 4K block) that can be used to add lots of features to your system. My fix routine:

```
FIXT        STA     AERR    *  set error switch
            MVI     A,2     *
            STA     NOLA    *  set label count to two
*** set pointers for LMOV routine ******* **********
            LXI     H,SYMT  *  point to start of symbol table
            LXI     D,FTAB  *  point to fix table
            MVI     C,ØFFH  *  LMOV stop character
*
            CALL    LMOV    *  copy FTAB to SYMT
            XRA     A       *  clear to initialize pass indicator
            JMP     ASM3    *  return to assembly flow
*
*
AERR        EQU     ØD06AH  *  error flag location
NOLA        EQU     ØD074H  *  label counter
SYMT        EQU     ØD0FFH  *  symbol table address
ASM3        EQU     ØF678H  *
LMOV        EQU     ØF534H  *  block move routine
*
*
FTAB        DB      'P'     *  five character label
            DB      'S'
            DB      'W'
            DW      Ø
            DB      Ø       *  two byte definition
            DB      6       *
            DB      'S'
            DB      'P'
            DW      Ø
            DW      Ø
            DB      6
            DB      ØFFH    *  LMOV stop character
*
            END
```

The routine copies FTAB into the symbol table and returns to the main assembly flow with the accumulator cleared to set the system to pass 1. It also sets the label count to two for the labels just loaded. FTAB can be extended to define other reserved labels by following the same format. Five slots must be used by each label, followed by its two byte definition. This same modification should work for the IMSAI version of Sys8 (which loads at 0 instead of F000). In the version that I have, the above equates would be:

```
AERR        EQU     108EH
NOLA        EQU     1Ø98H
SYMT        EQU     1123H
ASM3        EQU     Ø732H
LMOV        EQU     Ø5BEH
```

Atlantic City Seminar

MICROPROCESSORS:
WHERE THEY CAME FROM AND WHERE THEY ARE GOING – AN ANALYSIS OF ALL PRODUCTS ON THE MARKET TODAY

This seminar will be given by Dr. Adam Osborne at the Personal Computing Conference '77 in Atlantic City, at the Shelburne Hotel, from 9:00 a.m. to 4:00 p.m. The course will be offered twice, once on Friday, August 26th and the second time on Saturday, August 27th.

The cost of the one-day seminar is $30.00. This fee includes lunch.

This seminar will last approximately six hours.

During the first hour the origins of the various types of microprocessors on the market today will be identified. This introduction will cover some of the less well known, but critical circumstances surrounding the companies and products that first appeared. This introductory background is very important; in addition to defining the origins of microprocessors on the market today, it explains why many microprocessor strengths and weaknesses are the result of luck or inexperience on the part of designers and manufacturers.

The next four hours of the seminar are devoted to highlighting the important characteristics of all common 8-bit and 16-bit microprocessors on the market today, together with their support devices. In order to cover this formidable range of material in such a short time, the 8080A microprocessor is used as a frame of reference. Each microprocessor is described in terms of its signal interface, CPU architecture and instruction set. The justification for support devices is given, together with the effectiveness of each implementation. When microprocessors are compared wtih the 8080A, conceptual and functional differences are emphasized. In addition, other microprocessors are compared to each other where they compete, or where they have been directly influenced by a device other than the 8080A. The purpose of this four hour session is to describe the important characteristics of each microprocessor so that the types of application best suited to each microprocessor may be identified, and the types of application where the choice of microprocessor is virtually irrelevant are easily understood.

During the last hour of this seminar the status of the microprocessor industry today and the directions it is taking in the coming year will be defined. This discussion will cover sales volumes of various microprocessors on the market today, plus products planned by various manufacturers in the coming year. The customer base for microprocessors will also be described, since this is one area where supposedly knowledgeable industry observers have been proven completely wrong. As the microprocessor industry matures, we are seeing a customer profile emerge that differs markedly from the customer profile which was anticipated by manufacturers a year ago. This is having important repercussions on the economic fortunes of companies who are participating in the microprocessor and related industries.

TAPE LOAD & SAVE ROUTINES FOR P.T.'s BASIC

Dear Bob,

For those readers who have processor tech. BASIC and would like to be able to store their BASIC programs on tape, I've written the following two subroutines.

Since I do not have a VDM, I used memory locations OEBO thru OEE3 for the TSAV & TLOAD routines. (These locations are normally part of the VDM driver so if you have a VDM, relocate the two routines elsewhere in memory).

Instructions:

1) Load the TSAV & TLOAD routines into memory.
2) At location 07DC, store BO) TSAV
   At location 07DD, store OE)
   At location 07DE, store E3) TLOAD
   At location 07DF, store OE)

   This tells the interpreter that TSAVE is at OEBO and TLOAD is at OE, E3.
3) When you want to record a program in BASIC, just start the cassette tape, then type "TSAV".
4) When you want to play back a BASIC program from cassette tape into memory, just start the tape, and then type "TLOAD".
5) The TSAVE & TLOAD routines are written for an Altair cassette interface addressed at 006 & 007.

Thanks,
Ron Santore

```
TSAV 016,260  041 LXI  HL
              116
              031
     016,263  076 MVI  A
              000
     016,265  323 OUT
              007
     016,267  006 MVI  B
              006
     016,271  315 CALL:  OUTST
              331
              016
     016,274  043 INX  HL
     016,275  005 DCR  B
     016,276  302 JNZ
              271
              016
     016,301  052 LHLD
              116
              031
     016,304  353 XCHG
     016,305  052 LHLD
              120
              031
     016,310  315 CALL:  OUTST
              331
              016
     016,313  173 MOV A,E
     016,314  275 CMP  L
     016,315  302 JNZ
              325
              016
     016,320  172 MOV A,D
     016,321  274 CMP  H
     016,322  312 JZ
              142
              000
     016,325  043 INX  HL
     016,326  303 JMP
              310
              016
OUTST016,331  333 IN
              006
     016,333  007 RLC
     016,334  332 JC:  OUTST
              331
              016
     016,337  176 MOV A,M
     016,340  323 OUT
              007
     016,342  311 RET
```

TSAV – 016,260 → OEBO
TLOAD – 016,343 → OEE3

```
TLOAD 016,343 041 LXI  HL
              116
              031
     016,346  006 MVI  B
              006
     016,350  333 IN
              006
     016,352  017 RRC
     016,353  332 JC
              350
              016
     016,356  333 IN
              007
     016,360  376 CPI
              000
     016,362  302 JNZ
              350
              016
     016,365  333 IN
              006
     016,367  017 RRC
     016,370  332 JC
              365
              016
     016,373  333 IN
              007
     016, 375 167 MOV M,A
     016,376  043 INX  HL
     016,377  005 DCR  B
     017,000  302 JNZ
              365
              016
     017,003  052 LHLD
              116
              031
     017,006  353 XCHG
     017,007  052 LHLD
              120
              031
     017,012  333 IN
              006
```

More

MINI WORD PROCESSING SOFTWARE

News Release Received: 77 Jun 16

The Software Store has released A Mini Word Processing System running on MITS Altair equipment under Disk Extended BASIC for just $150.00. Mini Word Processing is designed to help an operator generate letters, text and mailing labels or envelopes. The System consists of seven programs which are driven by a menu select routine from which any of the seven processing programs can be utilized. Each program interacts with the operator to establish file names and drive numbers. The options are selected by the operator using simple Y or N (Yes or No) responses to the detailed program prompts. After each function is completed the System reloads the menu routine.

A complete User's Manual consisting of 51 pages is provided with the System. The manual includes detailed instructions concerning all operator prompts, system error messages, a number of examples, and programming considerations for custom applications.

The Software Store develops and sells application software systems designed for the first time user of microcomputer. The Software Store markets its products through selected computer stores and direct to end users.

The Software Store is located at 706 Chippewa Square, Marquette, Michigan 49855. Area Code 906 - 228-7622. Contact Terry L Horton. [Dated 77 May 20]

ALTA LOMA (CALIF.) MICRO PROGRAMMING COURSE CHAFFEY COLLEGE

News Release 77 Jun 7

Chaffey Community College at Alta Loma, Calif., has scheduled a fall-quarter course in microcomputer programming that will be offered in two 12-week class sections.

Both classes will be from 7 to 10 p.m., one on Tuesdays starting Sept. 13 and the other on Wednesdays beginning Sept. 14.

The only prerequisite for the three-unit course is that a student be a high school graduate or 18 years of age. There is no tuition fee entailed for state residents.

Further information may be obtained by writing to Donald J. Ketchum, data processing professor, Chaffey Community College, Alta Loma, CA 91701. 987-1737, 822-4484 or 735-0242, Ext. 216.

```
017,014  017 RRC
017,015  332 JC
         012
         017
017,020  333 IN
         007
017,022  167 MOV M,A
017,023  175 MOV A,L
017,024  273 CMP E
017,025  302 JNZ
         035
         017
017,030  174 MOV A,H
017,031  272 CMP D
017,032  312 JZ
         142
         000
017,035  043 INX  HL
017,036  303 JMP
         012
         017
017,041  000 NOP
```

FAIRCHILD & RCA RUMORS

Dear *DDJ*: 9 - Jun - 77

Your Readers might be interested in hearing about a smart new video game that should be on the market in time for Christmas: It's made by Fairchild (makers of the F8 microprocessor), and I think it's going to be called "Channel F." I had an opportunity to play with one at a shopping center demo, and I was intrigued. Its compact cabinet contains four games and produces *color* images. Playing control is by two joystick-like handles which do not give fully variable control, but instead have switches inside which are activated by moving the handle forward, back, left, right, up, down, clockwise, or counter-clockwise; a total of 8 bits, curiously enough. The big plus is that there will be cassettes available which plug into a socket on the cabinet and allow lots of additional games. The unit, without extra cassettes, will be available for a paltry $170!

I have had a chance to study an F8 manual, and it seems pretty obvious that that's what's inside the game. The cassettes would contain type 3851 "program storage units" (ROM + I/O ports, timer, & interrupt logic), and the 3854 DMA chip is a natural for the video interface.

What excites me is the possibility that some clever hacker might build a box that could plug in where the game cassettes go, and have another plug for a conventional computer terminal. If my guesses about the game's innards are correct, this wouldn't be hard at all. Then we could use this computer-disguised-as-a-game as a real computer! At $170 for CPU, RAM (must be at least 2K bytes for the display), joystick input devices, and color video interface, this could be the biggest bargain yet for computer hobbyists – and if you get tired of programming, you can always play pong!

Incidentally, RCA is coming out with a similar product called "Studio II". It presumably contains a COSMAC. It has two calculator keyboards instead of joysticks, and is black & white only, but will be somewhat less expensive.

I haven't bought a computer yet; I'm waiting to hear more about these two items. Maybe someone from Fairchild or RCA could be so generous as to supply some info on how to build an interface of the type described above. Only 205 more days until Christmas! Cogito Ergo Hackum,

Kent A. Multer
#116 Towne Lyne Apts.
Clinton, MA 01510

EDUCATIONAL DATA SYSTEMS ARE A FUNCTION OF HARDWARE, SOFTWARE, AND PEOPLEWARE

Received: 77 Jun 15

The fifteenth annual convention proceedings are now available from the Association for Educational Data Systems. Over eighty original papers presented April 25-29, 1977, in Fort Worth, Texas, are included in the publication. Categories include:

Instructional Support & Curriculum
Managing Computers and Computing
Information Systems & Applications
Certification
Human Values in Educational Data Systems
Social Implications of Computers
Computer Impact on Society

Copies may be ordered by sending $10.00 to:

AEDS Proceedings
1201 16th Street, N.W.
Washington, D.C. 20036

(Postage and handling will be charges on orders not prepaid.)

KIM-I BREAKPOINT ROUTINES Plain & Fancy

Dear Mr. Warren, Sept. 13, 1976

I received all seven issues of your publication at one time, one week ago. Talk about an information explosion! Your *Dr. Dobb's Journal* is really dynamite.

Since I now own a KIM, the article on a Breakpoint Routine for the 6502 in the March, 1976 issue caught my eye. As soon as I realized how useful such a routine would be for debugging, I decided to try it out on my KIM. I am enclosing a listing of my version, as I was able to reduce the size of the routine from 224 to 124 bytes. Since memory space on the basic KIM is not large, some of your readers might be interested in a smaller version. I have also enclosed a slightly larger version which makes the output of the routine more readable by prefacing each register display with an identifying character string.

The listing of the breakpoint routines was generated using a cross assembler I have written. I wrote this assembler program because I got fed up with hand assembling stuff for my KIM and because I have free access to an IBM 360/40. Currently, the assembler recognizes only the 6502 instruction set, but I plan to add the 8080 and 6800 instruction sets and the ability to print machine code in either hex or split octal.

A trick I picked up from *Radio-Electronics* may be of interest to your readers who have both a KIM and a TTY. You can use the paper tape load routine to enter programs into memory using the keyboard instead of the paper tape reader. Refer to Appendix F of the *KIM-1 User Manual* for the paper tape format. Don't worry about calculating the check sum, just enter four zeros. The TTY will brint "ERR KIM", but what you entered will be in memory. The count of bytes can be any value, not just the hex 18 specified in Appendix F. For example, to enter the character string "KIM" into location 100, type the following: L;0301004B494D0000

Now that you've printed all this neat software, I guess I'll have to go buy more memory to put it in.

Very truly yours,
Willi Kushe

P.O. Box 115
Haddonfield, NJ 08033

P.S. Since I wrote this, I have reduced the 6502 lunar lander you published to KIM size. Would you want my version??
[Sure! – Jim]

```
                KIM BREAKPOINT ROUTINE - MINIMUM                WILLI KUSCHE'S HAPPY ASSEMBLER

     LOC  CODE      SOURCE STATEMENT

                    ;
                    ;    SAVE AREAS ARE DEFINED BELOW KIM MPU REG. SAVX AREA IN PAGE 0
                    ;
                    ;
                    *=$E8
     00E8           SAVEP  *=*+1        PROCESSOR STATUS REGISTER
     00E9           SAVES  *=*+1        STACK POINTER
     00EA           SAVEY  *=*+1        INDEX REGISTER Y
     00EB           SAVEX  *=*+1        INDEX REGISTER X
     00EC           SAVEA  *=*+1        ACCUMULATOR
     00ED           PCLO   *=*+1        PROGRAM COUNTER LOW BYTE
     00EE           PCHI   *=*+1        PROGRAM COUNTER HIGH BYTE
                    ;
                    ;    BREAKPOINT ROUTINE BEGINS HERE
                    ;
                    *=$388
     00EF
     0388  85 EC    BRKPT  STA  SAVEA
     038A  86 EB           STX  SAVEX
     038C  84 EA           STY  SAVEY
     038E  68              PLA          PULL STATUS REG. TO A FROM STACK
     038F  85 E8           STA  SAVEP
     0391  68              PLA          PULL PROG. CNTR LOW BYTE TO A FROM STACK
     0392  85 ED           STA  PCLO
     0394  68              PLA          PULL PROG. CNTR HIGH BYTE TO A FROM STACK
     0395  85 EE           STA  PCHI
     0397  BA              TSX
     0398  86 E9           STX  SAVES
     039A  D8              CLD
     039B  20 2F 1E        JSR  CRLF    PRINT CARRIAGE RETURN AND LINE FEED
     039E  A5 ED           LDA  PCLO
     03A0  D0 02           BNE  ALT1    IF PCLO = 0
     03A2  C6 EE           DEC  PCHI      THEN LET PCHI = PCHI - 1
     03A4  C6 ED           DEC  PCLO    LET PCLO = PCLO - 1
     03A6  D0 02    ALT1   BNE  ALT2    IF PCLO = 0
     03A8  C6 EE           DEC  PCHI      THEN LET PCHI = PCHI - 1
     03AA  C6 ED    ALT2   DEC  PCLO    LET PCLO = PCLO - 1
     03AC  A5 EE           LDA  PCHI
     03AE  20 3B 1E        JSR  PRTBYT  PRINT PCHI USING KIM HEX DISPLAY
     03B1  A5 ED           LDA  PCLO
     03B3  20 3B 1E        JSR  PRTBYT  PRINT PCLO USING KIM HEX DISPLAY
     03B6  20 9E 1E        JSR  OUTSP   SPACE 1 CHARACTER USING KIM ROUTINE
     03B9  A5 E8           LDA  SAVEP   LOAD A WITH STATUS REGISTER
                    ;
                    ;    THE FOLLOWING LOOP WILL PRINT A ONE CHARACTER ABBREVIATION
                    ;    FOR EACH FLAG IN THE MPU STATUS REGISTER IF THAT FLAG IS SET
                    ;    IF THE FLAG IS NOT SET (EQUAL TO ZERO), NOTHING WILL BE PRINTED
                    ;
     03BB  A2 08           LDX  #8      INITIALIZE FOR 8 TIMES THROUGH LOOP
     03BD  2A       SHFTLP ROL  A       ROTATE RIGHTMOST FLAG TO CARRY
     03BE  90 08           BCC  DECRX   IF CARRY SET
     03C0  48              PHA            THEN HOLD SHIFTED A ON STACK
     03C1  BD F4 03        LDA  CHTBL-1,X   AND LOAD A WITH CHAR. CORRES. TO FLAG
     03C4  20 A0 1E        JSR  OUTCH   PRINT FLAG ABBREVIATION CHARACTER
     03C7  68              PLA          RESTORE A FOR 'ROL' OF NEXT FLAG
     03C8  CA       DECRX  DEX          DECREMENT LOOP CONTROL

                KIM BREAKPOINT ROUTINE - MINIMUM                WILLI KUSCHE'S HAPPY ASSEMBLER

     LOC  CODE      SOURCE STATEMENT

     03C9  D0 F2           BNE  SHFTLP    IF X NOT = 0, LOOP AGAIN
                    ;
                    ;    THE FOLLOWING LOOP PRINTS THE ACCUMULATOR, X AND Y REGISTERS
                    ;    AND THE STACK POINTER
                    ;
     03CB  A2 04           LDX  #4      INITIALIZE FOR 4 TIMES THROUGH LOOP
     03CD  20 9E 1E  PRLP  JSR  OUTSP   SPACE 1 CHARACTER
     03D0  B5 E8           LDA  SAVES-1,X   LOAD A WITH NEXT REGISTER
     03D2  20 3B 1E        JSR  PRTBYT  PRINT NEXT REGISTER
     03D5  CA              DEX          DECREMENT LOOP CONTROL
     03D6  D0 F5           BNE  PRLP    IF X NOT = 0, LOOP AGAIN
     03D8  20 2F 1E        JSR  CRLF    KIM CR & LF ROUTINE
     03DB  20 9D 1F        JSR  GETBYT  READ VALID OPCODE
     03DE  A2 00           LDX  #0      MAKE NEXT INSTRUCTION PLAIN INDIRECT
     03E0  81 ED           STA  (PCLO,X)  REPLACE 'BRK' INST. WITH OPCODE JUST READ
```

```
                KIM BREAKPOINT ROUTINE - FANCIER FORMAT
                        WILLI KUSCHE'S HAPPY ASSEMBLER


00EA            SAVEY   *=*+1                   INDEX REGISTER Y
00EB            SAVEX   *=*+1                   INDEX REGISTER X
00EC            SAVEA   *=*+1                   ACCUMULATOR
00ED            PCLO    *=*+1                   PROGRAM COUNTER LOW BYTE
00EE            PCHI    *=*+1                   PROGRAM COUNTER HIGH BYTE
                ;
                ;       BREAKPOINT ROUTINE BEGINS HERE
                ;
00EF                    *=$370
0370  85 EC     BRKPT   STA   SAVEA
0372  86 EB             STX   SAVEX
0374  84 EA             STY   SAVEY
0376  68                PLA                     PULL STATUS REG. TO A FROM STACK
0377  85 E8             STA   SAVEP
0379  68                PLA                     PULL PROG. CNTR LOW BYTE TO A FROM STACK
037A  85 ED             STA   PCLO
037C  68                PLA                     PULL PROG. CNTR HIGH BYTE TO A FROM STACK
037D  85 EE             STA   PCHI
037F  BA                TSX
0380  86 E9             STX   SAVES
0382  D8                CLD
0383  20 2F 1E          JSR   CRLF              PRINT CARRIAGE RETURN AND LINE FEED
0386  A5 ED             LDA   PCLO
0388  D0 02             BNE   ALT1              IF PCLO = 0
038A  C6 EE             DEC   PCHI               THEN LET PCHI = PCHI - 1
038C  C6 ED     ALT1    DEC   PCLO              LET PCLO = PCLO - 1
038E  D0 02             BNE   ALT2              IF PCLO = 0
0390  C6 EE             DEC   PCHI               THEN LET PCHI = PCHI - 1
0392  C6 ED     ALT2    DEC   PCLO              LET PCLO = PCLO - 1
0394  A5 EE             LDA   PCHI
0396  20 3B 1E          JSR   PRTBYT            PRINT PCHI USING KIM HEX DISPLAY
0399  A5 ED             LDA   PCLO
039B  20 3B 1E          JSR   PRTBYT            PRINT PCLO USING KIM HEX DISPLAY
039E  20 9E 1E          JSR   OUTSP             SPACE 1 CHARACTER USING KIM ROUTINE
03A1  A5 E8             LDA   SAVEP             LOAD A WITH STATUS REGISTER
                ;
                ;       THE FOLLOWING LOOP WILL PRINT A ONE CHARACTER ABBREVIATION
                ;       FOR EACH FLAG IN THE MPU STATUS REGISTER IF THAT FLAG IS SET
                ;       IF THE FLAG IS NOT SET (EQUAL TO ZERO), A ~~SPACE~~ ZERO WILL BE PRINTED
                ;
03A3  A2 08             LDX   #8                INITIALIZE FOR 8 TIMES THROUGH LOOP
03A5  2A        SHFTLP  ROL   A                 ROTATE RIGHTMOST FLAG TO CARRY
03A6  48                PHA                     HOLD SHIFTED A ON STACK
03A7  90 09             BCC   PRZRO             IF CARRY SET
03A9  BD EF 03          LDA   CHTBL-1,X          THEN LOAD A WITH CHAR. CORRES. TO FLAG
03AC  20 A0 1E          JSR   OUTCH             PRINT FLAG ABBREVIATION CHARACTER
                ;       THE FOLLOWING TWO INSTRUCTIONS ARE A RELOCATABLE
                ;       SUBSTITUTE FOR A 'JMP' INSTRUCTION
 LOC    CODE       SOURCE STATEMENT                    WILLI KUSCHE'S HAPPY ASSEMBLER
03AF  B8                CLV                     CLEAR OVERFLOW STATUS BIT
03B0  50 05             BVC   PULLA             WILL ALWAYS BRANCH
03B2  A9 30     PRZRO   LDA   #'0
03B4  20 A0 1E          JSR   OUTCH             FLAG NOT SET, SO PRINT A ZERO
03B7  68        PULLA   PLA                     RESTORE A FOR 'ROL' OF NEXT FLAG
03B8  CA                DEX                     DECREMENT LOOP CONTROL
03B9  D0 EA             BNE   SHFTLP            IF X NOT = 0, LOOP AGAIN
                ;
                ;       THE FOLLOWING LOOP PRINTS THE ACCUMULATOR, X AND Y REGISTERS
                ;       AND THE STACK POINTER
                ;
03BB  A2 04             LDX   #4                INITIALIZE FOR 4 TIMES THROUGH LOOP
03BD  20 9E 1E  PRLP    JSR   OUTSP             SPACE 1 CHARACTER
03C0  BD F7 03          LDA   REGTBL-1,X
03C3  20 A0 1E          JSR   OUTCH
03C6  A9 3D             LDA   #'=
03C8  20 A0 1E          JSR   OUTCH
03CB  B5 E8             LDA   SAVES-1,X         LOAD A WITH NEXT REGISTER
03CD  20 3B 1E          JSR   PRTBYT            PRINT NEXT REGISTER
03D0  CA                DEX                     DECREMENT LOOP CONTROL
03D1  D0 EA             BNE   PRLP              IF X NOT = 0, LOOP AGAIN
03D3  20 2F 1E          JSR   CRLF              KIM CR & LF ROUTINE
03D6  20 9D 1F          JSR   GETBYT            READ VALID OPCODE
03D9  A2 00             LDX   #0                MAKE NEXT INSTRUCTION PLAIN INDIRECT
03DB  81 ED             STA   (PCLO,X)          REPLACE 'BRK' INST. WITH OPCODE JUST READ
03DD  A6 E9             LDX   SAVES
03E2  A6 E9             LDX   SAVES
03E4  9A                TXS                     RESTORE STACK POINTER
03E5  A5 EE             LDA   PCHI
03E7  48                PHA                     RESTORE PC HIGH BYTE TO STACK
03E8  A5 ED             LDA   PCLO
03EA  48                PHA                     RESTORE PC LOW BYTE TO STACK
03EB  A5 E8             LDA   SAVEP
03ED  48                PHA                     RESTORE MPU STATUS REGISTER TO STACK
03EE  A4 EA             LDY   SAVEY             RESTORE Y
03F0  A6 EB             LDX   SAVEX             RESTORE X
03F2  A5 EC             LDA   SAVEA             RESTORE A
03F4  40                RTI                     RETURN TO PROGRAM
                ;
                ;       TABLE OF CHARACTERS CORRESPONDING TO STATUS REGISTER FLAG NAMES
                ;       STORED IN REVERSE ORDER TO WORK WITH DECREMENTING X REGISTER
                ;       LOOP CONTROL
                ;
03F5  43 5A 49  CHTBL   .BYTE 'CZIDB',$00,'VN'
03F8  44 42 00
03FB  56 4E
                ;
                ;       SET KIM IRQ VECTOR
                ;
03FD                    *=$17FE
17FE  88 03             .WORD BRKPT
                ;
                ;       EQUATES FOR KIM ROUTINES
                ;
                CRLF    =$1E2F
                PRTBYT  =$1E3B
                OUTSP   =$1E9E
                OUTCH   =$1EA0
                GETBYT  =$1F9D
                        .END
```

THE FOLLOWING IS A LOAD OF THE SHORTER VERSION TITLED 'MINIMUM'

```
KIM
0000 00 LQ
;18038885EC86EB84EA6885E86885ED6885EEBA86E9D8202F1EA5ED0F7D
;1803A0D002C6EEC6EDD002C6EFC6EDA5EE203D1EA5ED203D1E209E0E02
;1803B81EA5E8A2082A900848BDF40320A01E68CAD0F2020209E1E0B3A
;1803D0B5E8203B1ECAD0F5202F1E209D1FA20081EDA6E99AA5EE480CED
;1803E8A5ED48A5E848A4EAA6EBA5EC40435A49444200564FF5F4750E10
;0000050005 KIM
KIM
0005 00
SET THE IRQ VECTOR TO THE START ADDRESS OF THE ROUTINE
KIM
0000 00 17FE
17FE 00 88.
17FF 00 03.
1800 A9
LOAD A JUMP TO KIM AT LOCATION 0000
KIM
0000 00 4C.
0001 00 4F.
0002 00 1C.
0003 00
CHANGE THE FIRST BYTE OF THE JUMP INSTRUCTION TO A 'BRK'
KIM
0000 4C 00.
0001 4F
STARTING EXECUTION AT LOCATION 0000 CAUSES ROUTINE OUTPUT
KIM
0000 00 G
0000 NVBIZ 01 08 FF FD
4C
KIM
0000 4C
```

KIM PROGRAM BOOK TO BE AVAILABLE ABOUT AUGUST 15

The First Book of KIM
By Stan Ockers/Eric Rehnke/Jim Butterfield

The First Book of KIM ($9 from ORB, P.O. Box 311, Argonne, IL 60439): a collection of dozens of programs – some useful, some recreational, all tested and documented – to run on the basic Commodore KIM-1 system. Also included is a Beginner's Guide and other information useful to current and prospective KIM owners.

POMONA, CALIFORNIA AREA CLUB

The Pomona Valley Chapter of the SCCS held its first meeting May 19, and discussed ideas for future programs and club activities. A survey of the members showed high interest in programming methods and techniques, programming languages, computer applications, and hardware design and construction. Machines owned or used by those present include four 8080's (Altair or Gnat), four 6592's (Apple or Kim), a 2650 (Processor Applications), and a couple of PDP-8's.

The Programs for July and August meetings (first Thursday of each month) were discussed, and will include hardware demonstrations and talks on microprocessor fundamentals and assembly language programming.

Meetings are held from 7 to 9 PM at the Pomona Public Library, 625 S. Garey Ave., Pomona, CA.

For additional information contact:

Al Sutton
4155 Oak Hollow Rd.
Claremont, CA 91711
(714) 593-6635 evenings

GODBOUT GLORY

Several different people, at both of the last two Homebrew Computer Club meetings, have popped up during "Serial Access" (Announcements-time) and made adamantly favorable comments about service from Godbout Electronics. Truly amazing reports:

They order something, and receive it within days of placing the order. They send back a faulty part (even Godbout is subject to the physiological frailty of electronic components), and receive a repaired or replaced part by return mail.

In our industry, where reports of 6-month back orders (with no cash returned, of course) are reported, regularly if not often, it is a delight to report these unsolicited comments praising Godbout's responsiveness and service to his customers.

Godbout is located at Box 2355, Oakland Airport, CA 94614, (415) 562-0636.

TEE-VEE TYPO

Dave Fylstra mentioned seeing an interesting ad for TV components, recently: "TV yorks available."

TRIPPY COMPANY NAME NOTED

Joining the ranks of such straight-laced companies as Parasitic Engineering and The Barefoot Computer Store, we recently noted a new California company, specializing in repairing sickly equipment: MicroMouse (415-493-7404). Now, with a name like that, you *know* they gotta be good people.

```
03DF  9A           TXS    RESTORE STACK POINTER
03E0  A5 EE        LDA    PCHI
03E2  48           PHA    RESTORE PC HIGH BYTE TO STACK
03E3  A5 ED        LDA    PCLO
03E5  48           PHA    RESTORE PC LOW BYTE TO STACK
03E6  A5 E8        LDA    SAVEP
03E8  48           PHA    RESTORE MPU STATUS REGISTER TO STACK
03E9  A4 EA        LDY    SAVEY  RESTORE Y
03EB  A6 EB        LDX    SAVEX  RESTORE X
03ED  A5 EC        LDA    SAVEA  RESTORE A
03EF  40           RTI    RETURN TO PROGRAM
                   ;
                   ;      TABLE OF CHARACTERS CORRESPONDING TO STATUS REGISTER FLAG NAMES
                   ;      STORED IN REVERSE ORDER TO WORK WITH DECREMENTING X REGISTER
                   ;      LOOP CONTROL
                   ;
03F0  43 5A 49     CHTBL  .BYTE 'CZIDBIVN'
03F3  44 42 31
03F6  56 4E
                   ;
03F8  53 59 58     REGTBL .BYTE 'SYXA'
03FB  41
                   ;
                   ;      SET KIM IRQ VECTOR
                   ;
                          *=$17FE
03FC                      .WORD BRKPT
17FE  70 03        ;      EQUATES FOR KIM ROUTINES
                          SOURCE STATEMENT
                   ;
                   CRLF   =$1E2F
                   PRTBYT =$1E3B
                   OUTSP  =$1E9E
                   OUTCH  =$1EA0
                   GETBYT =$1F9D
                          .END

THE FOLLOWING IS A LOAD OF THE LONGER VERSION

KIM
0000 4C LQ
J1803700585EC86EB84EA6885ER6R85FD6R85EERA86E9D8202F1EA5ED0F65
J1803BD002C6EEC6EDD002C6EFC6EDA5EE203R1EA5ED203R1E209E0DEA
J1803A01EA5E8A2082A4890009BDEF0320A01E8B5005A9302CA01E6809D4
J1803B8CAD0EAA204209E1EBDF70320A01EA93D20A01FB5E8203B1E0B48
J1803D0CAD0EA202F1E209D1FA20081EDA6E99AA5EE48A5ED48A5E80E33
J1803E848A4EAA6EBA5EC40435A4944231564E53595841 4EF5F4750C6D
J0000060006 KIM

KIM
0006 00
SET THE IRQ VECTOR TO THE START ADDRESS OF THE ROUTINE
KIM
0006 00 17FE
17FE 88 70.
17FF 03 03.
1800 A9
LOAD A JUMP TO KIM AT LOCATION 0000
KIM
0000 00 4C.
0001 00 4F.
0002 00 1C.
0003 00
CHANGE THE FIRST BYTE OF THE JUMP INSTRUCTION TO A 'BRK'
KIM
0000 4C .
0001 4F
STARTING EXECUTION AT LOCATION 0000 CAUSES ROUTINE OUTPUT
KIM
0000 00 G
0000 NV1B0IZ0 A=01 X=00 Y=FF S=FD
4C
KIM
0000 4C
```

A 'TURTLE' FOR A VDM

Dear Jim, *77 June 24*

Here is the program I mentioned last night at the Homebrew meeting. Li-Chen Wang gave me so many good ideas for improvements that it must be considered as a version 1.0. I am sort of happy with it and would like to keep a little control over it's dissemination. I plan to copyright TURTLE but will allow your readers to copy it for any non commerical purposes. The copyright will be held by the Educational Micro Computer Company which is my new fictitious name.

Yours,
Marvin R. Winzenread

P.S. CUTS cassette or paper tape copies of TURTLE, HIC, OOPS, CHASE and/or LIFE are available on an individual basis from:

Educational Micro Computer Company
3360 Tonga Lane
Alameda CA 94501

Turtle

© 1977 Educational Micro Computer Company

A real time interpreter for turtle geometry on a direct memory access video display.

The turtle starts at the center of the screen and is under direct control of a keyboard. It obeys the following commands:

Fn	Move Forward n spaces
Bn	Move Backward n spaces
Rn	Turn Right n turns
Ln	Turn Left n turns
Tc	Leave a Trace of character c.

In the above commands n is limited to being a single digit but c can be any keyboard character (including space).

In addition to the turtle commands, the following commands are also executable:

An	Repeat all previous commands n times
CS	Clear the Screen and start over.

As above n is limited to a single digit.

Turtle was written as a subroutine. The carriage return key will return control to the calling routine. To make this a stand alone routine delete statements - and set a stack pointer with sufficient depth to allow some nesting of the A command.

This program was written for a SOL but it should work on any direct memory access 16 x 64 character generating video display. VDM is the starting address of the VDM RAM. LVDM is the high byte of the starting VDM address. VDMP is the control part for the VDM. SINP is the systems keyboard input routine. A zero flag is set if there is no input. SPEED is a data storage location for the video speed. FF is the slowest and 0 is the faster speed. All of the above EQUates need to be changed for other systems.

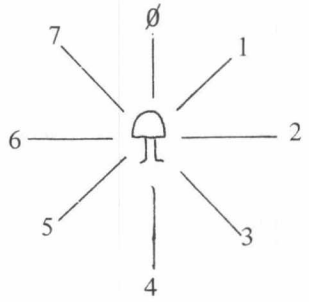
The commands that are typed on the keyboard appear on the screen at the bottom, TXT is a pointer for this text. An additional copy of the command text is saved in an input buffer IBUF so that it will not be destroyed by the turtle's movements. The pointer for this copy is called BUFP. The Again command reads from the input buffer instead of the screen.

Besides the VDM, the program used the following RAM:

TRACE	1 Byte	The trace character
DIR	1 byte	The turtle's direction
PDIR	1 byte	A psuedo direction for it
TXT	2 bytes	Text pointer
BUFP	2 bytes	Buffer pointer
LOC	2 bytes	Location of turtle on the screen
IBUF	64 bytes	Input Buffer

The error routine emits a '?' if a command is not understood. The '?' is removed when an understandable command replaces the erroneous one.

The directions the turtle can face are:




```
                        0100 IN XRA A          Initialize VDM Port
                        0110 ØUT VDMP          Initialize direction as Ø
                        0120 STA DIR           Clear the screen
C906 CD 62 CA           0130 CALL CL           Initialize the turtle's location
C909 21 E0 CD           0140 LXI H,VDM+1FOH
C90C 22 7F CA           0150 SHLD LØC          Put him on the screen
C90F 36 07              0160 MVI M,7           Initialize text pointer as the
C911 21 C0 CF           0170 LXI H,VDM+3COH    lower left hand corner of screen
C914 22 7B CA           0180 SHLD TXT          Initialize input buffer pointer
C917 21 31 CA           0190 LXI H,IBUF        Get a byte and save it
C91A 22 7D CA           0200 SHLD BUFP
C91D CD 63 C9           0500 BACK CALL INBT    Get a second byte and save it
C920 47                 0510 MØV B,A
C921 CD 63 C9           0520 CALL INBT         Execute the instruction
C924 4F                 0530 MØV C,A
C925 CD 2B C9           1003 CALL SRCH         Go back and do it all again
C928 C3 1D C9           1006 JMP BACK          A jump table search routine
C92B 21 35 C9           1020 SRCH LXI H,TBL    that jumps to the routine
C92E 7E                 1030 B1 MØV A,M
C92F B7                 1040 ØRA A             in TBL depending
C930 CA 42 C9           1050 JZ ERR            on the first byte of the
C933 B8                 1060 CMP B             instruction as stored in
C934 23                 1070 INX H             register B.  If first byte is not
C935 C2 3D C9           1080 JNZ B2            a proper command it will branch
C938 5E                 1090 MØV E,M           to the error routine ERR
C939 23                 1100 INX H
C93A 56                 1110 MØV D,M
C93B EB                 1120 XCHG
C93C E9                 1130 PCHL
C93D 23                 1140 B2 INX H
C93E 23                 1145 INX H
C93F C3 2E C9           1150 JMP B1
C942 2A 7B CA           1500 ERR LHLD TXT      Error Routine
C945 36 3F              1510 MVI M,'?'         Print '?' on the screen after
C947 2B                 1520 DCX H             text
C948 2B                 1530 DCX H             Back up text pointer two spaces
C949 22 7B CA           1550 SHLD TXT
```

```
C94C 2A 7D CA
C94F 2B
C950 2B
C951 22 7D CA
C954 E1
C955 C3 1D C9
C958 2A 0A C8
C95B 2C
C95C AF
C95D 2B
C95E BC
C95F C2 5D C9
C962 C9

C963 CD 1F C0
C966 CA 63 C9
C969 E6 7F
C96B FE 0D
C96D C2 72 C9
C970 E1
C971 C9
C972 2A 7B CA
C975 77
C976 23
C977 36 20
C979 22 7B CA
C97C 2A 7D CA
C97F 77
C980 23
C981 22 7D CA
C984 C9
C985 41
C986 37 CA
C988 46
C989 C3 C9
C98B 42
C98C B6 C9
C98E 52
C98F A0 C9
C991 4C
C992 9B C9
C994 54
C995 32 CA
C997 43
C998 73 CA
C99A 00
C99B 06 FF
C99D C3 A2 C9
C9A0 06 01
C9A2 CD 1F CA
C9A5 3A 78 CA
C9A8 30
C9A9 E6 07
C9AB 32 78 CA
C9AE CD 58 C9
C9B1 0D
C9B2 C2 A5 C9
C9B5 C9
C9B6 3A 78 CA
C9B9 C6 04
C9BB E6 07
C9BD 32 79 CA
C9C0 C3 C9 C9
C9C3 3A 78 CA
C9C6 32 79 CA

C9C9 CD 1F CA
C9CC 2A 7F CA
C9CF 46
C9D0 3A 77 CA
C9D3 77
C9D4 CD E9 C9
C9D7 CD 04 CA
C9DA CD F4 C9
C9DD 70
C9DE 22 7F CA
C9E1 CD 58 C9
C9E4 0D
C9E5 C2 CC C9

1560  LHLD BUFP        Same for input buffer pointer
1570  DCX H
1575  DCX H
1580  SHLD BUFP       Reset stack pointer
1590  P0P H
1595  JMP BACK
2300 DELAY LHLD SPEED-1 A variable delay routine
2810  INR L
2320  XRA A           The delay time is inversely
2830 D1 DCX H         proportional to the contents
2340  CMP H           of SPEED.  The contents of
2850  JNZ D1          SPEED-1 is relatively
2860  RET             unimportant

2900 INBT CALL SINP   Input a byte
2910  JZ INBT
2915  ANI 7FH         Mask off parity bit
2920  CPI 0DH         If it is a CR, return to
2930  JNZ I1            calling routine
2933  P0P H
2936  RET
2940 I1 LHLD TXT
2950  M0V M,A         Print the byte on the screen
2960  INX H           Point to next location
2965  MVI M,20H       Erase "?" if it is there
2970  SHLD TXT        Save pointer
2980  LHLD BUFP       Place byte in input buffer
2985  M0V M,A
2990  INX H           Point to next location
2995  SHLD BUFP       Save pointer
2993  RET
3030 TBL DB 'A'       Look up table for acceptable
3035  DW AG           commands.
3040  DB 'F'
3045  DW FW
3050  DB 'B'            A = Do it Again Sam.
3055  DW BA
3060  DB 'R'            F = Move Forward
3065  DW RT
3070  DB 'L'            B = Move Backward
3075  DW LF
3080  DB 'T'            R = Turn Right
3085  DW TR
3090  DB 'C'            L = Turn Left
3095  DW CS
3999  DB 0              T = Leave a Trace
4000 LF MVI B,-1
4010  JMP R1          C = Clear screen
4100 RT MVI B,1       The left and right turn
4110 R1 CALL NBR      routines first convert the
4115 R2 LDA DIR       second byte to binary and
4120  ADD B           use it as a counter.  They
4130  ANI 7           then increment or decrement
4140  STA DIR         the direction by that counter
4300  CALL DELAY      modulo 7.  This code could be
4310  DCR C           shortened but the way it is allows
4320  JNZ R2          for some neat improvements.
4330  RET
5000 BA LDA DIR       Ø = EOF
5010  ADI 4
5020  ANI 7           The backward routine reverses
5030  STA PDIR        the direction by adding 4 mod 7
5040  JMP F1          and this is stored as a psudo-
5100 FW LDA DIR       direction
5110  STA PDIR

5130 F1 CALL NBR      In FW  PDIR = DIR
5135 F2 LHLD L0C      If (C) is an ASCII number convert it
5140  M0V B,M         to binary
5150  LDA TRACE       Get the turtle pointer
5160  M0V M,A         Save the turtle
5200  CALL 0TT        Get the trace
5210  CALL M0V        Put it on the screen where the turtle
5220  CALL TT0          was
5300 F3 M0V M,B       Convert the screen address to
5305  SHLD L0C          co-ordinates
5310  CALL DELAY      Move the pointer in the proper
5320  DCR C           direction
5330  JNZ F2          Convert the co-ordinates to a screen
                      address
```

```
C9E8 C9
C9E9 7D
C9EA E6 3F
C9EC 29
C9ED 29
C9EE 6F
C9EF 7C
C9F0 E6 0F
C9F2 67
C9F3 C9
C9F4 EB
C9F5 7A
C9F6 0F
C9F7 0F
C9F8 6F
C9F9 E6 03
C9FB C6 CC
C9FD 67
C9FE 7D
C9FF E6 C0
CA01 83
CA02 6F
CA03 C9
CA04 3A 79 CA
CA07 25
CA08 B7
CA09 C8
CA0A 2C
CA0B 3D
CA0C C8
CA0D 24
CA0E 3D
CA0F C8
CA10 24
CA11 3D
CA12 C8

CA13 2D
CA14 3D
CA15 C8
CA16 2D
CA17 3D
CA18 C8
CA19 25
CA1A 3D
CA1B C8
CA1C 25
CA1D 3D
CA1E C8
CA1F 79
CA20 FE 31
CA22 DA 2E CA
CA25 FE 3A
CA27 D2 2E CA
CA2A E6 0F
CA2C 4F
CA2D C9
CA2E E1
CA2F C3 42 C9
CA32 79
CA33 32 77 CA
CA36 C9
CA37 CD 1F CA
CA3A 79
CA3B 32 7A CA
CA3E 21 81 CA
CA41 22 7D CA
CA44 2A 7D CA
CA47 7E
CA48 FE 41
CA4A C2 55 CA
CA4D 3A 7A CA
CA50 3D
CA51 C2 3B CA
CA54 C9
CA55 47
CA56 23
CA57 4E
CA58 23
```

MORE

```
5340  RET             Put the turtle back on the screen
5400 0TT M0V A,L      Save the new location
5410  ANI 3FH         Wait awhile
5420  DAD H           Be sure to do it C times.
5430  DAD H           The address of a video screen
5440  M0V L,A         looks like this:
5450  M0V A,H
5460  ANI 0FH           H                    L
5470  M0V H,A         | Memory | line # | position |
5430  RET               6 bits   4 bits    6 bits
5500 TT0 XCHG
5505  M0V A,D         OTT converts this to
5510  RRC             co-ordinates that look like
5515  RRC             this:
5520  M0V L,A           H                  L
5530  ANI 3           | Ø000 | line # | 00 | position |
5540  ADI L'DM
5550  M0V H,A         TTO preforms the opposite
5555  M0V A,L         conversion.
5560  ANI 0C0H
5570  ADD E
5580  M0V L,A         With these two subroutines
5590  RET             as an overhead the actual
5600 M0V LDA PDIR     MOVE routine is greatly
5610  DCR H           simplified since it uses the
5620  0RA A           geometry of a co-ordinate
5630  RZ              system instead of ad-hoc
5640  INR L           addresses.
5650  DCR A
5660  RZ
5670  INR H
5680  DCR A
5690  RZ
5700  INR H
5710  DCR A           It continues to decrement PDIR
5720  RZ              until Ø then exits.

                      Dcr H
                      Ø
                    7   1
Dcr L                       Dcr H
Dcr H                       Inr L
Dcr L  6      Q      2 Inr L
Inr H                       Inr L
Dcr L    5    4    3
                Inr H
5730  DCR L
5740  DCR A
5750  RZ
5760  DCR L
5770  DCR A
5780  RZ
5790  DCR H
5800  DCR A
5810  RZ
5820  DCR H
5830  DCR A
5840  RZ
6000 NBR M0V A,C      Look at the contents of C
6040  CPI '1'         If it is not an ASCII numeral
6050  JC N2           unstack one level and go to
6060  CPI ':'         the ERRor routine.  Else
6070  JNC N2          convert it to binary and return
6030  ANI 0FH
6090  M0V C,A
6095  RET
6100 N2 P0P H
6110  JMP ERR
7000 TR M0V A,C       Set the trace.  Anything (except
7010  STA TRACE       CR) is acceptable
7020  RET
7100 AG CALL NBR      Do it again Sam stores it's
7120  M0V A,C         counter in memory and reads
7130 A1 STA CT        it's instructions from the input
7140  LXI H,IBUF      buffer because the text on the
7150  SHLD BUFP       screen may be destroyed by
7160 A2 LHLD BUFP     the turtles' movements.  No errors
7170  M0V A,M         should exist because they were already
7130  CPI 'A'         corrected earlier.  It continues until
7190  JNZ A3          it encounters the first "A" in the
7200  LDA CT          input buffer.  It uses the same
7210  DCR A           jump table as the main program.
7220  JNZ A1
7230  RET
7235 A3 M0V B,A
7240  INX H
7245  M0V C,M
7247  INX H
```

GOOD LANGUAGE, IMPLEMENTATION, & MAG CARDS

Dear Sir: Received: 77 Jun 6

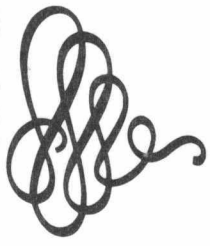
There's been a lot of letter writing about good and bad languages and good and bad implementations. A lot of this war has been fought out already at length. The results seem to be (to me of course) that a good "relatively machine independent language" would look a lot like Simula 67 (the Algol-like structure, the modular protection, and the idea of never do at runtime what you can do at compile time). A dependent language would look like Bliss. The implementation of Simula is more difficult than Basic, but the documentation on it is superb.

In view of this it might be better to debate what we will translate these languages (and others) into. I propose a standard intermediate virtual machine. This could be software simulated with existing hardware for portability, but even better it could be emulated by microprogramming bit slice machines. These machines would, of course, be compatible with the various popular buses, so most people would only lose a CPU card, and even that could be used as a disk controller or something. Comments?

Sincerely,
Peter Norman
Systems Programming Consultant
3 Clinton Street
Cambridge, MA 02139
(617) 864-2674

P.S. Has anyone looked at mag card devices for micros (like the IBM datacell or the old Olivetti P101. It would make it much easier to mail software.

```
                7250  SALD BUFP
                726C  CALL SPCH
                7270  JMP A2
                8000  C1 LXI H,VDM  Clear the screen
                3010  LXI B,420H    4 is a counter 20 is the ASCII space.
                3020  C1 MOV M,C
                3030  INX H
                3040  CMP L
                3050  JNZ C1
                3060  DCR B
                3070  JNZ C1
                3030  RET
                8100  CS POP H      Clear the stack
                8110  JMP IN
                9000  VDM EQU 0CC00H Initialize and start again
                9001  TRACE DS 1
                9002  DIR DS 1
                9003  PDIR DS 1
                9004  CT DS 1
                9005  TXT DS 2
                9006  BUFP DS 2
                9007  L0C DS 2
                9008  IBUF DS 64
                9010  SINP EQU 0C01FH
                9020  SPEED EQU 0C80BH
                9030  VDMP EQU 0FEH
                9040  LVDM EQU 0CCH
CA59 22 7D CA
CA5C CD 2B C9
CA5F C3 44 CA
CA62 21 00 CC
CA65 01 20 04
CA68 71
CA69 23
CA6A B8
CA6B C2 63 CA
CA6E 05
CA6F C2 63 CA
CA72 C9
CA73 E1
CA74 C3 00 C9
CA77
CA77
CA78
CA79
CA7A
CA7B
CA7D
CA7F
CA81
CAC1
CAC1
CAC1
CAC1
CAC1
```



IMPATIENT ROBOTS ADDED TO "CHASE"

Dear Jim: 77-Jul-19

Mr. Sanger's version of "CHASE" in your May 1977 issue is really well done. It's well worth the effort required to key it in. His electric fence post effect is great!

I have come up with the following simple alteration which makes the game operate in real time. The code I added runs a timer while waiting for your move. If you don't make a move before the timer expires, the robots automatically take their turn. This code addition makes the game even more interesting and challenging.

```
051B  CDF405          CALL  INP2        USE TIMED KBD INPUT ROUTINE

05F4  210098  INP2    LXI   H,9800H     SET TIMER DELAY VALUE
05F7  DB00    INDLY   IN    STAT
05F9  E640            ANI   RDA
05FB  C2B001          JNZ   INP+7       JUMP TO GET INPUT & RETURN
05FE  2B              DCX   H           DECREMENT TIMER
05FF  7C              MOV   A,H          AND TEST
0600  B5              ORA   L             FOR ZERO
0601  C8              RZ
0602  C3F705          JMP   INDLY
```

Changing the initial timer value (locations 05F5 and 05F6) results in different rates of robot movement.

A second set of patches increases the effective field size by reducing the horizontal movement to one square per move. The last five changes in this group close in the "holes" in the fence. Leaving them off allows you to maneuver outside of the field, top and bottom only. But beware, the robots will even chase you outside the field and there are some invisible fence posts out there.

1.	0369 was	2B change to	00
2.	036C	23	00
3.	0035	02	01
4.	0042	20	58
5.	004A	1D	00
6.	0057	18	19
7.	005D	20	58

Sincerely,
Andrew A. Recupero
1388 Kinsport Lane
San Jose, CA 95120

FREE DOOR PRIZES FOR CLUB EVENTS

News Release from Radio Shack. Received: 77-5-13

Looking for some door prizes for your next coffee break, hamfest, swapfest, convention or whatever?

For a limited time, Radio Shack will send you five free copies of their new Archer Semiconductor Reference Handbook.

The 128-page Handbook, which regularly sells for $1.95 at Radio Shack stores, lists over 36,000 replacement transistors, diodes, and other interchangeable devices. Every Archer device listed is guaranteed prime and of top-quality with known JEDEC, EIA or manufacturer's numbers.

The Handbook contains a cross-reference listing, sections on the care and handling of transistors, soldering precautions, case styles and dimensions, how to test transistors, and a glossary of words, symbols and abbreviations.

To get your five free copies of the Archer Semiconductor Reference Handbook for your next event, write, on your club or organization's stationery, to Radio Shack, Dept. SRH, 2617 W. Seventh Street, Fort Worth Texas 76107.

An Example of an M6800-Based GPIB* Interface

S. C. Baunach, Tektronix, Inc., Beaverton OR 97077

Editor's Note: The IEEE (Institute of Electrical and Electronics Engineers) has adopted a number of hardware standards over the years. The IEEE 488 Standard describes an interface bus structure appropriate for connection of a large variety of devices to a number of different computer architectures. Unlike the two Altair bus structures and the multitude of Variations on the Theme of the S-100 Bus, the IEEE 488 bus is carefully designed to be a general purpose *structure (*not *designed to match only a single microprocessor chip pin-out), and – even more delightfully – it is* completely *defined in a publication of the IEEE (contact the Standards Office, IEEE, 345 E 47th St, New York, NY 10017).*

This article describes a hardware and software implementation, connecting a M6800 microprocessor to the 488 bus. It should be of particular interest to you hardware hackers, at least partially because the soon-to-be-delivered Commodore "PET" computer uses the 488 bus and *uses the MOS Technology 6502 – a microprocessor that is quite similar to Motorola's 6800.*

We are endebted to Jack Grimes, a project manager for Tektronics and the current Technical Editor for the IEEE Computer Society's Computer *magazine, for contributing this article to* Dr. Dobb's. *He passed it off to us in the Dallas Airport as we were all evacuating from the National Computer Conference, last month.*

INTRODUCTION

This document describes a working implementation of an IEEE 488 peripheral interface. The hardware and firmware described within are used to interface a peripheral mass storage device (the TEKTRONIX 4924 3M cartridge tape unit) to the IEEE 488 general purpose interface bus (GPIB). This implementation is suitable for medium speed applications (about 5K bytes/second) which contain a Motorola M6800 microprocessor. The design goal was to achieve a reasonable trade-off among transfer speed, cost, and firmware complexity.

Before beginning the description of this particular interface, a brief review of the IEEE 488-1975 standard's characteristics will be presented. The IEEE 488-1975 standard is also called the ANSI MC1.1-1975 standard, the proposed IEC bus, the ASCII bus, the HP-IB, the GPIB and other names.

BACKGROUND

The IEEE 488 interface bus provides an internationally standardized communication link among several instruments whether they be measurement apparatus, computers, mass storage devices, or other peripherals limited only by one's imagination. The charter of the IEEE 488 standard is meant to provide the logistics for signal management along the bus connecting the system devices. The standard describes the means for addressing, handling interrupt conditions and data interchange between devices as well as the electrical and mechanical specifications for the bus. As with many other standardization documents, this standard suffers from lack of readability. However, it is complete and thorough. It is the objective of this paper to aid in the understanding of the GPIB via a specific example.

With the advent of inexpensive microprocessors, reasonably sophisticated instruments and systems will become cost-effective and popular. Since microprocessors allow for greater flexibility and device local processing, this writer feels that the GPIB will provide a sufficient means of local data interchange for all but the highest speed applications.

**GPIB – IEEE 488 General Purpose Interface Bus*

Devices on the bus are grouped into three functional categories:

1) listeners – devices which can receive data from the bus,
2) talkers – devices which can send data along the bus and,
3) controllers – devices which provide the management function of assigning who talks to whom and when.

A device may have the capability to assume any, or all, of the above three functions. Additionally, there may be only one active controller and one active talker at any one time along with any number of listeners, (up to 14 as limited by the electrical bus loading). When the controller is giving commands, it is the talker. All other instruments are forced to be listeners, listening to the commands. After the controller has finished giving commands, the bus is free for data interchange among the devices which have been instructed to communicate by the controller. Although the standard does sufficiently describe the logistics of addressing and data transfer, it does not make any attempt to describe the content of data messages which are passed along the bus. Although incompatible messages may seem like a major point, it has proved to be only a minor inconvenience. This stems from two related phenomena:

1) devices are somewhat "intelligent" in that they can perform some local data processing to format data into a palatable state, and
2) most manufacturers are using the ASCII character code in communicating data and status information.

The GPIB is a byte serial - 8 bit parallel communication bus. In addition to the 8 bi-directional data lines there are 3 handshake control lines and 5 bus management lines.

The 3 control lines are used to provide a fully synchronized byte transfer on the 8 data lines (DIO lines). Thus, byte transfers occur under the control of a three wire handshake. The control lines, names and functions, are described below:

DAV – Data Valid. Asserted by the current talker when data is valid.

NRFD – Not Ready For Data. Asserted when the listening devices are not ready to receive data.

NDAC – Not Data Accepted. Asserted when the current data has not been read by all listening devices.

The 5 bus management lines provide the required functions to manage the usage of the bus. In some cases they augment information that appears on the data lines. The names and functions of these management lines are:

ATN – Attention. Asserted by the controller when it is issuing commands on the data lines.

IFC – InterFace Clear. Asserted by the controller to reset all devices to the idle state.

SRQ – Service ReQuest. Asserted by devices to request service. This is an asynchronous interrupt request line.

REN – Remote ENable. Asserted by the controller to activate the bus.

EOI – End Or Identify. Optionally asserted by the talker to end messages. Also defined for use in a high speed parallel poll of interrupting devices.

THE DESIGN

The M6800 microprocessor, and associated firmware, is used to drive the GPIB interface hardware and is also used to control other functions within the peripheral. The interface hardware consists of 1 and 1/2 PIA's (M6820 – peripheral interface adapter), a handful of SSI TTL devices and the required bus transceivers. The interface firmware uses about 750 bytes of ROM out of a total of 6K bytes of ROM contained in the 4924. The interface also requires some of the 768 bytes of R/W memory.

The hardware has three modes of operation:

1) IDLE – This is the unaddressed state where the bus drivers are disabled and the hardware is only "listening" to attention (ATN) and interface clear (IFC – system reset).
2) LISTEN – In this mode, the hardware is driving the NRFD and NDAC handshake lines while listening to the DAV handshake line as well as the associated data and management lines. The management lines used, in this implementation, are ATN and EOI (end or identify) although REN (remote enable) is available for use if necessary.
3) TALK – In this mode, the hardware is driving the DAV handshake line and the appropriate data lines in addition to the EOI bus management line while listening to the NRFD and NDAC handshake lines. The hardware also has the capability of driving SRQ to request service.

The microprocessor is interrupted by the GPIB hardware under the following conditions:

1) IFC is asserted – Asserting IFC informs the firmware that a reset function is required.
2) ATN transistions – Both the assertion and unassertion of ATN causes the M6800 to be interrupted because some intervening action is required.
3) A handshake cycle relevant to the device. The microprocessor is only interrupted to participate in handshake cycles that occur on the bus when the device has been addressed. (See IDLE above.) If in the LISTEN mode, the hardware generates an interrupt when DAV is asserted. If in TALK mode, the hardware generates an interrupt when the slowest current listener indicates its readiness to receive data by allowing NRFD to go high.

TWO HANDSHAKE EXAMPLES

The commands in these examples are given to the system controller which is assumed to be a TEKTRONIX 4051. Furthermore, the 4051 provides the other necessary complementary function, i.e., the talker function in the first sequence and the listener function in the second sequence.

The Listener

In the first example we execute an ASCII transfer of a logical record, i.e., the four character data sequence ABC⟨cr⟩, where ⟨cr⟩ represents the ASCII character RETURN. The command is:

PRINT @1,12: "ABC"

which tells device #1 on the bus (our mass storage device) to receive (and store) the four data characters A, B, C and ⟨cr⟩. The packets of information presented on the bus are shown in Figure 1. A detailed handshake sequence is shown in Figure 2.

The information packets presented in Figure 1 can be summarized as follows:

MLA-1 . . . . The 4051 controller is setting up peripheral device #1 as a listener.

MSA-12 . . . The 4051 controller is further telling device #1 to perform the PRINT command. This is interpreted as My Secondary Address (MSA) #12 or my PRINT command. This byte tells the peripheral device that an ASCII transfer is coming from a talker on the bus.

ABC⟨cr⟩ . . . The controller has now changed roles and is an ordinary talker transmitting data. The data sent is the ASCII data to be stored in the buffer and eventually stored on the magnetic media.

UNL . . . . . The controller has finished the command and is telling the peripheral to get off the bus (stop listening).

The text which follows, along with Figures 2 and 3, describe in detail the hardware/firmware/bus interactions. (Figure 3 is a pull-out schematic in the back of this application note.)

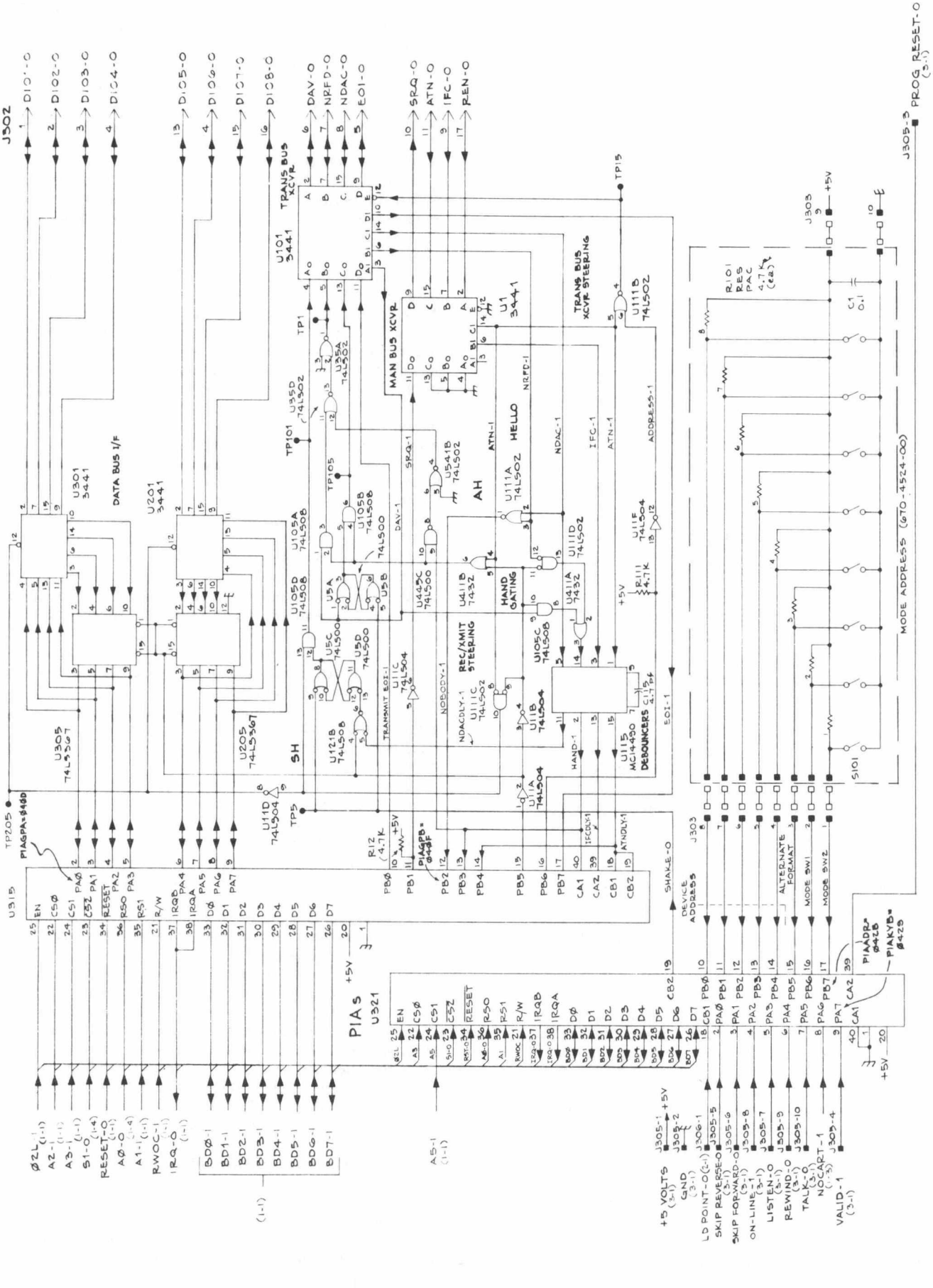


Fig.3.GPIB interface
hardware diagram

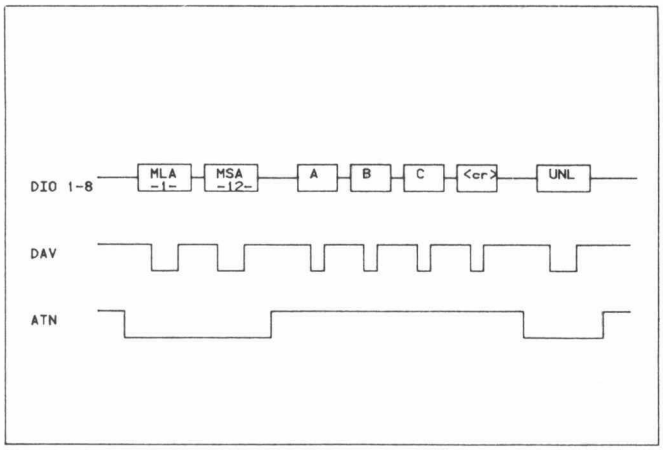


Fig. 1. PRINT @1,12: "ABC"

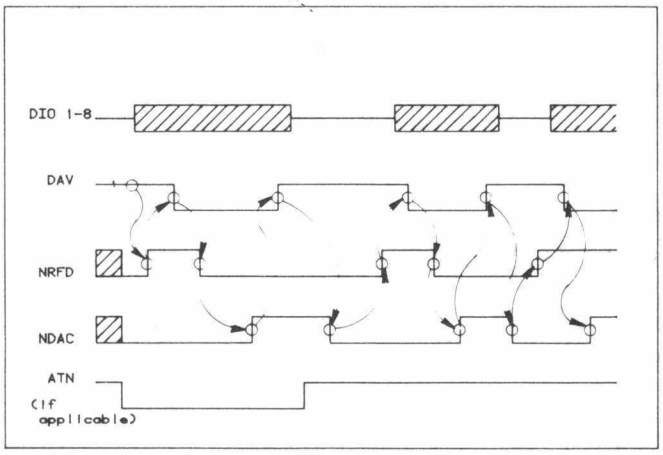


Fig. 2. Detailed Handshake Timing

1) ATN is asserted – This tells all peripherals on the GPIB that the controller is going to send a command (or address) to all peripherals and they must listen. The assertion of ATN causes an interrupt in the M6800 to inform the firmware of the event. ATN also causes the NRFD and NDAC handshake drivers to be placed on the bus in preparation for receiving of the address byte.
2) DAV (data valid) is asserted – This informs the devices connected to the GPIB that the data appearing on the DIO (data) lines is valid and ready for action. Since ATN is also asserted, the peripherals interpret the data byte as a command (the primary address – MLA-1 in Fig. 1). The assertion of DAV causes an interrupt in the M6800 via the HAND line (CA1 - U315 pin 40 in Figure 3) (note: This line is alternately used in TALK mode when NRFD goes high). The information presented on the data lines is interpreted by the peripheral to be its primary listen address. This is determined by the firmware looking at the address switches and comparing them with the address received over the bus. Upon determining that the peripheral has been addressed, the firmware proceeds to enable the addressed bit (PB6 - U315 pin 16 in Figure 3) so that the hardware will be in the proper listen state after the controller is through talking, e.g., ATN goes high. Then the firmware advances to the next state. Meanwhile the hardware has caused NRFD (Not Ready For Data) to go low, indicating that the device is not ready to receive data.
3) A SHAKE pulse is issued by the firmware. This shake pulse causes the R-S flip-flop (U5a - U5b in Figure 3) to enter the reset state which in turn allows NDAC (Not Data ACcept) to go high indicating to the controller that the data has been accepted. Upon seeing NDAC high, the controller proceeds to un-assert DAV (set high) while it prepares the next byte. DAV going high causes the hardware to set NRFD high as well as setting the R-S flip-flop allowing NDAC to go low. The hardware is now ready to receive the next byte.
4) The controller then issues the secondary address 12 which the peripheral accepts as before. The information passed across the GPIB thus far has set device #1 (our device) as a listener (MLA-1), and has told it via the secondary address (MSA-12) to prepare for a PRINT command, where the data following shall be interpreted as data to be stored in the current open file.
5) ATN goes high. Attention going high informs the peripherals on the bus that the controller is finished giving orders and the bus in now free for data interchange. This second transition of attention again causes an interrupt in the M6800 to inform the firmware of the change of state and the hardware acts accordingly. Since this peripheral was instructed to enter the listen state, the hardware stays in the LISTEN mode as determined by the ADDRESSED, and TALK/LISTEN lines which were set appropriately in step 2. If the peripheral had not been addressed, the ADDRESSED line would have been left un-asserted and the drivers would have been off the bus so data interchange among other peripherals could proceed at a rate unrelated to the speed of this device.
6) The talker (also the controller in this example) proceeds to send the four characters of information as data. Once again DAV is asserted, which causes an interrupt in the M6800 and the data to be read. This process is repeated until all appropriate data is sent and received, at which point the controller steps in again and sends the unlisten (UNL) or unaddress command.
7) Upon receiving the unaddress command, the peripheral gets off the bus and begins executing the command requested which in this case was to store four characters of data.

Note that normal data transfers, those without ATN asserted, cause the firmware to store the data received in a buffer rather than being acted upon one byte at a time. This buffering is done for three reasons:

1) Buffering allows data transfers at a maximum rate.
2) It is somewhat simpler to write conversion routines and scanners when a whole buffer is available at a time.
3) This peripheral is a magnetic tape drive and requires the data to be buffered into physical blocks before accessing the actual recording media.

When looking at the firmware listing that follows this discussion, note that ocassionally the hardware can be placed into a suspended state. These suspended states are envoked when a buffer becomes full or when the monitor (overall peripheral control program) is off doing simething more important, like completing the excution of the last command. While in one of these suspended states, the hardware is left "sitting" on the bus refusing to handshake which effectively holds everything. When the monitor has completed the condition that caused the suspended state to be envoked, it can subsequently restart the handshake. One aspect of this buffer management scheme is that it is typical to receive an unaddress command (UNL or UNT) to execute every command (MSA). The unaddress forces the monitor to come out from the idle state and process the current buffer even though it is not full. This generates a clean solution because each command is explicitly delimited.

The Talker

A peripheral talk sequence proceeds in much the same way as the listen sequence perviously discussed and will be illustrated by executing an input data request. The command being performed is:

INPUT @1:A$

This command requests device #1 to send a data stream from the currently open file. The bus information packets are shown in Figure 4. The handshake detail is shown in Figure 2.

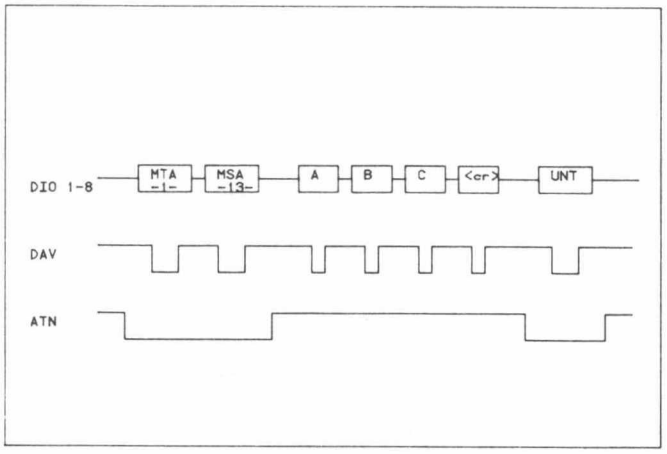


Fig. 4. INPUT @1:A$

A detailed discussion of Figure 4:

1) The first two command bytes, those sent with ATN asserted, are handled by the hardware/firmware in much the same way as the previous example. However, since the first byte instructed the device to enter the talk mode, communicated by the designation MTA-1 (My Talk Address), the firmware asserts the ADDRESSED line and sets the TALK/LISTEN line to the TALK mode. This allows the hardware to enter the TALK state after ATN goes away.
2) When ATN does go away the processor is interrupted, at which point it reverses the data registers so that they can be used for output.
3) As the last (slowest) listening device becomes ready to receive data, the NRFD line goes high. NRFD going high causes an interrupt via the HAND line, this in-turn tells the firmware to put a data byte on the DIO (data) lines and to issue the SHAKE pulse.
4) While in the TALK state, a SHAKE pulse controls a second R-S flip-flop (U5c - U5d in Figure 3). SHAKE puts the flip-flop into the Set state which in turn asserts the DAV line on the bus, indicating that the data is valid. As the slowest listener accepts the data, the NDAC line is driven high. This action Resets the flip-flop, taking away DAV, and allows the listeners to once again enter the RFD (Ready For Data) state.
5) As the RFD state of the listener is reached, an interrupt is once again received on the HAND line and the process repeats itself until the controller is satisfied that enough data has been passed at which point attention (ATN) is reasserted.
6) This example is a special case where the controller and listener are the same device. In general, the TALKER asserts EOI with the last byte of the transfer, thereby signalling the CONTROLLER that the data transfer is complete.
7) ATN is asserted by the CONTROLLER. At this point, the hardware is forced into a listen state by some appropriate gating. The firmware is informed of this action by an interrupt on the attention line and proceeds to interpret further interrupts of the HAND line as being data to be received (note: the data register must be turned around to receive data). Since the message received was an Untalk (UNT) the firmware informs the hardware of this change in state by clearing the ADDRESSED line and setting the TALK/LISTEN bit back to the LISTEN mode.

While in theTALK mode, the bus can be suspended by running out of data in a buffer. This generally calls on the monitor to get another physical buffer from the tape. After getting more data to work with, the handshake is continued.

MISCELLANEOUS COMMENTS

Following are some general implementation comments and/or suggestions.

The EOI (End Or Identify) line is presented as a level and can be received or transmitted by the firmware when required. In this particular implementation the EOI line is used during some operations to indicate the end of information or end of file.

SRQ – This line may be activated by the firmware for requesting service. In the 4924, it is used primarily to indicate that an error condition exists. For example, a write operation was commanded when the media was write protected.

A digital debounce chip (U115 M14490 in Figure 3) is included on the control lines to help control noise problems. However, the delay also slows down the bus and it may be eliminated.

The careful observer will note that the hardware does not really get off the bus when IFC is asserted. The IFC function is executed by the firmware and as such may take longer than 100 uSec. to execute. This is not considered too serious but can be corrected with the addition of a latch and some appropriate gates and a clearing mechanism.

NOBODY – This line is asserted anytime both NRFD and NDAC are sensed high. This condition means that nobody is listening to the bus which is considered an error state if someone is talking. It is appropriate that the talking device either exit from its talk state or wait until some is listening. As an aside, in this implementation it was decided that the device would wait if nobody was on the bus and it was operating in the normal on-line mode. However, if it was operating in the off-line data logging mode (with no controller, it uses a front panel) the device would exit from the current command.

Acknowledgments: C. Roger Muller and Lang Lok were responsible for the development of the hardware used in this interface.

MACROASSEMBLER LISTING

The following is a macroassembler listing which is substantially the same as the interface driver program used in the TEKTRONIX 4924. This listing is believed to be accurate and correct. However, the usual disclaimer that exists for all software, applies here, too.

● ● ● ● ●

GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10
11-Jan-77 22:34:27
TABLE OF CONTENTS

1- 3 INTRODUCTION

```
 4- 1  GLOBAL DECLARATIONS
 5- 1  MONITR  EXAMPLE MONITOR FOR GPIB
       INTERFACE
 6- 1  HWISRV  INTERFACE INTERRUPT HANDLER
 7- 1  UTILS – IFC, UNDRES, SETSRQ
 8- 1  ATNTRU – ATN, HAS BECOME TRUE
 9- 1  ATNFAL – ATN, HAS BECOME FALSE
10- 1  HANDSK – DAV, OR NRFD CONTROL
11- 1  ATNBYT – ACCEPT ATTENTION COMMAND
       BYTE
12- 1  MLASUB, MTASUB – PRIMARY ADDRESS
       REQUEST
13- 1  SPE – SERIAL POLL CONTROL
14- 1  RCVBYT – ENTER A DATA BYTE INTO BUFFER
15- 1  SNDBYT – SEND A DATA BYTE FROM THE
       BUFFER
17- 1  TLKTBL, LSNTBL – SEC, ADR, TABLES
18- 1  IECRST – RESTARTS HANDSHAKES AFTER A
       SUSPENSION
19- 1  PIASET – PIA INITIALIZATION
20- 1  PIATBL'S – PIA DEF.'S AND INIT TABLES

GPIB INTERFACE AN EXAMPLE  RT-11 MMAC  VM02-10
11-Jan-77 22:34:27  PAGE 1

. TITLE   GPIB INTERFACE  AN EXAMPLE
. ENABLE  LC
. SBTTL   INTRODUCTION
```

● ● ● ● ●

FLOW OF CONTROL

The following pages present a verbal picture for the flow of control through the firmware code which is contained in the assembler listing. This picture is to provide an aid in wading (nay drowning) through the actual listing. The major flow of control is depicted but a few nuances have been omitted here for clarity. Routine names will be shown in capital letters with linkages indicated by a series of dots. Four dots indicate calls to routines. Two dots indicate entry points.

● ● ● ● ●

INTERRUPT LEVEL FLOW OF CONTROL

IFC interrupt .... IFC, IFC1
Resets interface PIA to idle state. Sends cleanup command (CLRCMD) to the monitor.

.. RCVBYT .... Checks the status of the system and performs the following appropriate functions.
1) No command active or an error condition – receive the byte and discard it.
2) Command active but no buffer space available – suspends the handshake to wait for new buffer space. Monitor will restart the handshake when the buffer space is free.
3) Command active and data available (the normal case) – accept the data byte and enter it into the buffer.

.. SHAKER .... Issues the SHAKE pulse to the hardware to allow the handshake to complete.

.. SNDBYT .... Check the status of the system and performs the following appropriate functions.
1) No command active or an error condition – transmit an end-of-file byte (256.) with EOI and returns.
2) Command active but buffer empty – suspends handshake and informs monitor that the buffer is empty. When the monitor has new data available it will cause the handshake to proceed.
3) Command active and data available (the normal case) – sends a byte from the buffer.

.. SPE .... Forms the serial poll response byte and transmits that byte.

.. UNDRES .... Inform the monitor that the unaddress has occurred. Tell the hardware to enter the IDLE state .... SHAKER.

ATN true interrupt .... ATNTRU, ATNTR
Sets the attention flag (ATNFG := 255.). Sets the hardware into LISTEN mode and enables HAND interrupts.

ATN false interrupt .... ATNFAL, ATNFLS
Clears the attention flag (ATNFG := 0). Sets the hardware into TALK, LISTEN, or IDLE state as indicated by the hardware mode status byte HWMODE.

HAND interrupt .... HANDSK
Dispatches according to the present state.
1) ATN true (ATNFG set) .... ATNBYT
2) LISTEN mode .... RCVBYT
3) TALK mode and not in SERIAL POLL state .... SNDBYT
4) TALK mode and in SERIAL POLL state .... SPE

MAJOR SUBROUTINES

.. ATNBYT .... Reads current data byte and dispatches accordingly.
1) Any LISTEN address .... MLASUB
2) Any TALK address .... MTASUB
3) Addressed state and any SECONDARY address .... MSACMD
4) SERIAL POLL ENABLE sets the serial poll enabled flag (SPEFG := 0)
6) Other .... SHAKER

.. MLASUB .... Dispatches as follows based on the data byte.
1) If UNLISTEN .... UNDRES
2) If MY LISTEN ADDRESS set LISTEN state .... SHAKER
3) Others .... SHAKER

.. MSACMD .... Takes the secondary address as the command and forms the control pointers to inform the monitor of the command pending. Then .... SHAKER.

.. MTASUB .... Dispatches as follows based on the data byte.
1) If UNTALK .... UNDRES
2) If MY TALK ADDRESS set TALK state .... SHAKER
3) Others .... SHAKER

● ● ● ● ●

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  V02-10   11-Jan-77 22:34:27
GLOBAL DECLARATIONS

                                        .SBTTL  GLOBAL DECLARATIONS
                                ;
                                ;
                                ; Some globals and constants.
                                ;
                                        .GLOBL  HWMODE          ; Hardware mode byte
                                ;       Hwlsn   =       1       ; listen mode
                                ;       Hwtlk   =       128.    ; talk mode
                                ;       Hwlsns  =       4       ; listen suspended
                                ;       Hwtlks  =       8       ; talk suspended
                                ;       Hwmas   =       16      ; pri. adrs. suspended
                                ;       Hwunas  =       2       ; unaddress suspended
                                ;       HwIFCs  =       32      ; IFC suspended
                                ;       Hwsusp  =       hwlsns+hwtlks+hwmas+hwunas+hwIFCs
                                ;       Hwsusn  =       255.-hwsusp
                                        .GLOBL  EOIPTR          ; Pointer to EOI byte
                                        .GLOBL  SPEFG           ; Flag for serial poll enabled
                                        .GLOBL  ATNFG           ; Flag for ATN true
                                        .GLOBL  CNREGA          ; Cont reg and out reg used with idx
                                        .GLOBL  DTREGA          ; Clears control reg and stores
                                                                ; Cont reg and out reg
                                        .GLOBL  CNREGB          ; Same except its attention
                                        .GLOBL  DTREGB
                                        .GLOBL  ALTFLG          ; Set if in alternate format mode
                                        .GLOBL  NEWCMD          ; Pointer to new command block
                                                                ;  --- used by monitor
                                                                ; And is the main communication between
                                                                ; Interupt level and program level

                                        .GLOBL  PIAGPA          ; Iec data PIA
                                        .GLOBL  PIAGPB          ; Iec control line PIA
                                        .GLOBL  PIAADR          ; Iec address switch PIA
                                        .GLOBL  SNDBYT          ; Routine to send a byte from
                                                                ; The current output buffer
                                        .GLOBL  RCVBYT          ; Routine to stuff new char. into
                                                                ; Current input buffer.
                                        .GLOBL  OFFLIN          ; Set by monitor if in offline mode
                                ;
            0010                        ATNLVL  = 16.           ; Atn. dc level input
            0008                        HNDLVL  = 8.            ; Hand dc level input
```

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  VM02-10   11-Jan-77 22:34:27
MONITR  EXAMPLE MONITOR FOR GPIB INTERFACE

                                        .SBTTL  MONITR  EXAMPLE MONITOR FOR GPIB INTERFACE
                                ;
                                ; These few modules are included in this listing to serve
                                ; as an illustration and is a very simplified version of the
                                ; powerup and monitor routines actually used in the TEKTRONIX 4924.
                                ;
                                ; Note: Piaset is a table driven PIA initialization
                                ; routine.  The code and tables for this routine
                                ; is at the end of this listing.
                                ;
                                        .Globl  Hiram           ; Highest point in R/W memory.
                                ;
0000    0F              Start:  Sei                             ; No interrupts!
0001    8E      0000G           Lds     #Hiram                  ; Initialize stack pointer.
0004    CE      03DC'           Ldx     #Iecpia                 ; Set up the GPIB interface.
0007    BD      03A3'           Jsr     Piaset
000A    CE      03C9'           Ldx     #Kbdpia                 ; Set up the address and
                                                                ; front panel switches.
000D    BD      03A3'           Jsr     Piaset
0010    0E                      Cli                             ; Enable interrupts now.
0011    DE        00G   Idle:   Ldx     Newcmd,d                ; New command requested?
0013    26        13            Bne     Dispat                  ; If command go process it.
0015    96        00G           Lda A   Bfratt,d                ; Check if any buffers need service.
0017    26        0F            Bne     Dispat                  ; Service buffers if required.
0019    96        00G   Idle,a: Lda A   Hwmode,d                ; Test if hardware is in a suspended
                                                                ; state and needs to be restarted.
001B    85        00G           Bit A   #Hwsusp
001D    27        03            Beq     1$                      ; Hardware OK.
001F    BD      0375'           Jsr     Iecrst                  ; Go restart hardware.
0022    96        00G   1$:     Lda A   Cmmode,d                ; Check monitor status.
0024    2B        02            Bmi     Dispat                  ; If neg, then need to preform clean-up.
0026    20        E9            Bra     Idle                    ; Else just loop waiting for something.
                                ;
                                ;
                                ; This is the extreamly short form of the
                                ; command dispatcher.
                                ;
                                        .Globl  Functions
                                ;
0028    BD      0000G   Dispat: Jsr     Function                ; Go to appropriate function.
002B    20        EC            Bra     Idle,a                  ; Re-enter Idle loop.
```

```
                                                .SBTTL  HWISRV  INTERFACE INTERRUPT HANDLER

                                        ; This is a "recursive" interrupt service routine
                                        ; After servicing one interrupt request control is passed back to
                                        ; The beginning <hwisrv> UNTil no more interrupts are
                                        ; Pending in the iec bus registers,
                                        ; Cnrega,cnregb save the contents of control regs, A and B
                                        ; Respectively before resetting the interrupt bits in the PIA,
                                        ; The saved control regs, and "recursivness" make it such that
                                        ; No interrupts can be missed.
                                        ;
                                        ;
                                        ; Note: the hardware is configured such that
                                        ; An --- LDX PIAXXX-1 --- will read the control reg, then
                                        ; The data reg. <which clears the hardware interrupt reg.>
                                        ;
                                        ;
                                                .GLOBL  HWISRV
002D    96      00G             HWISRV: LDA A   CNREGA,D        ; Maintain homoginiety
002F    F6      040C                    LDA B   PIAGPA-1        ; Lets close off that window!!!!!
0032    C5      C0                      BIT B   #H0C0,I
0034    27      09                      BEQ     3$              ; Go around if no interrupt there
0036    FE      040C                    LDX     PIAGPA-1        ; Get A side
0039    DF      00G                     STX     CNREGA,D
003B    9A      00G                     ORA A   CNREGA,D
003D    97      00G                     STA A   CNREGA,D
003F    96      00G             3$:     LDA A   CNREGB,D
0041    F6      040E                    LDA B   PIAGPB-1        ; Go around if no interrupt there
0044    C5      C0                      BIT B   #H0C0,I
0046    27      09                      BEQ     4$
0048    FE      040E                    LDX     PIAGPB-1        ; Get B side
004B    DF      00G                     STX     CNREGB,D
004D    9A      00G                     ORA A   CNREGB,D
004F    97      00G                     STA A   CNREGB,D
0051    96      00G             4$:     LDA A   CNREGA,D
0053    85      40                      BIT A   #H40,I          ; Ifc asserted?
0055    26      14                      BNE     IFC             ; Yes-service it
0057    96      00G                     LDA A   CNREGB,D        ; No- ATN asserted?
0059    85      40                      BIT A   #H40,I          ; No-ATN going away?
005B    27      03                      BEQ     2$
005D    7E      00FC'                   JMP     ATNFAL          ; Service ATN, false
0060    4D                      2$:     TST A
0061    2B      72                      BMI     ATNTRU          ; Service ATN true
0063    96      00G                     LDA A   CNREGA,D        ; Handshake interrupt
0065    2B      01                      BMI     1$              ; Yes-service interrupt
0067    3B                              RTI
0068    7E      0142'           1$:     JMP     HANDSK
```

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  VM02-10   11-Jan-77 22:34:27
UTILS--- IFC, UNDRES, SETSRQ

                                                .SBTTL  UTILS--- IFC, UNDRES, SETSRQ
                                        ;
                                        ;
                                        ; Service IFC-reset interface functions,
                                        ; If SRQ was asserted,reset it too,
                                        ; when 4924 is addressed again,SRQ will be asserted
                                        ; Again,(.......or??????)
                                        ;
                                                .GLOBL  IECPIA, PIASET
                                                .GLOBL  HWSUSP
                                                .GLOBL  DTREGB
                                        ;
0068    8D      02              IFC:    BSR     IFC1
006D    20      BE                      BRA     HWISRV
                                        ;
                                        ;
                                                .GLOBL  IFC1, HWIFCS
006F    96      00G             IFC1:   LDA A   CNREGB,D        ; Save ATN status
0071    36                              PSH A
0072    96      00G                     LDA A   CNREGA,D        ; Save other status
0074    36                              PSH A
0075    CE      03DC'                   LDX     IECPIA,I        ; Reset PIA's and edge's
0078    BD      03A3'                   JSR     PIASET
0078    96      00G                     LDA A   HWMODE,D        ; See if have anything suspended
007D    85      00G                     BIT A   HWSUSP,I
007F    27      03                      BEQ     1$
0081    B7      042B                    STA A   PIAADR          ; If suspended clean up hardware
0084    4F                      1$:     CLR A                   ; Reset some flags
0085    97      00G                     STA A   HWMODE,D
0087    97      00G                     STA A   SPEFG,D
0089    32                              PUL A                   ; Restore byte status
008A    84      80                      AND A   #H80,I
008C    9A      00G                     ORA A   CNREGA,D
008E    97      00G                     STA A   CNREGA,D
0090    32                              PUL A                   ; Reset ATN status
0091    84      C0                      AND A   #HC0,I
0093    9A      00G                     ORA A   CNREGB,D
0095    97      00G                     STA A   CNREGB,D
0097    20      09                      BRA     IFCDON
                                        ;
                                        ; This is the program/interrupt level routine to clear the
                                        ; Monitor status,
                                                .GLOBL  IFCCLR
                                        ;
0099    DE      00G             IFCCLR: LDX     NEWCMD,D        ; See if can set up clear command
009B    27      05                      BEQ     IFCDON
009D    86      00G                     LDA A   HWIFCS,I        ; Set IFC suspended
```

```
 48   009F   97   03C          STA A   HSMODE,D
 49   00A1   39                RTS
 50   00A2   96   00G    IFCDON: LDA A   CMMODE,D         ; Already idle?
 51   00A4   27   05           BEQ     IFC..
 52   00A6   CE   0000G        LDX     CLRCMD,I
 53   00A9   DF   03C          STX     NEWCMD,D
 54   00AB   39                IFC..:  RTS
 55                            ;
 56                            ;
 57                            ; Unaddress function
 58                            ; If interface enabled --- disable the hardware and tell
 59                            ; The monitor that an unaddress occured,
 60                            ;
 61                                    .GLOBL  CLRCMD, NEWCMD
 62                                    .GLOBL  UNDRES, HWUNAS
 63   00AC   96   00G    UNDRES: LDA A   HWMODE,D         ; Check if enabled
 64   00AE   7F   0000G          CLR     HWMODE
 65   00B1   85   81             BIT A   #H81,I          ; Lan or tlk?
 66   00B3   27   08             BEQ     2$
 67   00B5   8D   E2             BSR     IFCCLR
 68   00B7   96   00G          LDA A   DTREGB,D
 69   00B9   8A   60             ORA A   #H60,I          ; Ln=>1
 70   00BB   97   00C          STA A   DTREGB,D
 71   00BD   B7   040F         STA A   PIAGPB
 72   00C0   B7   042B   2$:     STA A   PIAADR          ; Toggle shake
 73   00C3   39                RTS
 74                            ;
 75                            ; Assert SRQ on the bus
 76                            ;
 77                                    .GLOBL  SETSRQ
 78                            ;
 79   00C4   96   00G    SETSRQ: LDA A   DTREGB,D
 80   00C6   84   FD             AND A   375,I           ; Assert SRQ
 81   00C8   97   00G          STA A   DTREGB,D
 82   00CA   B7   040F         STA A   PIAGPB
 83   00CD   39                RTS
```

● ● ●

● ● ● ● ●

GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10 11-Jan-77 22:34:27
ATNTRU--- ATN, HAS BECOME TRUE

```
  1                                     .SBTTL  ATNTRU--- ATN, HAS BECOME TRUE
  2                             ;
  3                             ;
  4                             ; Controller has asserted attention
  5                             ; So arm handshake interrupt and
  6                             ; Get ready to listen,
```

```
                                        ;
00D5    8D      03          ATNTRU: BSR     ATNTR
00D7    7E    002D'                 JMP     HWISRV
                                        ;
                                        ;
                                        .GLOBL  ATNTR
                                        ;
00DA    96      00G         ATNTR:  LDA A   CNREGB,D
00DC    84      7F                  AND A   #-1-128,,I
00DE    97      00G                 STA A   CNREGB,D
00E0    86      FF          1$:     LDA A   #H00FF,I        ; Set ATN flg
00L2    97      00G                 STA A   ATNFG,D
00E4    96      00G                 LDA A   CNREGA,D        ; Arm handshake interrupt
00E6    8A      01                  ORA A   1,I
00E8    97      00G                 STA A   CNREGA,D
00EA    B7    040C                  STA A   PIAGPA-1
00ED    96      00G                 LDA A   DTREGB,D
                                                            ; Set up for listen
00EF    8A      20                  ORA A   #H20,I
00F1    84      FF                  AND A   #H00FE,I        ; Clear EOI
00F3    97      00G                 STA A   DTREGB,D
00F5    B7    040F                  STA A   PIAGPB
00F8    BD    01C7'         3$:     JSR     SETLSN
00FB    39                          RTS
```

GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10 11-Jan-77 22:34:27
ATNFAL--- ATN. HAS BECOME FALSE

```
                                        .SBTTL  ATNFAL--- ATN. HAS BECOME FALSE
                                        ;
                                        ;
                                        ; Controller has released attention,
                                        ; So check whether 4924 has
                                        ; Been assigned as talker or listener,
                                        ; If so set up the interface accordingly,
                                        ; If not, disarm the handshake interrupt
                                        ; And remain in idle state.
                                        ;
                                        ;
                                        ;
                                        ;
                                        .GLOBL  TEVEN,TLODD,LEVEN,LSEVEN,TKEVEN,LSODD,CMDODD,TKODD
                                        .GLOBL  MSAOK,ALTFLG,ALTCMD,CMDACT
                                        ;
                                        ;
00FC    8D      03          ATNFAL: BSR     ATNFLS
00FE    7E    002D'                 JMP     HWISRV
```

```
        20                                      ;
        21                                      ;
        22                                              .GLOBL  ATNFLS
        23                                      ;
        24  0101    96      00G         ATNFLS: LDA A   CNREGB,D
        25  0103    84      BF                  AND A   #-1-64,,I       ; 127 or 3f the hard way
        26  0105    97      00G                 STA A   CNREGB,D
        27  0107    96      00G                 LDA A   ATNFG,D         ; Atn flag set?
        28  0109    2A      1B                  BPL     4$
        29  010B    7F    0000G                 CLR     ATNFG           ; Clear flag
        30  010E    96      00G                 LDA A   HNMODE,D        ; Am I in talk?
        31  0110    28      15                  BMI     2$              ; Yes-set up to talk
        32  0112    26      12                  BNE     4$              ; If fale, disarm handshake int
        33  0114    96      00G                 LDA A   CNREGA,D
        34  0116    84      FE                  AND A   #H0FE,I
        35  0118    97      00G                 STA A   CNREGA,D
        36  011A    B7    040C                  STA A   PIAGPA-1
        37  011D    96      00G                 LDA A   DTREGB,D        ; Reset address line
        38  011F    8A      40                  ORA A   100,I
        39  0121    97      00G                 STA A   DTREGB,D
        40  0123    B7    040F                  STA A   PIAGPB
        41  0126    39                  4$:     RTS
        42                                      ;
        43                                      ;
        44  0127    D6      00G         2$:     LDA B   DTREGB,D        ; Go to talk state
        45  0129    C4      1E                  AND B   #H1E,I          ; Make sure EOI is not true
        46  012B    D7      00G                 STA B   DTREGB,D
        47  012D    F7    040F                  STA B   PIAGPB
        48  0130    96      00G                 LDA A   CNREGA,D        ; Address direction reg
        49  0132    84      FB                  AND A   #H0FB,I
        50  0134    B7    040C                  STA A   PIAGPA-1
        51  0137    C6      FF                  LDA B   #H0FF,I         ; Set up all outputs
        52  0139    F7    040D                  STA B   PIAGPA
        53  013C    8A      04                  ORA A   4,I
        54  013E    B7    040C                  STA A   PIAGPA-1
        55  0141    39                          RTS
```

```
         1                                              .SBTTL  HANDSK---- DAV, OR NRFD CONTROL
         2                                      ;
         3                                      ;
         4                                      ; This routine may be servicing DAV or NRFD
         5                                      ; Depending whether the 4924 is in listen or
         6                                      ; Talk state.
         7                                      ;
```

```
                                        .GLOBL  DTREGA
                                ;
0142    96      00G     HANDSK: LDA A   CNREGA,D        ; Get present control reg.
0144    84      7F              AND A   #H7F,I
0146    97      00G             STA A   CNREGA,D
0148    D6      00G             LDA B   DTREGA,D        ; Retrieve data reg.
014A    96      00G             LDA A   ATNFG,D         ; Attntion on?
014C    27      04              BEQ     1$              ; No-branch
014E    8D      18              BSR     ATNBYT          ; Decode command
0150    20      13              BRA     HANDXT
0152    96      00G     1$:     LDA A   HWMODE,D        ; Am I in talk?
0154    2B      05              BMI     TALK            ; Yes-get talk byte
0156    BD    023F'             JSR     RCVBYT          ; Get byte and try to stash it
0159    20      0A              BRA     HANDXT
015B    96      00G     TALK:   LDA A   SPEFG,D         ; Was serial polling enabled?
015D    27      03              BEQ     3$              ; No-unload byte from data buffer
015F    7E    01FA'             JMP     SPE             ; Service serial polling
0162    BD    028C'     3$:     JSR     SNDBYT          ; Try to send byte
0165    7E    002D'     HANDXT: JMP     HWISRV
```

● ● ●

● ● ● ● ●

GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10 11-Jan-77 22:34:27 PAGE 11
ATNBYT--- ACCEPT ATNENTION COMMAND BYTE

```
                                        .SBTTL  ATNBYT--- ACCEPT ATNENTION COMMAND BYTE
                                ; Atnbyt-routine to decode attention command groups
                                ; And dispatches accordingly.
                                        .GLOBL  MSACMD
                                        .GLOBL  ATNBYT
                                ;
0168    D6      00G     ATNBYT: LDA B   DTREGA,D
016A    17                      TBA                     ; Get data byte
016B    C4      1F              AND B   #H1F,I
016D    84      E0              AND A   #H0E0,I
016F    27      19              BEQ     3$              ; Do control functions
0171    81      60              CMP A   #H60,I          ; MSA?
0173    26      09              BNE     1$              ; No-keep checking
0175    96      00G             LDA A   HWMODE,D
0177    85      81              BIT A   #H81,I          ; Check if addressed
0179    27      20              BEQ     SHAKER
017B    7E    02F1'             JMP     MSACMD          ; Do it if addressed
                                ;
017E    81      40      1$:     CMP A   #H40,I          ; MTA?
0180    26      02              BNE     2$              ; No-continue
0182    20      3C              BRA     MTASUB          ; Yes-service it
                                ;
0184    81      20      2$:     CMP A   #H20,I          ; MLA?
0186    26      13              BNE     SHAKER          ; No-continue
```

```
25   0188   20   15          BRA     MLASUB
26                      ;
27   018A   C1   18     3$:  CMP B   #H18,I        ; Serial poll enable?
28   018C   26   06          BNE     4$
29   018E   86   FF          LDA A   377,I
30   0190   97   00G         STA A   SPEFG,D       ; Yes-set spe flag
31   0192   20   07          BRA     SHAKER
32   0194   C1   19     4$:  CMP B   31,I          ; Serial poll disable?
33   0196   26   03          BNE     SHAKER
34   0198   7F   0000G       CLR     SPEFG         ; Yes-clear spe flag
35   019B   B7   042B   SHAKER: STA A PIAADR       ; Toggle shake
36   019E   39                  RTS
37
```

● ● ●

● ● ● ● ●


```
1                           .SBTTL  MLASUB, MTASUB --- PRIMARY ADDRESS REQUEST
2                       ;
3                       ; Look at listen address
4                       ;
5    019F   C1   1F     MLASUB: CMP B   #H1F,I        ; Unlisten command
6    01A1   26   03          BNE     1$            ; No-return
7    01A3   7E   00AC'       JMP     UNDRES
8    01A6   8D   43     1$:  BSR     RDADDR        ; Check address switch
9    01A8   26   F1          BNE     SHAKER        ; If not mine then just shake
10   01AA   DE   00G         LDX     NEWCMD,D      ; Else: if command pending set susp. bit
11   01AC   26   45          BNE     MASUSP
12   01AE   8D   02          BSR     MLASET        ; Else: set listen mode and shake
13   01B0   20   E9          BRA     SHAKER
14                      ;
15                      ;
16                      ; Routine to set hardware into listen mode
17                      ;
18                          .GLOBL  MLASET
19   01B2   96   00G     MLASET: LDA A   DTREGB,D
20   01B4   84   BF          AND A   #H00BF,I      ; Set hardware into addressed mode
21   01B6   97   00G         STA A   DTREGB,D
22   01B8   B7   040F        STA A   PIAGPB
23   01BB   86   01          LDA A   1,I           ; Flag listen mode
24   01BD   97   00G         STA A   HWMODE,D
25   01BF   39                   RTS
26                      ;
27                      ; Look at talk address
28                      ;
29   01C0   C1   1F     MTASUB: CMP B   #H1F,I        ; Is this UNTalk?
30   01C2   26   13          BNE     MTA.1         ; No-check for MTA
31   01C4   BD   00AC'  UNTALK: JSR     UNDRES        ; Untalk: really clean up hardware also
```

```
                                ;
                                ;
                                ; Service serial poll
                                ; Send status byte
                                ;
```

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  VM02-10   11-Jan-77 22:34:27 PAGE 13
SPE------ SERIAL POLL CONTROL
```

```
                                        .SBTTL  SPE------ SERIAL POLL CONTROL
                                ;
                                ;
                                ; Service serial poll
                                ; Send status byte
                                ;
```

```
   32  01C7  96  00G     SETLSN: LDA A   CNREGA,D         ; Address direction reg
   33  01C9  84  FB             AND A   #H0FB,I
   34  01CB  B7  040C           STA A   PIAGPA-1
   35  01CE  7F  040D           CLR     PIAGPA           ; Set up all inputs
   36  01D1  8A  04             ORA A   4,I
   37  01D3  B7  040C           STA A   PIAGPA-1
   38  01D6  39                 RTS
   39  01D7  8D  12     MTA,1:  BSR     RDADDR           ; Read address switches
   40  01D9  26  0A             BNE     1$               ; Not mine so do implied UNT
   41  01DB  DE  00G            LDX     NEWCMD,D         ; Else: if command pending set susp. bit
   42  01DD  26  14             BNE     MASUSP
   43  01DF  86  80             LDA A   #H80,I           ; Else: set talk flag
   44  01E1  97  00G            STA A   HWMODE,D
   45  01E3  20  B6             BRA     SHAKER           ; And shake
   46                       ;
   47  01E5  96  00G     1$:     LDA A   HWMODE,D         ; Am I talker?
   48  01E7  2A  B2             BPL     SHAKER           ; No so no action
   49  01E9  20  D9             BRA     UNTALK           ; Else do UNTalk
   50                       ;
   51                       ;
   52                       ; This is the place where we read the address switch
   53                       ;
   54                       ;
   55  01EB  B6  042B     RDADDR: LDA A   PIAADR           ; Get address
   56  01EE  43                 COM A                    ; Complement to get true data
   57  01EF  84  1F             AND A   37,I             ; Turn off extra bits
   58  01F1  11                 CBA
   59  01F2  39                 RTS
   60                       ;
   61                       ;
   62                       ; Suspend primary address handshake
   63                       ;
   64                       ;
   65                               .GLOBL  HWMAS            ; Suspend primary adrs.
   66  01F3  86  00G      MASUSP: LDA A   HWMAS,I
   67  01F5  9A  00G              ORA A   HWMODE,D
   68  01F7  97  00G              STA A   HWMODE,D
   69  01F9  39                   RTS
```

```
                                                ;
                                                ;
                                                        .GLOBL  ERRCD, CMMODE, ALTFLG, PIAKYB, ONLIN
 01FA    96    00G       SPE:                    LDA A   DTREGB,D
 01FC    36                                      PSH A                   ; Save for later
 01FD    8A    02                                ORA A   2,I             ; Reset SRQ
 01FF    B7    040F                              STA A   PIAGPB
 0202    97    00G                               STA A   DTREGB,D
 0204    8D    10                                BSR     POLSTT          ; Go get status byte
 0206    32                                      PUL A                   ; See if I was one who caused SRQ
 0207    85    02                                BIT A   2,I
 0209    26    02                                BNE     5$
 020B    CA    40                                ORA B   64.,I
 020D    F7    040D      5$:                     STA B   PIAGPA          ; Store byte
 0210    B7    042B                              STA A   PIAADR          ; Like shaker
 0213    7E    w02D'                             JMP     HWISRV
                                                ;
                                                ;
                                                ;       Establish poll status byte
                                                ;
                                                        .GLOBL  EREOF, EREOM
                                                        .GLOBL  POLSTT
 0216    5F                      POLSTT:         CLR B                   ; Form status byte
 0217    96    00G                               LDA A   ERRCD,D
 0219    27    0E                                BEQ     1$
 021B    CA    20                                ORA B   32.,I           ; Set error bit
 021D    81    00G                               CMP A   EREOF,I
 021F    26    02                                BNE     5$
 0221    CA    01                                ORA B   1,I             ; Set EOF bit
 0223    81    00G       5$:                     CMP A   EREOM,I
 0225    26    02                                BNE     1$
 0227    CA    02                                ORA B   2,I             ; Set EOM bit
 0229    96    00G       1$:                     LDA A   CMMODE,D
 022B    27    02                                BEQ     2$
 022D    CA    10                                ORA B   16.,I           ; Set busy bit
 022F    96    00G       2$:                     LDA A   ALTFLG,D
 0231    27    02                                BEQ     3$
 0233    CA    08                                ORA B   8.,I            ; Set alternate mode bit
 0235    B6    0429      3$:                     LDA A   PIAKYB
 0238    85    00C                               BIT A   ONLIN,I
 023A    26    02                                BNE     4$
 023C    CA    04                                ORA B   4,I             ; Set online bit
 023E    39                      4$:             RTS
```

GPIB INTERFACE AN EXAMPLE RT-11 MMAC V02-10 11-Jan-77 22:34:27 PAGE 14
RCVBYT--- ENTER A DATA BYTE INTO BUFFER

```
                                                        .SBTTL  RCVBYT--- ENTER A DATA BYTE INTO BUFFER
                                                ; This is the interrupt level routine to enter a byte into the
                                                ; Present output buffer
```

```
                                                ;
                                                        .GLOBL  A.IN, A.OUT, A.MAX      ; Buffer pointers
                                                ;
                                                        .GLOBL  RCVBYT
                                                        .GLOBL  ERRCD,PIAADR, BFRFUL, BFRSTT, NEWCMD, DTREGA
                                                        .GLOBL  DTREGB,CMDACT
                                                ;
                                                ;
                                                ;
                                                ;
023F    DE      00G             RCVBYT: LDX     NEWCMD,D
0241    26      42                      BNE     NOGOOD
0243    96      00G                     LDA A   ERRCD,D
0245    27      03                      BEQ     4$
0247    7E      02ED'           5$:     JMP     SHAKEY          ; If error set then just shake
024A    96      00G             4$:     LDA A   CMMODE,D        ; Look at monitor status"
024C    27      F9                      BEQ     5$              ; If no command processing then throw out
024E    96      00G                     LDA A   BFRSTT,D        ; Test if buffer available              byte
0250    26      33                      BNE     NOGOOD          ; If buffers full then set susp. bit
0252    DE      00G                     LDX     A.IN,D          ; Get pointer
0254    96      00G                     LDA A   DTREGA,D        ; Get byte
0256    A7      00                      STA A   0,X             ; Save the byte
0258    08                              INX
0259    DF      00G                     STX     A.IN,D
025B    96      00G                     LDA A   ASCFNC,D        ; Need to look at EOI?
025D    27      0B                      BEQ     6$
025F    8D      16                      BSR     DTBSAV          ; Save current hardware status
0261    2A      07                      BPL     6$
0263    DE      00G                     LDX     A.IN,D          ; Mark EOI byte if EOI set
0265    09                              DEX
0266    DF      00G                     STX     EOIPTR,D
0268    20      07                      BRA     3$              ; Also issue end so monitor can act on it
026A    DE      00G             6$:     LDX     A.IN,D
026C    09                              DEX
026D    9C      00G                     CPX     A.MAX,D         ; Buffer full?
026F    26      7C                      BNE     SHAKEY
0271    86      00G             3$:     LDA A   BFRFUL,I        ; Set buffer full
0273    97      00G                     STA A   BFRSTT,D
0275    20      76                      BRA     SHAKEY
                                ;
                                ;
                                ; Routine to carefully look at iec bus control line
                                ; Register without losing interrupts,
                                ;
0277    D6      00G             DTBSAV: LDA B   CNREGB,D        ; Carefully so don't lose interrupts
0279    FE      040E                    LDX     PIAGPB-1
027C    DF      00G                     STX     CNREGB,D
027E    DA      00G                     ORA B   CNREGB,D
0280    D7      00G                     STA B   CNREGB,D
0282    D6      00G                     LDA B   DTREGB,D        ; See if EOI asserted
0284    39                              RTS
```

● ● ●

```
                                        .SBTTL  SNDBYT--- SEND A DATA BYTE FROM THE BUFFER
                                ; This is the interrupt level routine to send a byte from the present
                                ; Data buffer
                                ;
                                        .GLOBL  SNDBYT, BFREMP, PIAGPA
                                        .GLOBL  EREOF, ASCFNC
                                        .GLOBL  CLNUP
0004                                    HELLO   = 4             ; Bit to see if anybody listening on
                                                                ; The bus.
                                                                ; Only used in offline mode.

                                ;
                                ;
                                ; These routines are used to put the hardware in a suspended
                                ; State so that a return from the interrupt level can be effected,
                                ; A suspended state will be envoked if buffers are not available
                                ; Or post commands haven't finished cleaning up.
                                ;
0285  86  00G                   NOGOOD: LDA A   HWLSNS,I        ; Set listen suspended
0287  9A  00G                   EXSUSP: ORA A   HWMODE,D        ; Shared suspension exit
0289  97  00G                           STA A   HWMODE,D
028B  39                                RTS
                                ;
                                ;
                                ;
028C  DE  00G                   SNDBYT: LDX     NEWCMD,D        ; See if can send a byte
028E  26  0C                            BNE     1$              ; Nope so set suspended
0290  96  00G                           LDA A   ERRCD,D         ; If error set send back "^h0ff"
0292  26  49                            BNE     3$
0294  96  00G                           LDA A   BFRSTT,D        ; Check if buffer available
0296  27  04                            BEQ     1$
0298  86  00G                           LDA A   HWTLKS,I        ; Set talk suspended is encounter empty
029A  20  EB                            BRA     EXSUSP                                          buffer.
029C  96  00G                   1$:     LDA A   CMMODE,D        ; If no command then send dummy byte
029E  27  3D                            BEQ     3$
02A0  DE  00G                           LDX     A.OUT,D         ; Get char pointer
02A2  A6  00                            LDA A   0,X             ; Get char
02A4  2A  12                            BPL     4$              ; Don't worry about it if high bit off
02A6  D6  00G                           LDA B   ASCFNC,D        ; Need to check for ASCII logical EOF
02A8  27  0E                            BEQ     4$
02AA  81  FF                            CMP A   255.,I          ; Really ASCII EOF
02AC  26  0A                            BNE     4$
02AE  C6  00G                           LDA B   EREOF,I         ; Set EOF
02B0  D7  00G                           STA B   ERRCD,D
02B2  86  00G                           LDA A   BFREMP,I        ; Set buffer empty to force action by
02B4  97  00G                           STA A   BFRSTT,D                                        dispatcher
02B6  20  25                            BRA     3$
```

```
                                                ;
                                                ; Work on sending the data byte
                                                ;
02B8    D6      00C     4$:     LDA B   OFFLIN,D
02BA    2A      0B              BPL     5$
02BC    8D      B9              BSR     DTBSAV          ; If offline check hello bit
02BE    C5      04              BIT B   HELLO,I
02C0    27      05              BEQ     5$              ; If nobody there then abort command
02C2    86      00G             LDA A   CLNUP,I
02C4    97      00G             STA A   CMMODE,D        ; Set abort
02C6    39                      RTS
02C7    97      00G     5$:     STA A   DTREGA,D        ; Set data byte on bus
02C9    B7    040D              STA A   PIAGPA
02CC    DE      00G             LDX     A.OUT,D         ; At end?
02CE    9C      00G             CPX     A.MAX,D
02D0    26      06              BNE     2$
02D2    86      00G             LDA A   BFREMP,I
02D4    97      00G             STA A   BFRSTT,D        ; If end: set buffer empty flag
02D6    20      15              BRA     SHAKLY
02D8    08              2$:     INX                     ; Update pointer
02D9    DF      00G             STX     A.OUT,D
02DB    20      10              BRA     SHAKEY          ; Finish handshake
                                ;
                                ; Send EOF with EOI
                                ; This is also used for the dummy byte if haven't anything better
                                ; To send.
                                ; This causes the 4051 to abort the current input condition
                                ; Because it looks like an EOF.
                                ;
02DD    86      FF      3$:     LDA A   255.,I
02DF    97      00G             STA A   DTREGA,D
02E1    B7    040D              STA A   PIAGPA
02E4    96      00G             LDA A   DTREGB,D        ; Set EOI
02E6    8A      01              ORA A   1,I
02E8    97      00G             STA A   DTREGB,D
02EA    B7    040F              STA A   PIAGPB
02ED    B7    042B      SHAKEY: STA A   PIAADR          ; Finish handshake
02F0    39                      RTS
                                ;
```

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  VM02-10   11-Jan-77 22:34:27 PAGE 16
SNDBYT--- SEND A DATA BYTE FROM THE BUFFER

                                        .GLOBL  OFFLIN, IOFUNK
                                        .GLOBL  MSACMD
                                        .GLOBL  DTREGA
                                ;
                                ; Look at secondary address to set up new command for the monitor.
```

```
                                        ;
                                        ;
                                        ;
02F1    96      00G     MSACMD: LDA A   OFFLIN,D         ; If offline
02F3    2B      F8              BMI     SHAKEY          ; Then ignore sec, adr,
02F5    CE    031B'             LDX     LSNTBL,I        ; Assume listen address
02F8    96      00G             LDA A   DTREGA,D        ; Get the MSA byte
02FA    84      1F              AND A   #H1F,I          ; Strip off excess
02FC    D6      00G             LDA B   HWMODE,D        ; Is it really talk?
02FE    2A      03              BPL     1$
0300    CE    034C'             LDX     TLKTBL,I        ; Then get talk table
0303    E6      00      1$:     LDA B   0,X             ; Done with table?
0305    2B      E6              BMI     SHAKEY          ; Then illegal sec, addr, so just shake
0307    11                      CBA                     ; Have we found command
0309    27      07              BEQ     MSAOK
030A    08                      INX
030B    08                      INX
030C    08                      INX
030D    08                      INX
030E    08                      INX
030F    20      F2              BRA     1$
0311    A6      03      MSAOK:  LDA A   3,X             ; Get iofunc code
0313    97      00G             STA A   IOFUNK,D
0315    EE      01              LDX     1,X             ; Get command address
0317    DF      00G             STX     NEXCMD,D
0319    20      D2              BRA     SHAKEY
```

```
                                        .SBTTL  TLKTBL, LSNTBL --- SEC. ADR. TABLES
                                        .MACRO  MSA     A,, B., C.
                                        .GLOBL  B.
                                        .BYTE   A.              ; Secondary address
                                        .WORD   B.              ; Command location
                                        .BYTE   C.              ; Iofunc
                                        .ENDM
                                ;
                000A                    .RADIX  10
                                        .GLOBL  LSNTBL
031B                    LSNTBL: MSA     12,PRINT,12     ; Print
031F                            MSA     15,PRINT,15     ; Write
0323                            MSA     1,PRINT,1       ; Save
0327                            MSA     27,FIND,0       ; Find
032B                            MSA     28,MARK,0       ; Mark
032F                            MSA     7,KILL,0        ; Kill
0333                            MSA     29,SECRET,0     ; Secret
0337                            MSA     2,CLOSE,0       ; Close
```

```
        19    033B                              MSA     0,STATIN,0      ; Status write
        20    033F                              MSA     17,PRINT,18     ; Binary save
        21    0343                              MSA     25,PRINT,17     ; Listen
        22    0347                              MSA     16,PRINT,20     ; Binary mem save
        23    0348      FF                      .BYTE   255             ; End of table
        24                                ;
        25                                      .GLOBL  TLKTBL
        26    034C                        TLKTBL: MSA     13,INPUT,13     ; Input
        27    0350                              MSA     14,INPUT,14     ; Read
        28    0354                              MSA     4,INPUT,4       ; Old
        29    0358                              MSA     0,STATOT,0      ; Status read
        30    035C                              MSA     6,TYPE,0        ; Type
        31    0360                              MSA     30,ERROR,0      ; Error
        32    0364                              MSA     9,HEADER,0      ; Header
        33    0368                              MSA     26,INPUT,16     ; Talk
        34    036C                              MSA     17,INPUT,19     ; Binary old
        35    0370                              MSA     24,TERR,0       ; Send magtape read errors
        36    0374      FF                      .BYTE   255             ; End of table
        37              0008                    .RADIX  8
```

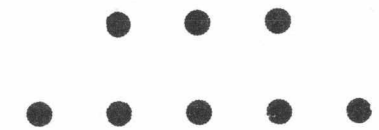
● ● ●

● ● ● ● ●

GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10 11-Jan-77 22:34:27 PAGE 18
IECRST--- RESTARTS HANDSHAKES AFTER A SUSPENSION

```
         1                                      .SBTTL  IECRST--- RESTARTS HANDSHAKES AFTER A SUSPENSION
         2                                      .GLOBL  IECRST
         3                                ;
         4                                ; This routine is called by the monitor to restart the handshake
         5                                ; After the monitor has cleaned up the reason for suspension,
         6                                ; This routine attempts to restart the hardware in an orderly fashion.
         7                                ;
         8                                      .GLOBL  IFCCLR
         9                                      .GLOBL  HWTLKS, HWLSNS, HWMAS, ATNBYT
        10                                      .GLOBL  ALTCMD, CLRCMD
        11                                      .GLOBL  UNDRES, HWSUSP, HWSUSN, HWUNAS, HWIFCS
        12    0375      D6      00G     IECRST: LDA B   HWMODE,D        ; Check for suspended handshakes
        13    0377      C5      00G             BIT B   HWSUSP,I        ; Check if anything suspended
        14    0379      27      1A              BEQ     RSTXIT
        15    037B      07                      TPA                     ; Disallow interrupts
        16    037C      0F                      SEI
        17    037D      36                      PSH A
        18    037E      17                      TBA
        19    037F      84      00C             AND A   HWSUSN,I        ; Clear suspended flags
        20    0381      97      00C             STA A   HWMODE,D
        21    0383      CE    0396'     RSTAGN: LDX     RSTTBL,I        ; Go find suspended function
        22    0386      E5      00      1$:     BIT B   0,X             ; This bit?
        23    0388      26      05              BNE     RSTSRV
        24    038A      08                      INX
        25    038B      08                      INX
```

```
        26   038C    08                      INX
        27   038D    20    F7                BRA     1$
        28                           ;
        29   038F    EE    01          RSTSRV: LDX     1,X             ; Grab appropriate service routine
        30   0391    AD    00                  JSR     0,X             ; Go service it
        31                           ;
        32   0393                      IECXIT:                         ; Clean up and exit
        33   0393    32                      PUL A
        34   0394    06                      TAP
        35   0395    39                RSTXIT: RTS
        36                           ;
        37                           ;
        38                           ;
        39                           ;
        40                                   .MACRO  RT      A.,B.   ; Reset table entry
        41                                   .BYTE   A.
        42                                   .WORD   B.
        43                                   .ENDM
        44                           ;
        45                           ;
        46   0396                    RSTTBL: RT      32.+2.,IFCCLR   ; Both UNListen and IFC
        47   0399                            RT      HWMAS,ATNBYT    ; My primary address
        48   039C                            RT      HWTLKS,SNDBYT   ; Talk suspension
        49   039F                            RT      HWLSNS,RCVBYT   ; Listen suspension
        50   03A2    00                      .BYTE   0
```



GPIB INTERFACE AN EXAMPLE RT-11 MMAC VM02-10 11-Jan-77 22:34:27 PAGE 19
PIASET--- PIA INITIALIZEATION

```
         1                                   .SBTTL  PIASET--- PIA INITIALIZEATION
         2                           ; THIS IS AN INTERPRETER FOR INITIALIZING THE PIA'S
         3                           ;       IT IS CALLED WITH THE INDEX REG, POINTING TO A TABLE
         4                           ;       THE FORMAT OF THE TABLE IS AS FOLLOWS:
         5                           ;       .BYTE  0/1/2/FF   INICATING COMMAND
         6                           ;       .WORD/.BYTE  OPERAND FOR THE COMMAND
         7                           ;
         8                           ;       NOTE: P0 AND P1 ARE DESTROYED
         9                           ;
        10                           ;       SEE THE MACRO DEF.'S ON NEXT PAGE FOR BETTER DISCRIPTION
        11                           ;
        12                                   .GLOBL  P0,P1
        13                                   .GLOBL  PIASET
        14                           ;
        15   03A3    DF    00G       PIASET: STX     P0,D            ; GET START OF TABLE
        16   03A5    DE    00G       PLOOP:  LDX     P0,D            ; GET PRESENT TABLE POINTER
        17   03A7    A6    00                LDA A   0,X             ; GET COMMAND BYTE
        18   03A9    2B    1D                BMI     PIADON          ; DONE
        19   03AB    27    10                BEQ     INDEXU          ; GO UPDATE INDEX REG.
```

```
    20    03AD    E6    01              LDA B   1,X             ; GET CONTROL/DATA BYTE
    21    03AF    08                    INX
    22    03B0    08                    INX
    23    03B1    DF    00G             STX     P0,D            ; SAVE POINTER
    24    03B3    DE    00G             LDX     P1,D            ; GET DESTINATION LOC.
    25    03B5    4A                    DEC A                   ; FORM PROPER STORAGE LOCATION
    26    03B6    27    01              BEQ     1$
    27    03B8    09                    DEX
    28    03B9    E7    00        1$:   STA B   0,X             ; SET IN THE BYTE
    29    03BB    20    EB              BRA     PLOOP
    30    03BD    EE    01        INDEXU: LDX   1,X             ; GET NEW INDEX FOR STORAGE
    31    03BF    DF    00G             STX     P1,D            ; SAVE FOR LATER
    32    03C1    DE    00G             LDX     P0,D            ; ADVANCE TABLE POINTER
    33    03C3    08                    INX
    34    03C4    08                    INX
    35    03C5    08                    INX
    36    03C6    20    DB              BRA     PIASET
    37    03C8    39        PIADON: RTS                         ; HERE'S WERE WE EXIT
```

● ● ●

● ● ● ● ●

```
GPIB INTERFACE AN EXAMPLE       RT-11 MMAC  VM02-10   11-Jan-77 22:34:27 PAGE 20
PIATBL'S--- PIA DEF.'S AND INIT TABLES

     1                                  .SBTTL  PIATBL'S--- PIA DEF.'S AND INIT TABLES
     2                          ; FIRST SOME TABLE GENERATION MACRO'S
     3                          ;
     4                                  .MACRO  NEWX    A.,B.   ; ESTABLISH STORE REG.
     5                                  .GLOBL  A.
     6                          A.=     ^H0'B.                  ; ASSIGN NAME AND VALUE
     7                                  .BYTE   0               ; FORM PROPER TABLE ENTRY
     8                                  .WORD   ^H0'B.
     9                                  .ENDM
    10                          ;
    11                          ;
    12                                  .MACRO  D       A.      ; SET UP INTO DATA REG.
    13                                  .BYTE   1               ; PROPER TABLE ENTRY
    14                                  .BYTE   ^H0'A.
    15                                  .ENDM
    16                          ;
    17                          ;
    18                                  .MACRO  C       A.      ; SET UP INTO CONTROL REG.
    19                                  .BYTE   2
    20                                  .BYTE   ^H0'A.
    21                                  .ENDM
    22                          ;
    23                          ;
    24                                  .MACRO  ENDG    ; END GROUP
    25                                  .BYTE   ^HFF
    26                                  .ENDM
    27                          ;
```

```
28                                      ;
29                                              .MACRO  NEWY    A,      ; SIMILAR TO NEWX
30                                              .BYTE   0
31                                              .GLOBL  A.
32                                              .WORD   A.
33                                              .ENDM
34                                      ;
35                                      ;
36                                      ;       THE TABLES
37                                      ;
38                                              .NLIST  ME
39
40                                              .GLOBL  KBDPIA
41      03C9                            KBDPIA: NEWX    PIAKYB,429
42      03CC                                    C       0
43      03CE                                    D       0
44      03D0                                    C       3C
45      03D2                                    NEWX    PIAADR,42B
46      03D5                                    C       0
47      03D7                                    D       0
48      03D9                                    C       2C
49      03DB                                    ENDG
50                                      ;
51                                              .GLOBL  IECPIA
52      03DC                            IECPIA: NEWX    PIAGPA,40D
53      03DF                                    C       0
54      03E1                                    D       0
55      03E3                                    C       1E
56      03E5                                    NEWY    DTREGA
57      03E8                                    C       1E
58      03EA                                    NEWX    PIAGPB,40F
59      03ED                                    C       4
60      03EF                                    D       62
61      03F1                                    C       0
62      03F3                                    D       63
63      03F5                                    C       F
64      03F7                                    NEWY    DTREGB
65      03FA                                    D       72
66      03FC                                    C       F
67      03FE                                    ENDG
68
69                                              .LIST   ME
70              0001'                           .END
```

ERRATA: TVT-II MOD HAD PIN NUMBERS PERMUTED (AN EXCERCISE IN SPIRITUAL STAMINA?)

Dear Jim, 26 April 1977

As a professional technical writer of 15 years standing, my heart rejoiced at the "beautiful, immaculate, exquisite . . . etc." schematics accompanying Lynn Smith's 64-character/scrolling conversion for the TVT-II. I may say that my heart nearly always rejoices when a new issue of *Dr. Dobb's Journal* appears, but this was a particularly powerful rejoicing, since I am using my system for text processing and needed these modifications badly.

I gloated over the clear instructions, and the beautifully flowing lines, logically so simple. I was even seduced by their charm into believing that the diagrams are accurate.

I first installed the scrolling conversion, which worked like a charm. I noticed that the diagram shows the carry-out (C4, pin 14 of the 7483) grounded, but since the text clearly states that the carry-in (pin 13) is grounded and the carry-out is ignored, that was no problem.

However, when I installed the 64-character conversion, I was nearly driven up the wall for three evenings by the fact that the scroll occurred at the 33rd character and the line then wrapped around to catch up on itself. At the end of the second evening, having carefully checked all the connections against Lynn's diagram (and had two other people do the same) and found no errors, I went ot bed muttering curses. On the third evening, I reluctantly plunged into the monstrous bowl of spaghetti that SWTP supplies instead of a logic diagram, and began tracing down all the counters–character, cursor, and line. I tried some changes based on what I thought was happening, but the line feed remained obstinately at the 33rd character, sometimes joined by the home. Finally, I traced out and redrew, with all pin numbers, the section associated with the line counter. And behold! I found that Lynn Smith's connections to IC40, pins 3 and 8, are reversed. When I reversed the two wires going to the new counter stage, everything worked like a charm. A marked up diagram showing the proper connections is enclosed.

After doing all that work, I thought of several letters to you, all starting with the English poet Charles Williams' acid remark "Hell is inaccurate," but before going to sleep I picked up my bedside book and happened on the following:

****

> The Mulla Nasrudin went into the market place and started to address the throng.
>
> "O people, do you want knowledge without difficulty, truth without falsehood, attainment without effort?"
>
> Very soon a large crowd had gathered, everyone shouting "Yes, yes!"
>
> "Excellent," said the Mulla. "I only wanted to know. You may rely upon me to tell you all about it if I ever find any such thing."*

****

After that, what can I do except thank you and Lynn Smith most gratefully for five evenings on which I was kept out of mischief, crowned by a most beautifully scrolling 64-character display, plus a considerably better understanding of how the TVT-II works?

With all best wishes,

Christopher Terry
324 East 35th St.
New York, NY 10016

*From "The Pleasantries of the Incredible Mulla Nasrudin" by Idries Shah (Dutton Paperback D306, New York, 1971)

P.S. This is, without exception the easiest conversion to install of any I have seen–most others, including one which needs only one extra chip, require chips to be removed in order to cut traces under them. Ugh!

CENTRONICS ANNOUNCES COMPACT MICROPRINTER

Hudson, New Hampshire (Monday, June 13, 1977) – Centronics Data Computer Corp. (NYSE) introduced a compact, high speed, low cost microprinter – the Micro-1 – at the 1977 National Computer Conference.

The microprinter produces copy on aluminum coated paper by discharging an electric arc to penetrate the coating, which is less than one micron thick. Toners and ribbons are not required.

The printed characters, unlike those resulting from thermal printing, are impervious to light, temperature, and humidity. In addition, the finished printed page may be reproduced on most office copy machines.

Aimed at the home, hobby, and microprocessor markets, the Centronics microprinter has a higher print speed (240 characters per second) and lower selling price ($595) than competitive models. Initial deliveries are slated to occur during the last calendar quarter of 1977.

ARTEC ELECTRONICS INTRODUCES 32K BYTE EXPANDABLE MEMORY BOARD

News Release Received: 77 Jun 15

SAN CARLOS, Calif. – Artec Electronics, Inc., has introduced a 32K byte static memory board in modular form designed primarily for microcomputer hobbyists but also applicable for small business uses.

Designated 32K-100, the board is fully compatible with the S-100 data bus and speed-compatible with Zilog Z80-based systems. The basic board with all support circuits, power regulator, 8K bytes of memory and assembly manual sells for $290.

Additional 8K bytes of static RAM cost $255 each, and a full 32K memory board sells for $1,055. The 32K-100 is available for immediate delivery June 15, 1977, with a five per cent discount on single-unit orders received before July 1, 1977. Volume discounts are allowed to electronic distributors and computer store outlets.

The 32K-100 requires plus-8 single voltage only with a power usage of 3 amps; access time of the DMA-compatible board is 250 nanoseconds. DMA compatibility allows users to access memory directly without going throught the central processing unit (CPU) on the microprocessor board.

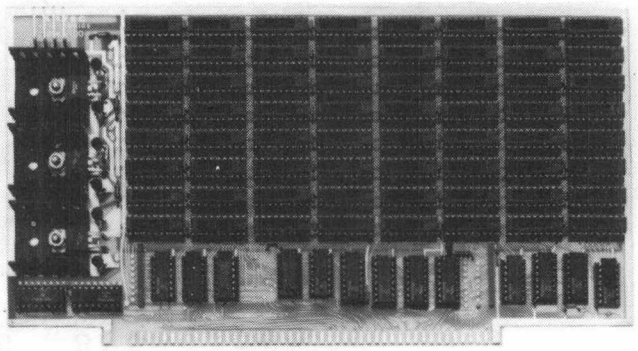
The 32K board, fully buffered on all address and data lines, also features battery back-up (allowing operation at reduced power consumption), and a "bank select" provision so users can select the blocks, or banks, of memory they want to address.

According to Artec President Robert Jones, the advantage of having all 32K of static RAM on one card is that "usually one has to put this much memory on two cards, sometimes up to eight cards. This way, you add 24K memory to the original 8K without any additional investment in more hardware."

While the predominant use of the 32K-100 is expected to be home computer users, Mr. Jones sees the board also having applications for the "small businessman for pricing, bookkeeping and process control."

Artec Electronics, formed about five years ago, produces a complete line of printed circuit boards for industrial and home computer use.

For more information, contact Robert Jones at ARTEC, 605 Old County Road, San Carlos, CA 94070, (415) 592-2740; or Paul Plansky at TYCER-FULTZ, Palo Alto, (415) 328-6300.



WIRE WRAPPING CATALOGETTE

News Release *Received: 77-Jul-18*

This new 12 page flyer features many new products that O.K. MACHINE AND TOOL CORPORATION has developed for the serious electronics hobbyist and the lab technician.

The products include revolutionary manual and battery

powered wire-wrapping tools as well as pre-cut and stripped wire, wire rolls, DIP sockets and wire wrapping "kits" specially designed for the individual enthusiast.

Leader of the line is the BW-630 battery powered wire-wrapping tool. The tool uses standard "C" size NiCad batteries, weighs only 11 ounces, features a high impact housing and comes already equipped with a built-in bit and sleeve for wrapping 30 AWG wire on standard DIP socket terminals (.025" square). At only $34.95 (less batteries), the tool boasts a positive indexing mechanism and a built-in "backforce" device to protect against over-wrapping, features normally found only in the finest industrial tools.

Most popular among the hobbyist on a tight budget, or to complement the BW-630 for occasional use, model WSU-30 is an extraordinary hand tool that performs 3 complete functions. Lightweight (less than ½ ounce) and compact (4½ inches long), the tool *wraps* 30 AWG wire on .025 square posts. In addition, WSU-30 also *unwraps*. Finally, it *strips* 30 AWG wire nick-free by means of a unique built-in stripper. Costing only $5.95 the WSU-30 performs the same functions as 3 industrial tools at less than 1/5 the price. Its rugged all metal construction ensures a lifetime of outstanding service.

The tool line is complemented by a complete selection of pre-cut and stripped wire, plus a remarkable new Wire Dispenser that features unique cutting and stripping capability. After the required length of wire is pulled from the dispenser a built-in plunger cuts length free from roll, while a gentle pull through the stripping blade removes the insulation without nicking the wire. Costing only $3.45, the dispenser includes 50 ft. roll of 30 AWG industrial quality kynar insulated OFHC silver plated solid copper wire.

All items available from stock at local electronics outlets or directly from O.K. MACHINE AND TOOL CORPORATION, 3455 Conner Street, Bronx, New York 10475. For more information just ask us for the HOBBY WRAP FLYER.

LOWER CASE MOD FOR SWTPC's CT-1024

Dear Mr. Warren: May 6, 1977

I'm forwarding another mod to the SWTPC CT-1024 for your consideration. This one came about because of my interest in word processing and a need for lower case on my TVT-II. It turns out that adding one of the readily available *lower case* version 2513 ROMs is a simple matter requiring only a few hours work and about $15.00 for parts. Hopefully the enclosed circuit may save someone a little time and effort in adding lower case to their TVT-II.

This is strictly "no frills." The circuitry does not include below line shifting or even Greek characters. All it does do is add the ASCII lower case alphabet. Since all characters are formed from a 5 x 7 dot pattern they are not as "smooth" or well defined as the newer terminals get with a 7 x 9 pattern, still they are clearly legible and, considering the price tag on a new terminal, look just great.

Only three IC are used for the mod. Two of them, the 2102-1 seventh bit memory and *lower case* 2513 character generator, are installed piggy-back as shown on the drawing. I used a small PC board for the 74SL00 and the cursor gating circuitry.

Because two character generator ROMs are now being used the cursor feed to the chip enable had to be changed slightly by running it through a set of diode gates so that both chips could be controlled from the one cursor line. To do this, the lead feeding IC-22, pin 11, must be opened. This may be done either by breaking the foil on the main board near the IC or by carefully cutting pin 11 free from the board and bending it out.

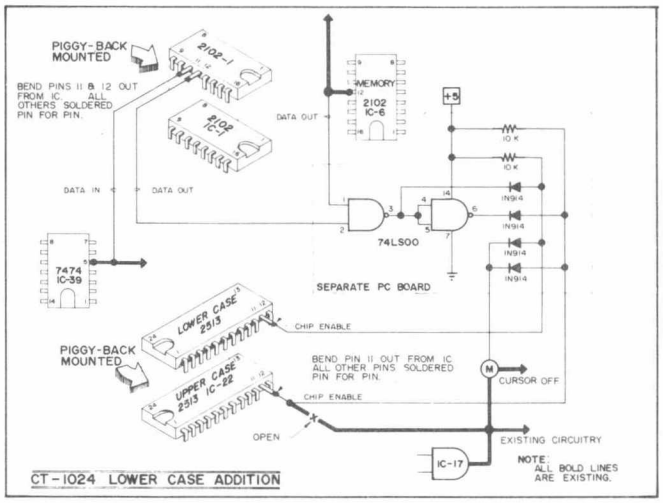
The cursor feed is connected to the diode gates by the wire running from point "m" on the main board.

Installation should present few if any problems. Of course, a moderate amount of care should be taken while soldering the two piggy-back IC's, overheating them could be catastrophic.

Sincerely,

R. Lynn Smith

PSC #1 Box 7327
APO San Francisco, CA 96286



dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 17 **August 1977** **Volume 2, Issue 7**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

CONSUMER NOTES

SCCS Group Buy – *Complaint & Response* 9
About Those ASCII TTY's That You *Cannot* Trade For . . ., David Jon Fylstra 10
Some Dire Warnings of a Modem, Jef Raskin 11
Pennywhistle Plays a Better Tune, Jef Raskin 13

SOFTWARE

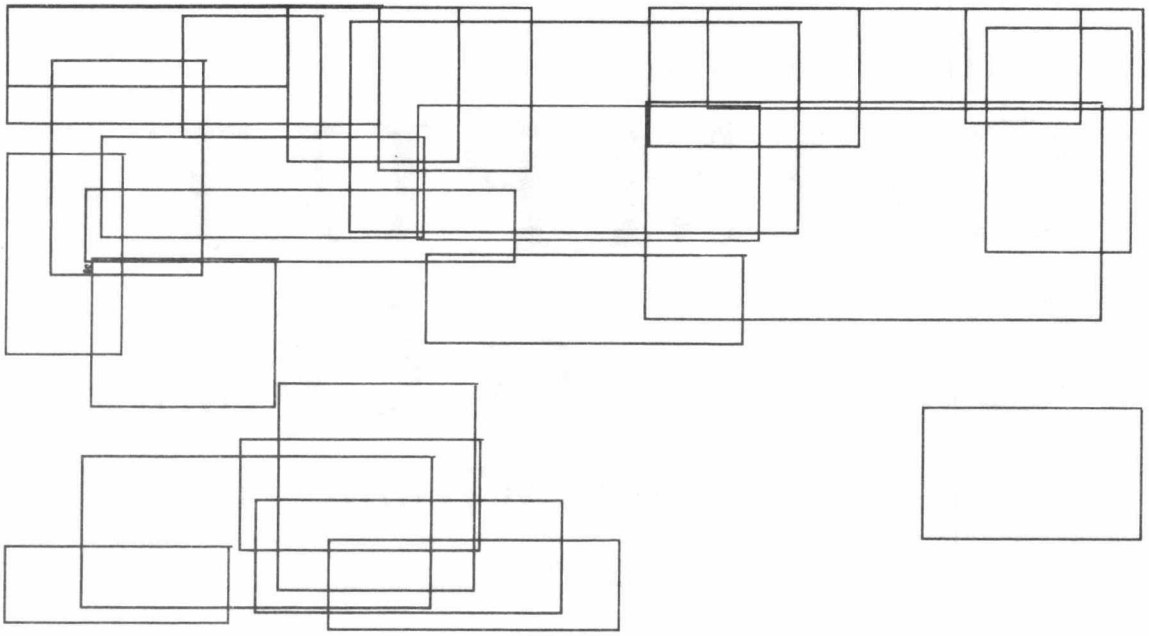
Towards a *Good* μ-C Language, Joshua P. Gordon 15
On the Effects of Filling Cavities Within the Fillings of Cavities Within . . ., Steve Witham 16
The APL Core Project, Texas A&M APL Committee 18
Decoding 650X Opcodes, H.T. Gordon 20
A High-Speed Memory Test Program for the 6502, Jim F. Butterfield 23
Mods to 8080 PILOT, Steven Spence 24
Z-80 Patches for MITS Basic, Martin D. Gray 25
Real-Time Modification for Sanger's Video CHASE, David J. Morgan 26
One-for-One Patch to P.T. 5K Basic, Morris G. Miller 26
An 8080 Tracer, John Walker 30
Friend Finds Factorial Flaw, Jef Raskin 31

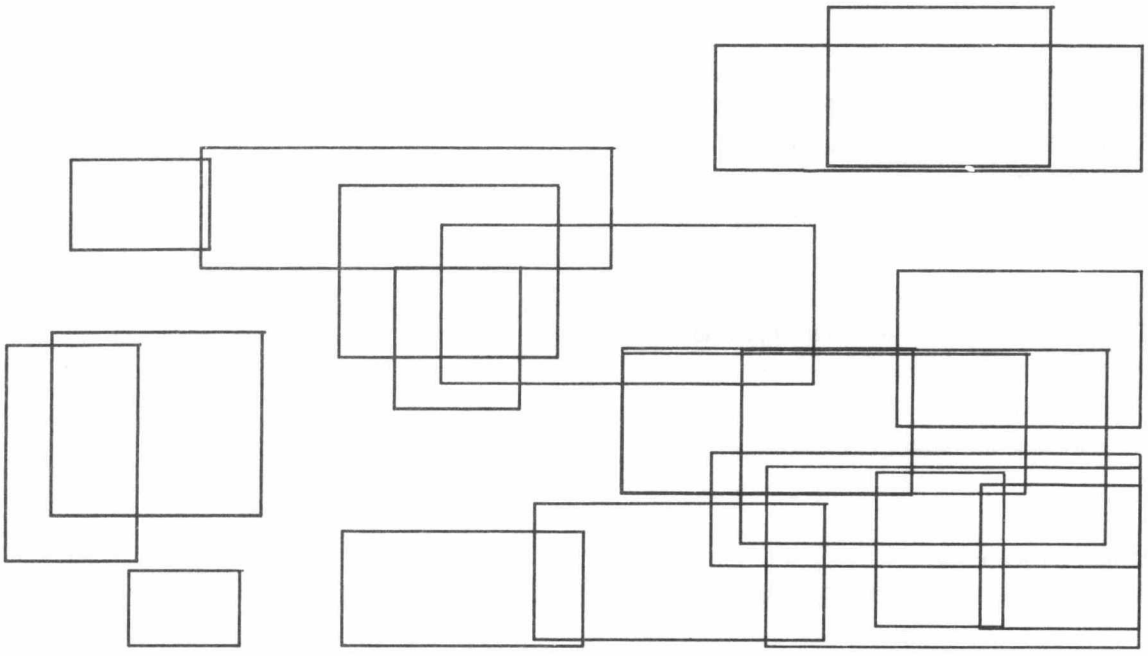
HARDWARE

VDM-1 Modifications, Lichen Wang 33
Dissecting the SWTPC Control Interface, Phillip Schuman 35

MISC.

PCF is Center for Personal Communications Law, Kenneth S. Widelitz 37
- - - plus lot's more - - -





© David Dameron 1979

SCCS GROUP BUY

Dear Jim, 1977 August 3

I'm writing this letter to you to let you know that all is not wine and roses in the computer hobbyist field. There are snakes in our Eden. I am one of approximately forty SCCS Group Purchase victims. By sharing my tragic experience with you, perhaps I might be able to prevent a similar situation from re-occurring to others.

In April and May of 1976, I placed orders for over $1000 worth of IMSAI products. My money was given by the SCCS to Mr. Thomas Hudson, President of Microprocessor Marketing. At the time, Microprocessor Marketing was the second authorized IMSAI dealer in the Southern California area. In addition, this dealer was carrying full page advertisements in three national publications:

(1) *Interface Age* (Sep 1976)
(2) *73 Magazine* (Aug, Sep, Oct, & Dec 1976)
(3) *SCCS Interface* (June & July 1976)

In these advertisements, Mr. Thomas Hudson portrayed himself as a friend of computer hobbyists everywhere and proclaimed that IMSAI "delivered in weeks, not months". It's been *well over* a year now and I have not received the shared-memory facility, three 4K static memory boards, and twenty-two 100-pin edge connectors that I had ordered and paid for. To the best of my knowledge, the SCCS does not know the present whereabouts of Mr. Thomas Hudson and has not been able to recover the money owed to us for undelivered products.

I personally feel that the SCCS as a whole has been relatively insensitive to the plight of the Group Purchase victims. No civil or criminal proceedings were ever filed by the SCCS against Microprocessor Marketing. As far as I'm concerned, the SCCS has let Mr. Thomas Hudson go scott free with my money and that of fellow members. At a minimum, the SCCS could have informed its 8,000 members through its magazine that Microprocessor Marketing had not yet delivered on old orders. If the club won't inform its membership in such a clear-cut case as this that there exists an unreliable dealer so that *other members won't get hurt,* I don't think it's much of a club. However, to be fair, I should add that Larry Press, the editor of *SCCS Interface,* did try to get this information into the magazine but was over-ruled by a majority of the Board of Directors.

Nevertheless, the Board of Directors did authorize that a 20% refund be made to Group Purchase victims in December 1976 and that a monthly 10% payment be made the following eight months. By August of 1977, the Group Purchase victims supposedly would have received back all of their money. I should add that no interest payments were requested by the vicitms nor was there any offerred by the SCCS. In my opinion, it is no coincidence that the term of the present officers and the Board of Directors expires at the end of August 1977. This was probably the minimum the SCCS could do to avoid getting sued by its own members. However, the SCCS still owes me a little over $300.00. No monthly payments were made in June or July. It remains to be seen if I'll receive a monthly payment for August. I will let you know if I do.

I spoke to IMSAI about Microprocessor Marketing. Microprocessor Marketing is no longer an authorized IMSAI distributor. I asked IMSAI if they felt they had any responsibility to the Group Purchase victims. Their answer (and I believe justifiably, though I don't like it) was no. The lesson I learned here is that authorized IMSAI dealers are *not* really corporate extensions of IMSAI. Therefore, it is the customer's own fault and tough luck if he deals with a bad dealer. I therefore suggest to every consumer that unless he can buy the product he wants off-the-shelf from his computer store dealer that he consider *not* spending his hard-earned money or that he order the product he wants C.O.D. *directly* from the manufacturer.

In spite of Microprocessor Marketing, I've managed to continue to develop my interest in personal computing and explore the fascinating world of microprocessors. It would have been a lot easier if this tragic and traumatic experience had not occurred. Try not to let it happen to you.

Sincerely,
Kenneth Young
3311 W. 3rd Street
Apt. 1-319
Los Angeles CA 90020
(213) 383-9666

LARRY PRESS REPLIES FOR SCCS

Dear Jim, *Received: 77 Aug 6*

Ken Young showed me a copy of his letter to you and I have a few comments. His basic point, that SCCS lost a lot of money in dealing with microprocessor marketing is true, but I must take exception to a few others.

A number of visits were made to the police and the district attorney's office regarding the possibility of criminal action. This was pursued to a fairly high level in the district attorney's office, but they would not prosecute. SCCS also spent $150 with attorneys investigating the advisability of civil action, but on their advice it was given to a collection agency instead. The basic reason is that it is costly to get a judgement, which is quite possibly worthless, while collection agencies only get a cut of what is actually collected. The collection agency has been unable to collect anything to date and it is doubtful that they will.

I cannot speak for anyone but myself, but as I stated in the January, 1977 issue of *Interface,* I consider repayment of the group purchase victims to be SCCS' primary short run, moral obligation. I am not a board member, but have made my views known to them on a number of occasions. To date 75% has been refunded (5% in August) and it is a struggle.

Finally, in my opinion, it is pure coincidence that the terms of office of the directors and officers expire in August.

Lest this seem like pure apology and defensiveness, let me repeat that SCCS was burned badly. We were burned because, caught up in the enthusiasm for a new movement, we behaved in an incredibly naive, unbusinesslike manner. I might also add that SCCS owes me a good deal more than it owes any of the group purchase victims, but I went into it with my rose colored eyes wide open.

THE *CORRECT* PHONE FOR THOSE CHEAPO 2311 DISC DRIVES

Last issue, we listed an incorrect phone number for Bob Perez, a source of super-cheap 2311-like disc drives. His *correct* phones are (415) 961-8941 or 961-8946.

LOW-COST WEST COAST TOUR TO NYC's PERSONAL COMPUTING EXPO

Leda Alpert a travel agent with Travel Coordinators, 9060 Santa Monica Blvd., Los Angeles 90069, (213) 278-7933, is planning a tour package – including low-cost air fare – for people attending *Byte's* Personal Computing Expo in New York City's Coliseum, Oct. 28-30th, from the west coast.

We Blew It –

ABOUT THOSE ASCII TTY'S THAT YOU *CANNOT* TRADE FOR. . . .

& NOTES ON TTY'S FOR THE DEAF

David Jon Fylstra
Box 10051
Stanford CA 94305

77 September 6

In the last issue, *Dr. Dobb's Journal* printed an article entitled "For Free: Replace your BAUDOT TTY with an ASCII TTY" (June/July 1977, p. 14). which claimed, among other things, that I was "assisting in arranging....trades" of BAUDOT teletypes in exchange for ASCII teletypes. This unfortunate article was published without my knowledge and without my permission, and has caused me considerable inconvenience. The worst of the article was that *Dr. Dobb's* saw fit to publish my business phone number as the only means of contacting me, resulting in a heavy load of calls for my secretary and me to deal with, and causing many people to waste their money making unnecessary long-distance calls during business hours. I thought about forwarding these calls to Editor Warren's home phone to let HIM deal with it, but I am too gentle a soul for that.

While it's true that I helped to bring together a few hobbyists at the Bay Area's Homebrew Computer Club with a local organization which serves the deaf for the purpose of exchanging ASR 33 machines for Model 19's, the supply of Model 33's was quickly exhausted; and we by no means have the personpower, let alone the machines, to cope with the volume of inquiries that this article produced. *Dr. Dobb's* somehow failed to notice that I have not made any announcements about this exchange at Homebrew meetings for more than two months.

Another source of confusion is the term DEAFNET. Deaf people would not have any idea what this refers to; it is simply a term that a few of us in the Bay Area have coined to refer to the large numbers of people in the U.S. who use teletypes to communicate with each other. It is not a network in any conventional sense: there are no store-and-forward nodes, there is no inter-node or end-to-end protocol, it does not packetize its information. It simply consists of the many thousands of people who initiate individual phone calls with a BAUDOT teletype and a special modem, in order to transact business, to get emergency help, or just to chat with a friend.

The real credit for this system used by the deaf, their friends, and service organizations, belongs to such pioneers as Robert H. Weitbrecht and H. Latham Breunig. The original acoustic coupler and terminal unit were invented by Weitbrecht in the spring of 1964. American Telegraph and Telephone Company agreed to release surplus TTY's to the Alexander Graham Bell Association for the Deaf in 1968, and Dr. Breunig and Jess M. Smith founded a not-for-profit corporation, Teletypes for the Deaf, Inc. (TDI), in June of that year, to organize the acquisition and distribution of the machines among the deaf. The 1976-77 International Directory published by TDI lists 6,000 member stations, but the actual number of TTY's in use among the deaf is surely in excess of that number. (Much of this historical information comes from this Directory, which is available at a reasonable cost from Teletypes for the Deaf, Inc., P.O. Box 28332, Washington, D.C., 20005.)

Readers interested in the technical details of this system would do well to peruse Weitbrecht's original patent, "Frequency-Shift Teletype," U.S. Patent number 3,507,997, patented April 21, 1970. This one-channel system uses a Mark frequency of 1400 Hz. and a Space frequency of 1800 Hz.; and when the transmitter is idling between bursts of characters, the line is silent. If the receiving party wants to interrupt the sender, he or she transmits the same frequency pairs, thus temporarily garbaging up the transmission. Most of you will quickly recognize that this system is completely incompatible with the Bell 103A standard, the method used by the approximately one-half million terminals around the world today. The reason for this is simple: Weitbrecht developed his modem before the Bell 103A standard came into common practice and was standardized in February of 1967. Add to this the fact that the deaf use Model 19 speeds and the BAUDOT character set instead of the universally-accepted 10-or30 character per second speed and ASCII character set, and you have a system that serves the needs of the deaf marvellously, but that excludes them from those one-half million potential sources of contact, and that ties them to that beast of a vanishing species, the BAUDOT teletype.

Enter the computer hobbyist. His desire for low-cost hardcopy has exacerbated the problem by generating a market for every last BAUDOT machine left (at least that's how it seems). Many deaf people are growing resentful of computer hobbyists and hams who contribute to the scarcity of BAUDOT machines. If you can possibly spare your machine, I urge you to find a deaf person in your area who can benefit from this means of access to the telephone system. Your machine might become that deaf person's first experience in independently communicating beyond the range of his or her eyesight.

Aside from parting with your ancient, noisy teletype, what else can you do to help the deaf? Here is an area where many engineers seem to rush in to create technical solutions for problems with which they have no direct experience. My personal opinion is that the best thing you can possibly do is to befriend a deaf person and learn to communicate with him on his terms -- you will discover the frustrations of fingerspelling, the tremendously high "bandwidth" of sign language, and the innummerable sources of confusion in visual communication; when you speak, you might notice yourself using your face more expressively, pantomiming with your hands. By the time you are successfully able to communicate to your friend the reason why this teletype cannot transmit to your ASR 33, you will be well on your way to thinking up solutions to these problems. My feeling is that through these kinds of experiences, we learn to accept the individual differences among people instead of pressuring them to bend in.

My other comment is that you should locate one of the organizations that serve the deaf in your area and offer your help. They have worked with the deaf for many years and may have some remarkable insights for you, and you might have just the technical skills that they need. Among other things, there is a tremendous need for help in maintaining those teletypes and acoustic couplers. One hobbyist group that called me from Greensboro, North Caroline, told me that they are setting up a dial-in news service for the deaf using an IMSAI 8080 system. There is a need for low-cost, reliable ASCII-to-BAUDOT-to-ASCII converters. Other ideas will surely occur to you.

In conclusion, in spite of the inconvenience caused me by last month's article, I must remark that it was a pleasure to talk with many of the people who called me from places as diverse as Spokane, Minneapolis, Chicago, Kingman (AZ), Clatskonie (OR), San Diego, and Houston. If this sample of peole was at all representative of the average computer hobbyist, then there are many fine people among hobbyists with a genuine interest in their fellow man.

SOUR NOTES ON A PENNYWHISTLE

SOME DIRE WARNINGS ON A MODEM

Jef Raskin *Received: 77 March*

The Pennywhistle 103, an originate modem by M & R enterprises in Sunnyvale, CA (which is a sunny place), is a good idea at a good price, with a few problems. For electronic whizzkids with a barrowful of test gear these problems will be easily surmounted, leaving them with a fine product. On the other hand a few details make this kit not a project for a beginner. There are some surprisingly inept features to the Pennywhistle, and it requires some rather fancy test equipment for alignment. But it is good value for the dollar, and is alone in its class for the moment.

The neologism "modem" stands for "modulator-demodulator". The need for modems arose when somebody wanted to send a digital signal to New Medford (for example) and noticed that there was a telephone line going to New Medford. But the telephone lines only carried audio. So this somebody made a device that turned a "0" into a 1070 Hz tone, and a "1" into a 1270 Hz tone. This was a modulator. At the other end (probably in New Medford) a device was built that translated a 1070 Hz tone into a "0" and 1270 Hz tone into a "1". This was a demodulator. Very much like a cassette interface.

The party in New Medford, as it happened, wanted to send digital signals back to our original inventor. At the same time. No problem at all, just use a different pair of frequencies to answer. As a person can hear and separate different tones, so can an electronic device. Mother Bell chose 2025 Hz and 2225 Hz for zeros and ones going in the other direction.

A modem that sends the low pair and listens for the high pair of tones is called an "originate" modem. A modem that sends the high pair and listens to the low pair is named an "answer" modem. By rather arbitrary convention, in time-sharing computer systems, the terminals are equipped with originate modems, the computer with answer modems. What the computer hobbyist needs is an originate *and* answer modem so that she or he can play either role with another hobbyist. A number of companies make them. They cost about $300 (and up, of course). The Pennywhistle is half of what the hobbyist needs. In any case, if you have a Pennywhistle and a friend has a Pennywhistle you just can't hook up your computers and have them talk back and forth. Fooey. You can go one direction at a time, of course, but you have to switch *manually*. I was dissappointed; but their advertising is accurate - - it is clearly sold (in the ad that I have before me) as an "originate modem". And it will transmit (but *not* recieve) in answer mode. What their advertising does not say is that test equipment not found around the average personal computer household is required, and that the instructions are misleading and incomplete.

I regret to have to write a negative review, since the circuitry is quite clever and quite tolerant of almost anything. In fact, it is excellent. The designer must have been told, when a child, to be discrete. He (the amiable and talented Lee Felsenstein) is discrete, for the printed circuit board is almost totally free of integrated circuits, but is full of resistors and capacitors - - all of the highest quality. No junk parts here. There are seven transistors, about two dozen diodes, over half a gross of resistors, and more than a score of carefully selected capacitors. The list of resistors in the manual has a typo (rare in this manual). On page 8 read R31 instead of R37 for one of the three 680-ohm resistors.

So far, for a negative review, things sound pretty good. The impression of quality persists throughout the instruction manual. It is a neat booklet, printed on good paper, typeset, with the schematics and drawings clear and professionally done. Pretty. And the writing starts out very well. The detailed explanation of *how* the circuit functions is clear and complete. But the assembly instructions start out fair-to-middlin' and quickly deteriorate to plain rotten. By the end, they are disastrous. Did someone get tired as they were writing?

At the beginning the instructions are idiot level explicit, as they should be: "Install the resistors. Bend the leads at right angles to the board . . ." While this step takes a while (remember how many resistors there are), nobody should have difficulty with it. Except for one thing. The stuffing diagram, with little notations like R1, R2 and so on, is on page 11; but the list of which resistors have which values is on page 8. You can tear apart the nice little booklet, or try to find each resistor's value from the schematic facing page 11. Either way, you are going to spend a lot more time than would be necessary. If the stuffing diagram had component values the builder would be saved a lot of effort, and would tend to make fewer errors.

All this criticism and this is only the first step of the instructions. The second step has the builder install the diodes. You are carefully told which end is the cathode, and where to put it (politely). The author of the instructions still assumes that you are of subnormal intelligence. This is true for me when I am building a kit, so I am not insulted. In the third step the manual is still excellent. I quote some good advice from it: "The capacitors which are mounted in the upright position have their lead holes intentionally spaced a bit wider than the lead spacing. This insures that a small amount of exposed wire is left above the board for test probes and clips." This is evidence that an experienced builder wrote the manual.

But step 4 - - install the IC sockets - - leads to an abrupt drop in quality. The question is, which way to install them? Where does pin 1 go? The instructions say nothing. The stuffing diagram says nothing. At no point on the PC board is there a little dot or square land to signify pin 1. *Nada. Niente.* I figured it out by comparing the lands with the schematic with the pinouts for the chips. It turns out, looking at the stuffing diagram, that all pin 1's go to the upper left. But the builder should have been told. The idiots are suddenly left to fend for themselves. Incidentally, you are never told to insert the IC's. The neophyte, following the instructions literally, will be left with a handful of parts. So in your manual add STEP 6a: Install the IC's. Maybe *not* 6a. Usually wait until soldering, etc., is complete.

Little problems crop up: the primary leads, supposedly black, were in fact white. The experienced kit builder is used to this. This particular problem – wrong color leads on a transformer (perhaps from a different supplier than that in the original) – is a very common occurrence and has plagued many kits that I have built. Big problems crop up: If the switches are soldered into place there is *no way* that they will fit into the holes in the case later. Maybe the two that I saw were a bad batch, but good advice is to leave the switches loose and solder them after temporarily attaching the metal strip onto which they mount. Ditto for the LED indicators. Use the case itself to determine the correct position.

If you follow the last four steps as written you are in severe difficulty. You are told to solder the line cord to the board. If you are really clever you thought to put it through the hole in the case and install the strain relief first. Otherwise you can just go back, unsolder the line cord, figure out what that funny little plastic thing is for, and do it right the second time. This is unpardonable. You are not even told to tin the leads first. What happened to the writer who remembered to tell you to bend the leads on the resistors?

The speaker cups come glued to the case. They promptly came off in every Pennywhistle I've heard about. Including

THE BEST OF BYTE - VOLUME 1

news release *Received: 77 Jul 12*

This 384 page blockbuster of a book contains the majority of material from the first 12 issues of *Byte* magazine. 146 pages are devoted to "Hardware" and are crammed full of how-to articles on everything from TV displays to joysticks to cassette interfaces. But a home computer without software might as well be a boat anchor, so there are 125 pages of "Software and Applications" ranging from on-line debuggers to games to a complete small business accounting system. A section on "Theory" examines the how and why behind the circuits and programs, and a final section "Opinion" looks at where this explosive new home computer hobby is heading.

Early issues of *Byte* are selling for as much as $15.00. If you're not a fanatic collector, this book represents a much better value per dollar. "The Best of Byte - Volume 1" is available for $12.95 postpaid from Creative Computing, Attn: Pamela, P.O. Box 789-M, Morristown, N.J., 07960, USA.

mine. Stick them back on with cyanoacrylate glue. I used "Hot Stuff". It worked fine.

The case was a headache throughout. There are *no instructions at all* about how it is to be assembled. I found a way of putting it together that made it fit OK. But next time I want a jigsaw puzzle I'll go buy a jigsaw puzzle.

Step 15 is another catastrophe waiting for a victim. I will quote the innocent sounding opening: "Mount the PC board on the metal case backplate . . ." You can guess one of the problems. Nowhere does it say which piece is the backplate. But that's really easy to figure out. The catch is that *once you mount the PC board into the backplate there is no way to make the adjustments called for in the next section of the manual!* Ho ho.

Did you have some rubber feet left over? You will if you follow the instructions. I'm sure you can find a use for four pretty rubber feet. But the final insult in the assembly instructions is step 16, the last step. "Install and wire jacks using ribbon cable provided." You are given a schematic and that's it. As to which goes where, or what those four little lugs on the back of each jack are for, or where to run the wire nothing is said. If you are not familiar with those clever switching jacks and how they operate you cannot complete this step. And your faithful equipment reviewer is waxing wroth.

So much for the assembly instructions. All you needed was the usual tools: a screwdriver, a soldering iron, strippers, some solder, the patience of Job and the detective qualities of Sherlock Holmes. Now comes the adjustment. The manual says, "Adjustment of the Pennywhistle 103 requires a frequency counter, an audio oscillator and VOM or VTVM. An oscilloscope is nice, but not absolutely necessary."

I must agree, an oscilloscope is nice. A frequency counter is nice, too. I have a VOM, and that's it. So much for my Modem project. The *ADVERTISING* for the Pennywhistle should state that exotic stuff is required. It doesn't. The manual has neat 'scope photos. For personal computing, the requirement of a frequency counter, and a statement that an oscilloscope is nice but not *absolutely* necessary (emphasis mine) just doesn't hack it. Your VOM has to be able to set the output level to 0.14 volts RMS, 0.39 volts P-P. Mine is a fancy VOM. Will everyone's VOM be able to read these voltages? I think not.

In sum, putting aside my reservations about the just-plain-bad assembly instructions and the requirement of special equipment for alignment, and considering that it is originate only, the Pennywhistle Modem half fills a need of the personal computer field.

ASSEMBLERS, BASICS, & VIDEOS

Received: 77 Jun 20

Dear Sir,

I just got hold of your March 1977 issue, and it looks good, calling for some comments:

1. The most complete 8080 assembler I've seen is Intel's. It's also the one most likely to be available to industrial 8080 users. Why not try to stick to Intel's formats? The macros are real nice, as are all the symbol-manipulation operators (AND, OR, XOR, SHR, SHL). "SYMBOL SHR 8" is the easy way to get the high 8 bits of SYMBOL. Of course, Intel wants a lot of money for their assembler.

2. I learned BASIC at Dartmouth. Darmouth's BASIC has two kinds of subroutines: the standard GOSUB and subroutine you CALL by NAME, with lists of passed parameters and local variables. Get a BASIC manual from Dartmouth and drool. . . I have a number of programs I wrote at Dartmouth, and I dread having to cram them into a run-of-the-mill BASIC. Has anyone implemented the full Dartmouth BASIC on an 8080? Incidentally, BASIC at Dartmouth is a compiler, just like FORTRAN. Typing RUN causes the system to compile your program, run it, and discard the object code. ALL the editing and saving commands in most BASICs are emulating Dartmouth's operating system, not their BASIC. Dartmouth even has a language, CPS, which accepts BASIC, FORTRAN, and assembly-language statements interchangeably.

3. "Time for Super Video?" Using the computer's main memory for video refresh grossly slows down the computer. The reason is that the video generator needs data so often (a byte every 700 to 800 nSec for a 64- character line display) that the buss spends most of its time servicing the video display,and the processor runs only during the video retrace intervals. The standard 8080 buss takes 1500 nS to access one byte and get ready for the next byte. With a video diplay trying to get a byte every 800 nS, when does the processor get the buss??? Video boards have their own memory so that the buss is not tied up with this continual refreshing of the TV screen.

Peter Traneus Anderson
103 N. Union St.
Burlington VT 05401

COMPUCOLOR USERS GROUP FORMS

Received: 77 Jul 13

Dear Sirs,

A group of us have gotten together to form a CompuColor Users Group. This Group is dedicated to the exchange of programs and technical data for the CompuColor. We anticipate issuing a news bulletin periodically. Subjects such as how to concatinate tapes and disks will be covered. For each accepted program, a member will receive a number of other programs. There will be an initial membership fee of $10.00. This is to cover the duplication and mailing of materials. Those wishing to join the Group may send the fee to S.P. Electronics at 5250 Van Nuys Blvd., Van Nuys CA 91401. Further information may be obtained by sending a large self-addressed envelope to this address. Among our present programs are illustrated versions of BLACKJACK, a right rectification program, a fantastically illustrated version of STARTREK, illustrated slot machines, etc. For the most part we will try to exchange recorded media rather than program listings. Anyone interested is welcome to drop us a line.

Stan Pro
S.P. Electronics
5250 Van Nuys Blvd.
Van Nuys CA 91401

TERSE HEX CLUB NOTE

Received: 77 Jul 7

Hex Users Club for systems with 16 - 24 keys. (EPA, Motorola Evaluation Kit II.) Restricted to 6800 based systems, for now. S.A.S.E. to Charles C. Worstell, 36012 Military Rd. S., Auburn WA 98002. Phone (206) 927-6038 (Tacoma).

PENNYWHISTLE PLAYS A BETTER TUNE

Jef Raskin 77 April 30

Update to "Sour Notes on a Pennywhistle"

This reviewer was gratified to learn that a lot of the problems that were discovered on the Pennywhistle in the accompanying review have also been discovered by M&R Enterprises. Better still, they have corrected many of the problems. This response to user feedback is most praiseworthy. However many of the early units are still out there and purchasers of them should write to the manufacturer at P.O. Box 61011, Sunnyvale CA 94088.

In doing the review I discovered that a cyanoacrylate glue was better suited to holding the acoustical cups than the silicone gunk that M&R used. Sure enough, they discovered the same thing, and the rubber parts now stick on properly.

The pots–the ones that require a frequency counter and an oscilloscope to set up–are now much easier to set as two of them are now multiturn pots, and the others are now cer-met. The old pots had so little resolution that the least fidget with the screwdriver would throw them way out of alignment. Unfortunately you still need the specialized equipment to build a Pennywhistle, and as Marty (who *is* M&R Enterprises) told me, these kits are *not* meant for the beginner, but for people with electronic expertise.

The revision C errata sheet also states which way the IC's are to be oriented. I'm glad they caught this one. A few resistor values have been changed and the errors in the parts listings have been corrected.

I was promised that future advertising for the Pennywhistle will:

1. Make it clear that it requires a frequency counter to align the Pennywhistle, and
2. Explain what an "originate only" modem is. (See the review for definitions and why this is important.)

While the instructions on final assembly have not been materially altered I have been told (although I have not tested a unit) that the case has been corrected and that the switches now fit in the holes they are supposed to. As the review states this is an excellent design electronically, and the parts quality is the best we've seen. There were problems. The manufacturer appears to have corrected most of them, and seems to be conscientiously working to improve his product further, and to make his advertising (which has never been untruthful) less misleading to the typical personal computer buff.

PENNYWHISTLE 103 REVISION C CHANGES

PARTS LIST

R 77 is changed to 5 K variable, cer-met, multi-turn
R 75 is changed to 10 K variable, cer-met, multi-turn
R 78 is changed to 20 K variable, cer-met, CTS 360S203B
R 79 is changed to 5 K variable, cer-met, CTS 360S502B

CIRCUIT BOARD ASSEMBLY

R 79 is moved to a position just below R 74. Adjustment slot should face the board edge.
R 75 and 77 positions are changed to accomodate low-profile multi-turn units. Adjustments slots should face bottom edge of the board.
R 76 and R 78 are moved slightly. Their adjustment slots should face away from the board edge.
D 25 is inserted to the right of D 19.

SCHEMATIC

Value of R 75 is changed to 10 K.
Value of R 78 is changed to 20 K.
Connection between E1 and E2 is removed.
D 22 and D 23 exchange reference designations.
R 37 (680 ohm 1/2 watt) should be R 31.
Value of R 79 is changed to 5 K.

ASSEMBLY INSTRUCTIONS

All IC's and the opto-isolator are inserted with their pin 1 end (with a notch or small indented dot) facing toward the top side of the board (the edge with the 25-pin connector).

NOTE: Part No. 1458 same as No. 5558
Polystyrene Film Capacitors can be substituted for Silver Mica types.

PCNET – A PERSONAL COMPUTING NETWORK

Recieved: 77 Jul 6

The PCNET (Personal Computer NETwork) Committee has been functioning in the Palo Alto area since the April Computer Faire. The committee's goal is the creation of regional (followed by national) personal computer networks for the computer-to-computer transfer of messages and files. A set of network protocols (sets of conventions defining all levels of intercomputer communication) is almost completely designed. These protocols should be operable in 8K bytes of machine code, and are designed to be implemented in string BASIC.

The committee believes this should be attractive to personal computer users. Participation will be voluntary; you can decide to participate (or not) on any given day of network operation. Network functioning will be relatively insensitive to the absence of an appreciable fraction of member computers.

Our current thinking indicates the following tentative equipment required for participation in the network:

A personal computer with 12-16K
of RAM and string BASIC.

An originate/answer MODEM
capable of 300BPS.

A message service – the ability to send a message (generally English text, although almost any file can be sent) is quite valuable. It doesn't sound very dramatic, but it is surprising how powerful and efficiency – improving such a message exchange facility is. What keeps ordinary message services (telephone, telegraph, mail) from working as well seems to be a combination of factors: too slow (mail); often hard to catch someone (phone); hard or time consuming to use (mail, telegrams); expensive in terms of characters per dollar (phone, telegrams); etc. A computer based message system overcomes most of these difficulties.

People regularly using such a system rapidly develop a whole new communication style. Most messages are brief – 500 characters or less. Content tends to be much more informal and direct than conventional media such as business letters. Message system users move rapidly toward a computer-based personally oriented file system containing messages, distribution lists, text files, etc. The difference is one of kind, not just of degree.

The PCNET Committee is about to start a series of experiments. We would welcome people with personal computer systems who would like to participate; we're especially interested in people in the Palo Alto dialing area. We would also be most interested in similar network efforts in other places. We'd like to avoid west coast chauvinism and want to work closely with people in other parts of the country.

For further information write or call:

Dave Caulkins
415-328-2411 (wk)
415-948-5753 (hm)
437 Mundel Way
Los Altos, CA. 94022

S-100 INTERFACE TO FRIDEN/BURROUGHS TERMINALS

Dear D.D. *Received: 77 Aug 1*

Just a quick note on a subject that should be of great interest to some of your readers: the Friden (or Burroughs) data terminals that were sold last year by Herbach & Rademann and others. After many months of sweat and toil my partner and I have come up with a Unique interface circuit for these terminals. Our circuit completely eliminates the logic box that was sold with these teminals, and in doing so it eliminates the special protocol required by that box. Instead, a conventional TTY driver routine may now be used to drive the terminal (Altair BASIC required NO changes!).

This circuit has a number of useful features. For example, a single switch will cause all lower case alphabetics typed at the keyboard to be transmitted as their upper case counterparts (The aforementioned BASIC can't digest lower case!). We've also managed to solve the half vs full duplex problem caused by the fact that the keyboard and printer are mechanically linked. We have built in logic that supresses the 'echo' characters normally sent by a full-duplex system. Of course, a single switch disables this feature, for use on systems that support half-duplex operation. Another nice feature is that no matter what case the last character printed by the computer was in, the terminal will always stop in lower case (if you've ever used one of these terminals, then you know this isn't as trivial as it sounds), making it much easier to use programs that take a lot of numeric input.

The entire interface circuit, consisting of some 30 or so IC's and a number of discretes, has been placed on a single S-100 PC board, which may be dropped directly into any computer using this bus. However, the card is so designed that by changing about 5 chips, it instantly converts to standard RS-232 serial I/O, able to communicate with virtually any computer. All of this is designed into the board and requires no trace-cutting or chip-gluing for either form of the circuit.

We had originally planned to do the circuit in wire-wrap, but after building a few of them we realized it's just too complex to be practical. We are planning on selling the printed circuit card for $55.00, or a complete kit of all parts, PC board with silk-screened parts layout, and complete documentation for $150.00. If interest is great enough to warrant a larger batch of PC boards than is currently planned, the prices will be reduced considerably.

Interested persons should write or call one of us at the address and phone # below:

Thanks,

Bruce Ratoff or Jeff Colter
(201) 963-0848

Box 28-D, Apt. 13B
1 Marineview Plaza
Hoboken, NJ 07030

INTRALINE EDITING MOD TO WANG'S PALO ALTO TINY BASIC (DDJ-5-76)

John Bunting
2444 N. Delaware K4
Springfield MO 65803

Received: 77-3-10 at Homebrew Computer Club from Lichen Wang

Rubout echos deleted character. This may be repeated until first character in line is reached.

Control R gives *CRLF* then echos corrected line.

This saves counting backslashes and allows an up-to-date echo of the current line at any time before *CR* by hitting *Control R.*

VDM-1 MODIFICATIONS

Lichen Wang
150 Tennyson Avenue
Palo Alto, CA. 94301

Received: 77-8-3 at Homebrew Computer Club

VDM-1 from Processor Technology has two little bugs. Bug number one is when you access the screen memory frequently, you get a log of flickers on the screen. Bug number two does not happen very often, but when it happens, it may bomb your program. Say you have a program like this:

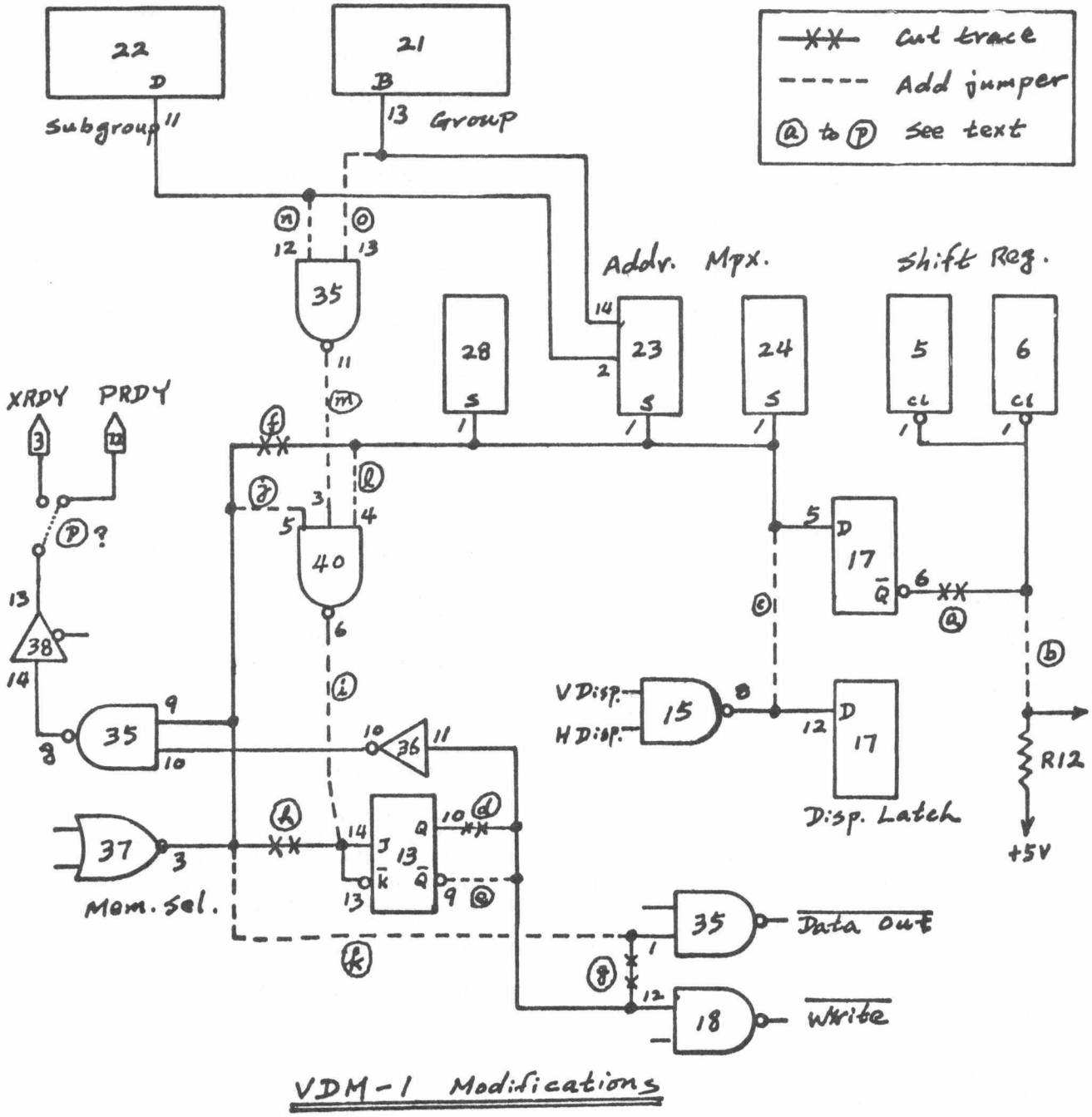
```
...
LDA   0CC00H      ;read from screen
MOV   E,A         ;or any other instruction
...
```

Sometimes the instruction MOV E,A will be ANDed with the byte read from CC00. This happens more often in Crommenco's Z-1 or Z-2 systems than other systems due to timing. Processor Technology published a modification on Access issue #1 and #2. But (1) you have to modify both hardware and software, (2) it slows down program execution considerably, (3) it does not remove bug number 2, and (4) it makes bug number one much much worse on some systems.

To cure bug number one, we can generate a wait state (by pulling RDY line on the bus down) when the CPU tries to access the screen memory while the video part is using the memory. The maximum wait will be 45 u sec. During truly random access the average wait will be 15 u sec. To our bug number two, we only need to change the timing of Data Out strobe.

What follows is a modification that will remove both bugs – at least for the systems that I have tried, Altair 8800, IMSAI, Z-1, and Z-2. No software modification is needed, screen memory access from CPU is slowed down somewhat, and have but one side effect that I know of. The side effect is that it is difficult to read the content of the screen memory from the front panel after this modification. There are 5 traces need to be cut, and 10 jumpers need to be soldered. All modifications are on the solder side of the board. Step (g) and (k) are for bug number two. If that is the only thing that bugs you, you may do these two steps only. On the other hand, if bug number two does not bug you, you may skip these two steps.

(a) Cut trace from IC17 pin 6.
(b) Follow the trace of step (a) to the lower right corner of the board, it should pass near a pull up resistor R12. Solder a jumper between the resistor and the trace.
(c) Solder a jumper between pin 5 and pin 12 of IC17.
(d) Cut trace from IC13 pin 10.
(e) Solder a jumper between IC13 pin 9 and the trace of step (d).
(f) Cut trace from IC37 pin 3.
(g) Cut trace from IC35 pin 1.
(h) Cut at IC35 the trace from IC35 pin 9 to IC13 pin 13 & 14.
(i) Solder a jumper between IC40 pin 6 and the trace of step.
(j) Solder a jumper between IC40 pin 5 and IC35 pin 9.
(k) Solder a jumper between pin 1 and pin 9 of IC35.
(l) Solder a jumper between IC40 pin 4 and the trace of step (f).
(m) Solder a jumper between IC40 pin 3 and IC35 pin 11.
(n) Solder a jumper between IC35 pin 12 and IC22 pin 11.
(o) Solder a jumper between IC35 pin 13 and IC21 pin 13.
(p) Check the jumper between IC38 pin 13 and bus pin 3 or 72. For most systems, this should be at bus pin 72.



VDM-1 Modifications

COMMENT ON Z-80 CODE STANDARD PROPOSED

Received: 77-Jul-1

Dear *DDJ*,

The proposal for standard Z-80 object code (DDJ, March 1977) precludes external references with an offset such as:

```
LDA xname+4
```

Thus, if reference is to be made to several elements in an array, each element would require a separate external name.

The problem can be relieved by adding a new type of record (05). This offset record would be applied after the address has been formed via relocation. The effect would be to add an offset to the final address. The format could be:

3.6 Offset Record (type 05)

The address in the program which must be offset is explicitly defined as is the offset value. One address is offset per record. An operator code is included for future expansion. Currently the operator code is always 0 (=addition). For subtraction add the twos complement.

Byte 1	Dollar sign ($) delimiter.
2-3	Type of operator, now always ASCII 0 (=addition/subtraction).
4-5	Most significant byte of address to be modified.
6-7	Least significant byte of address.
8-9	Record type 05.
10-11	Most significant byte of offset.
12-13	Least significant byte of offset.
14-15	Binary checksum.
CRLF	Carriage return, line feed.

Very truly yours,

David L. Ripps
Industrial Programming Inc.
9 Northern Boulevard
Greenvale, New York 11548

DISSECTING THE SWTPC CONTROL INTERFACE

Phillip Schuman
1354 Finley
Lombard IL 60148

For the new user of the SWTPC 6800 system, it seems easy to do I/O - just let MIKBUG* take care of it. But the question of how MIKBUG does I/O is a real mystery to the newcomer. I came up with that question and after searching through some manuals and schematics, finally got it all straight - - it's magic. Well, not really, but it is helpful to learn how the system uses both the software (MIKBUG) and the hardware (PIA device) at the control interface port.

I will approach this subject in four parts; the addressing and selection requirements of the system, the PIA (MC6820 device) itself, SWTPC and MIKBUG usage of the PIA, and the software of MIKBUG and its inner workings.

ADDRESSING

The MIKBUG ROM software makes certain assumptions about the hardware configuration of the system it will be running on. Each I/O interface is represented by four memory locations, as is the control interface that is used by MIKBUG. All interfaces are in an address range of 8000 to 801F (hex). The control interface is referenced at locations 8004 - 8007 (hex). This meets the requirements of the MIKBUG software. The actual mapping of the locations is shown in figure 1. The selection of the proper set of addresses (an I/O port) is done through decoding on the mother board. The appearance of an 80 (hex) on address lines A15, A14, A13 will indicate that the I/O ports are being accessed. Address lines A4, A3, A2 will be decoded and will select one of the eight I/O ports by presenting a logic low to the CS2 pin on the proper PIA.

To enable a PIA, the CS0 and SC1 (chip select zero and one) lines must be at logic high, and the CS2 line must be at logic low. The SC0 and CS1 lines are constantly held at logic high (+5 VDC). The remaining line CS2 is tied to the address decoding on the mother board. Having decoded to the proper port (PIA), the internal registers must now be accessed. This is easily done because the A0, A1 addressing lines are tied directly to the RS0, RS1 (register select) pins of the PIA. Figure 1 shows how the RS0, RS1 lines are used to select the internal registers of the PIA.

Figure 1. PIA ADDRESSING

	A_4	A_3	A_2	A_1	A_0	CR b_2	
8004_{16}	0	0	1	0	0	0	PIADRA
8004_{16}	0	0	1	0	0	1	PIAORA
8005_{16}	0	0	1	0	1		PIACRA
8006_{16}	0	0	1	1	0	0	PIADRB
8006_{16}	0	0	1	1	0	1	PIAORB
8007_{16}	0	0	1	1	1		PIACRB

THE PIA

The MC6820 PIA is comprised of two sections; A and B. Each section is essentially a twin except for voltage/current handling capabilities. Each section has three internal registers that are addressed from the outside world via the PIA memory locations. There are only two memory locations per PIA section, and there are three registers to be accessed, so one memory location has to do double duty. The single memory location is the control register (CR). The other memory location references either the output register (OR) or the data direction register (DDR), depending on a bit set in the control register.

THE PIA CONTROL REGISTER

The control register as shown in figure 2, contains bits that establish the status of the PIA. Bits b0, b1 would program the output control CA1, but are not used by the control interface. Bit b2 is the indicator for selecting the PIAOR or the PIADR: If b2 is zero the PIADR, if 1 the PIAOR. Bits b5, b4, b3 on the SWTPC system control the CB2 control line. In this usage, b5 is always held at logic high, indicating that CB2 will be acting in an output mode. Bit b4 is always logic high indicating that the CB2 output will be latched, following the logic state of bit b3. Bit b3 is a driver for the CB2 control line; if b3 is low so is CB2, if b3 is high CB2 is also high. Bits b7, b6, are used to control the interrupt (IRQ) flags, and are not used in the control port.

Figure 2. CONTROL REGISTER LAYOUT

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
IRQ1 FLAG	IRQ2 FLAG	CA2 (CB2) CONTROL			DDR ACCESS	CA1 (CB1) CONTROL	

THE PIA DATA DIRECTION REGISTER

The data direction register is a bit map of the output register. Each bit in the DDR will indicate how it's counterpart in the PIAOR will behave - as an input or output line. A one bit indicates an output mode, and a zero bit indicates an input mode of operation. The DDR is referenced by setting the b2 bit in the PIACR to a zero, and then addressing the PIAOR/PIADR memory location.

THE PIA OUTPUT REGISTER

The PIAOR is nothing more than a buffer between the two worlds we are dealing with. The manipulation of the eight data bits in the PIAOR depends on the settings of the bits in the DDR. Since each bit is a complete bi-directional data path, extreme care must be taken when accessing the register as a whole. This software concern will become apparent when we look at the driving software later. Each data bit is addressed in the outside world by an appropriate bit number as shown in figure 3 (note that not all bits are used by the SWTPC control interface). The PIAOR is referenced by setting the b2 bit in the PIACR, and then addressing the PIAOR/PIADR memory location.

SWTPC USAGE

The A section of the PIA will simulate a bi-directional serial I/O port. Bit b0 is used for serial output, and bit b7 is used for serial input. The two end bits were chosen because of software considerations as shall be seen later. No other bits of the A section are used, as can be seen in figure 3.A.

*MIKBUG is a registered trademark of Motorola Corporation

The B section is used as a clock, or rather as a counter driven by the data clock pulses. Since we are generating and receiving serial data by software simulation, the software has to be provided with proper timing pulses. Section B provides the control interface with an indication of elapsed time. The timed interval is equal to either the duration of a single bit, or a single ½ bit. The data clock rate is equal to sixteen times the baud rate. This 16X rate is fed to the counter, which divides by 16 for a full bit interval, or divides by 32 for a ½ bit interval. Bit b0 will reset the counter, (logic high) and the counter will not start until bit b0 goes logic low. The output status of the counter (divide by 16 or 32) is selected by bit b2. Bit b7 will have a logic high when the selected count is satisfied. Bit b6 is tied to logic high or logic low, and is used during the transmission of an ASCII character. It is used to indicate the number of stop bits to be generated, but it is not used for receiving characters.

Finally the CB2 control line has 2 purposes in the control interface. Control line CB2 follows certain bit settings in the control register. To turn CB2 on (logic high) a 3C (hex) is loaded into the PIACR bits b5, b4, b3, b2. The CB2 line acts as a reader control when it is turned on by a MIKBUG LOAD command. In addition, the CB2 line is logic gated with the output RS232 line from the control interface. The RS232 input line is wired to be echoed on the RS232 output line. The CB2 output controls this echo hardware feature. If you load a program via a MIKBUG LOAD command, the CB2 line is active, which also means that the RS232 output line is turned off. After the loading is complete, a store of a 34 (hex) into the PIACR bits b5, b4, b3, b2 will turn off the CB2 line and restore the hardware echo feature. The b2 bit was accessed to insure that the PIAOR would be selected upon referencing the other memory location.

MIKBUG SOFTWARE

Two MIKBUG programming functions will be considered; initialization of the PIA, and reception of characters at the control interface. Since the PIA is a programmable device, it must be configured upon a powerup or reset condition. The RESET line in the SWTPC system will initialize the PIA so that all registers will be zeroed. Starting from this zero state, MIKBUG software will configure the PIA for handling I/O operations. Figure 4.A contains a detailed listing of the PIA initializing sequence. Figure 4.B reflects the status of the PIA during this period of time. After this has been digested, the PIA input software can be looked into. This software is reflected in figure 5. It is very liberally commented and should be easily understood at this point. Note the use of the ROR and ROL operations to select off the bit from the PIA. This was the software consideration for selecting the two end bits for the I/O operations.

This is the end of the road, we hopefully have shed some light on the SWTPC control interface, along with bringing MIKBUG out from behind the shadows.

```
                    NAM   POWER
                * ********** ********** ********** **********
                * MIKBUG POWER UP SEQUENCE FOR INITIALIZE OF PIA
                * ********** ********** ********** **********
1                       LDX   8004      POINTS TO PIADDA
                *                       THE PIACRA IS ZERO, THUS DDRA BIT IS ZERO
2                       INC   0,X       SET BIT 0 TO "1" IN DDRA ---- OUTPUT
3                       LDA   A 07
4                       STA   A 1,X     RESET PIACRA FOR PIADRA ACCESS
5                       INC   0,X       SET "01" IN PIADR
                *                       BIT 0 IS THE SERIAL OUTPUT LINE
                *                       AND IT IS NOW SENDING ALL "1" BITS
                *                       TO HOLD IT IN AN "IDLE" STATE
6                       STA   A 2,X     SET DDRB TO "J7"
                *                       ACCESS IS TO DDRB, SINCE PIACRA IS ZERO
                *                       FROM THE INITIAL "RESET" FUNCTION
7                       LDA   A 34
8                       STA   A 8007    SET PIACRB TO "34"
                *                       THE CB2 LINE IS SET BY THE "3"
                *                       PIACRB ACCESS TO PIADDB SET BY "4"
9                       STA   A 8004    SET PIADDR TO "J4"--RESET COUNTER
```

PCF IS CENTER FOR PERSONAL COMMUNICATIONS LAW

news release *Received: 77 Aug 1*

The Personal Communications Foundation (PCF) is no longer in its formational state. It has been lifted off the drawing board, is actually in operation and helping Hams, CBers and their attorneys.

The PCF is a non-profit, tax-exempt foundation organized expressly to serve as a central clearinghouse of information and research in the area of personal communications law. The PCF is the only organization of its kind because personal communications law is not yet formally recognized as a legal specialization. This is true although statistics indicate that in 1976 alone there were 7,000 legal matters pending involving all aspects of personal communications.

The cost of litigating such a specialized lawsuit can be in excess of $10,000. Approximately 40% of that cost represents time devoted to research. An attorney with access to the PCF's specialized personal communications library of briefs, cases and in-depth studies can eliminate much of that research time and consequently reduce litigation costs by thousands of dollars. Further, because most attorneys are not familiar with the nuances of personal communications law, an attorney using PCF materials can better represent his client. For example, the PCF has spent in excess of $2,000 funding a study on the issue of Federal pre-emption of the personal communications field by the Communications Act of 1934. This study was conducted by a constitutional law expert who is also an engineer. The resulting legal memoranda has already been incorporated into a brief before the California Court of Appeals.

PCF is controlled by a twenty-four member Board of Trustees. Each Trustee is either a practicing attorney or law school professor. The Trustees hold amateur radio, citizens band and first and second class commercial licenses.

The PCF cannot run on the energies of its Trustees alone. In order to continue to build its library and fund additional sutdies in areas where existing case law is in conflict or in those areas which have not been adequately briefed to date, substantial funding is necessary. PCF estimates it needs $250,000 in its first two years to be fully effective.

Thus far the PCF has gotten off the ground thanks to significant contributions by the Amateur Radio Relay League, Wilson Electronics and Yaesu. But the funding necessary to function at full potential has not been reached. As the PCF is a membership corporation, much of the funding is expected to come from membership contributions. With over 300,000 amateur radio operators and 14,000,000 CBers in the U.S., membership contributions should be substantial. Four classes of membership have been established. These are known as associate membership, for a yearly contribution of $10.00, full membership for a yearly contribution of $25.00, contributing membership for a yearly contribution of $100.00 and life sustaining membership for a single contribution of $250.00 or more. All members except associate members are eligible to vote for Trustees. All members of the foundation will receive the quarterly PCF Newsletter.

Contributions to the PCF are *tax deductible.*

The PCF office has recently been relocated to 10960 Wilshire Blvd., Suite 1504, Los Angeles, California 90024. The move was made to facilitate administration of the PCF by its newly elected president, Kenneth S. Widelitz, a Los Angeles attorney. All membership contributions and inquiries should be sent to the PCF at the above address.

Figure 5.

```
                NAM   READ
* ********************************************************************
* ***   MIKRUS READ ROUTINE FOR ASCII--NO PARITY              ***
* ********************************************************************
IN1     LDA   A 0,X       LOOK FOR TRANSITION ON INPUT
*                         THE LINE IS HELD "TRUE" BY
*                         SENDING ALL ONE BITS OF "MARK" BITS
*       BMI   IN1         LOOP IF MINUS.....
*                         "MARK" BIT AT "PA7" PIN WILL MAKE
*                         THE REGISTER HAVE A "1" IN BIT 7 (MINUS)
* ********* A "START" BIT TRANSITION HAS OCCURRED
        CLR   2,X         SET COUNTER FOR 1/2 BIT TIME --PR2=0
        BSR   DEL         START COUNTER...
        BSR   DEL         WAIT FOR 1/2 START BIT TO GO BY
        LDA   B 04        SET COUNTER FOR FULL BIT TIME
        STA   B 2,X       SET COUNTER ...PR2=1
        ASL   B           SHIFT "04" LEFT GIVING "08"
*                         OUR BIT COUNT---7 DATA BITS + 1 PARITY
IN3     BSR   DEL         WAIT FOR 1 FULL BIT TIME
*                         ARE LOOKING AT "MIDDLE" OF BIT TO INSURE
*                         PROPER DECODING OF LOGIC STATE
*       SEC               SET CARRY BIT IN CONDITION REG
*                         THIS WILL BE USED TO KEEP THE "1" BIT
*                         IN THE "OUTPUT" LINE, AS WE USE THE
*                         REST OF THE "DIADRA"
        ROL   0,X         GET INPUT (PA7) BIT INTO CONDITION
*                         CODE REGISTER.....FULL ROTATE
*                         "OUTPUT LINE"=OLD CARRY BIT
*                         NEW CARRY BIT="INPUT LINE"
        DEC   A           SHIFT NEWEST INPUT BIT INTO REG A
* ******* THE INPUT CHARACTER WILL MARCH INTO THE SYSTEM
*         A BIT AT A TIME, AND BE STACKED INTO
*         REGISTER A AS THEY COME IN.
        DEC   B           SUBTRACT 1 FROM OUR BIT COUNT
        BNE   IN3         IF NOT ALL DATA BITS IN, GET THEM
        BSR   DEL         WAIT FOR SINGLE STOP BIT TO GO BY
        AND   A 7F        RESET "PARITY" BIT TO A "1" (NOT USED)
        CMP   A 7F        CHECK IF CHAR IS A RUBOUT/NULL  "7F"
        BEQ   IN1         IF A RUBOUT, FORGET IT, GET NEXT CHAR
* ********** AN ASCII CHARACTER IS NOW IN THE
*                         SYSTEM AND HELD IN REGISTER A
*                         DO WHAT YOU WILL WITH IT
DEL     TST   2,X         WAIT FOR COUNTER TO SET BIT --- PR7
        BPL   DEL         LOOP IF POSITIVE---NO BIT AT PR7
DE      INC   2,X         RESET COUNTER---PR2=1
        DEC   2,X         TAKE OFF RESET BIT, LET COUNTER RUN
        RTS               RETURN WHENCE WE CAME
```

TOWARDS A *GOOD* u-C LANGUAGE

Folks: *Received: 77 Jul 28*

I have some concepts which I would like to get some crosstalk on. Any response to these ideas from users of microcomputer systems in general, and those users with experience in the design and implementation of higher-level languages, would be most appreciated.

I am concerned about the process that is taking place regarding the implementation of languages such as BASIC, FORTRAN, and others for microprocessors. A great deal of time and energy is being spent in adapting these languages for smaller machines, and I believe that this time and energy is being misspent. The common, familiar high-level languages were designed with the big computers of the day in mind: today's FORTRAN is a lineal descendant of the FORTRAN of the late fifties, encompassing all of the corners that had to be cut to implement a high-level language for a big machine. Similarly, BASIC was designed for a time-sharing environment. But more important, these languages were designed in an earlier era, before many of today's concepts of what is a "good" programming language had been evolved.

I am not a member of the "kill BASIC" school: I believe that BASIC has its purposes. Indeed, as its acronym implies, it is a good beginners' language, excellent for those interested in neither the nuances and aesthetics of more sophisticated languages nor the specialization of the languages designed for more specific purposes. It is widespread (though the many dialects of BASIC make portability somewhat of a problem), and it is a relatively safe language for timesharing users – there is nothing the single user can do that is likely to bomb out an entire system. And, admittedly, it is quite easy to use.

The point is, however, that BASIC was not designed with the microcomputer user in mind. It is extremely important that researchers not spend time and effort re-inventing the wheel: the existence of journals such as ***DDJ*** and ***KB*** help propagate ideas to the public, and save a lot of important man-hours (person-hours, whatever). But it is even more important to realize that the tools appropriate for one job – writing programs for a megacomputer – may be grossly inappropriate for another job. I am considering pursuing research into this field in the near future (probably as my doctoral research). Here are some ideas I came up with just in the space of a few minutes:

1) Many uC users who will be using a high-level language on their own machines will have some, if not a lot, of familiarity with the assembly language or the machine language of their own machines. Facilities within my proposed language, which I refer to as GOLLY (Gordon's Original Language?) (Yes!), would include compile-time macro facilities, so that the user could include assembly language code within a GOLLY source program. This would also facilitate experimentation and real-time control.

2) There are already a good number of direct memory access devices on the market today, notably the VDM and Cromemco's Dazzler. However, high-level languages in particular, and BASIC and FORTRAN in particular, have no standard provision for writing and reading directly from memory. GOLLY would have explicit facilities for accessing any portion of memory easily and efficiently. Probably a complete implementation of GOLLY would include address checking so that certain areas of memory would be protected from writing (such as the compiler/interpreter itself). Note: I have no idea yet whether an interpretive language or a compilable language is the direction in which to go; I welcome comments.

3) I/O facilities should be carefully considered, and designed in such a way to take advantage of the I/O facilities of μC's. Special attention should be paid to the difference between regular I/O devices such as TTY's and DMA devices such as CRT's – different facilities should be available for each.

4) A language without recursive abilities is a waste of any microcomputer's potential. As far as I know, most μC's utilize a software stack and use it to store return addresses for called routines, a method which makes recursive subroutine call at the very least simple. Neither BASIC nor FORTRAN allow for recursive subroutines.

5) Current philosophy of language design, as well as the considerations of program proving and modularity, make a structured design imperative – FORTRAN and BASIC are the two least structurable languages around (except for maybe APL, which has both merits and problems of its own, not the least of which is its beautiful but uncommon character set). This will probably complicate the implementation of any compiler, but in my own humble opinion, the benefits of a structured language far outweigh the disadvantages.

These are just a few ideas. I would appreciate any response I could get from the readers of this magazine, either in the letters column or by personal correspondance. Or, for that matter, if you happen to be passing through Eugene, drop by.

Joshua P. Gordon
Dept. of Computer Science
University of Oregon
Eugene, OR 97403
Phone: (503) 686-4426

EMPL, A MICRO "APL" FOR 8080's

Dated: 77 May 18

EMPL, an APL-like interpreter, is available on Tarbell Cassette Tapes. It runs comfortably in 8K. The actual program takes 5,376 bytes. It has numeric and string vectors, user defined monadic and dyadic functions, double byte interger arithmetic (±32767), and APL operators adapted to the ASCII character set. The cassette & user's manual is available for $10. Write:

Erik T. Mueller
Britton House
Roosevelt, NJ 08555
(609) 448-2605

NO MO' 8080 DISASSEMBLER, SOON

77 July 6

NOTICE: The 8080 disassembler printed in February '77 *DDJ* and offered on tape by me through the Chicago Area Computer Hobbyist Exchange Software Library will no longer be available after October 1, 1977. I am perfectly willing to arrange for any person or group to pick up distribution.

Sincerely,
Ward Christensen
688 E. 154th Street
Dolton IL 60419

MICROCOMPUTER APL CONTEST ANNOUNCED

news release *Received: 77 Aug 8*

BYTE Publications, Inc., has announced "The Great APL Interpreter Contest" in the August 1977 issue of *Byte* magazine. The contest was inspired by a three part article appearing ing the August, September, and October issues of *Byte* documenting the functional design of an APL interpreter which can be implemented by personal computer experimenters. One or more $1000 prizes will be awarded to authors of contest entries, which must be postmarked no later than midnight, February 28, 1978.

Contestants are free to write their interpreters for any microprocessor they choose. Entries will, however, be judged on their suitability for use on small systems with a minimum of 16K bytes of memory, as well as on programming elegance and efficient use of space. Entries must be in the form of a publication quality manuscript which describes the implementation of the interpreter and which includes source code and object code. Contestants should also submit machine readable source and object code in the form of paper tape or cassette tape. Judging will be done by the editors of *Byte* magazine. Those seriously interested in entering this contest should call Carl Helmers, or Chris Morgan at *Byte*, (603) 924-7217 in order to be included in mailings of further information about the contest's progress.

BYTE Publications, Inc.
70 Main Street
Peterborough, NH 93458

ON THE EFFECTS OF FILLING CAVITIES WITHIN THE FILLINGS OF CAVITIES WITHIN . . .

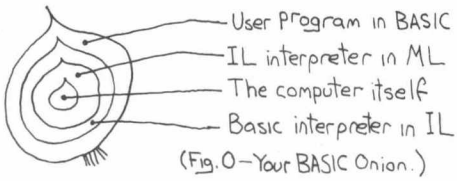
Presented to the members of the Dobbs School of Computational Dentistry

Steve Witham
168 Painter Road
Media, PA. 19063

Submitted for your approval: a small softwaric tidbit[1] found floating in a mental cove - - a piece of psychic driftwood . . .

* * *

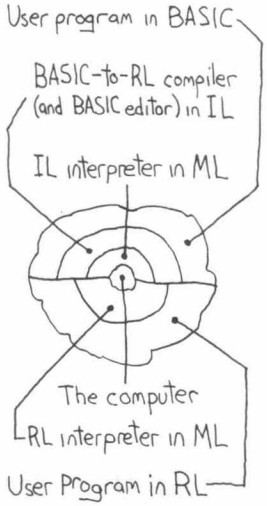
Consider the way many Tiny BASIC interpreters work: the core of the program is written in machine language (ML); it interprets another program which is written in "Interpretive Language" (IL), and then the IL program interprets your BASIC program. This layered approach has been compared to an onion. (See figure 0) Okay, you know this already.



(Fig. 0 — Your BASIC Onion.)

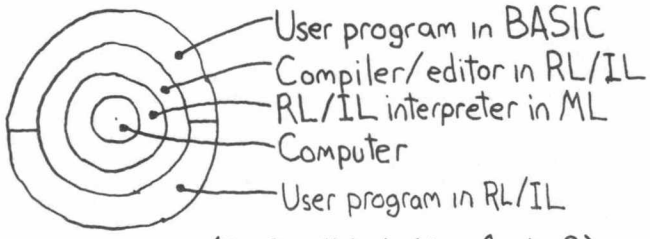
Now consider how some of the more advanced BASIC interpreters work - - for instance, Dick Wilcox' AM-100 / Sixteen/8 / CM-16 BASIC (See Jim Warren's explanation on p. 7, DDJ V2#1) and, I *think,* HP's BASICS. After you say RUN, your BASIC program is digested into a more compact, easier-to-run form, which is interpreted by a small run-time program. This compact form is not just a byte-for-word substitution, as with Altair 8800 and Wang 2200 BASICs, but a complete restructuring of statements. Line numbers and comments are removed, expressions are changed into Polish notation, line and variable references are converted into addresses, etc.. So your program actually gets translated (compiled?) into another language. Lets call it RL, for Run-time Language. (Let's *not* call it RTL!)

Now it seems to me that these two ideas have a natural attraction to each other. I mean, at least they both would appeal to the same (unconventional) type of compiler-writer. But suppose this attraction caused them to coalesce into one blob. If you're the type who likes to invent compact, special-purpose, intermediate language, why not write two of them into one compiler? Implementing both of these ideas in the same compiler/interpreter would create a thing like figure 1.



(Fig 1)

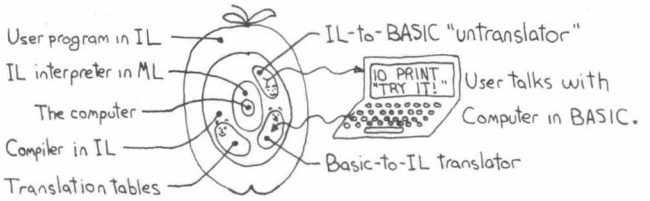
And now that they're together, the attraction grows stronger! Under tremendous pressures from within (limited memory space) and without (our hypothetical wierd programmer), the interpreter undergoes tremmors and shifts - - perhaps the programmer considers writing the IL interpreter in RL, or the RL interpreter in IL, or sharing subroutines between the sections of the interpreter - - and then it happens: rumble-rumble-Fwathlumph! The interpreter collapses into a tight ball, with all distinction between IL and RL lost! The interpreter would now look like figure 2.



(Fig. 2 — It looks like a fig, too?)

This thing is now the most primitive form (neutron star[2]) of what I call a "comp-IL-er". From now on, let's call this dual-purpose intermediate language IL, even tho it's a different concept than what we *were* using the term for.

But this isn't as far as it goes. If RL (now RL/IL, or just IL) is more compact than the straight ASCII representation of BASIC, why not do all of our storing of programs in IL right away? This would necessitate a routine (in IL, of course) to translate IL-coded-BASIC back into human readable BASIC, for LISTing and pointing out errors, for instance. You could probably share a lot of the already-needed BASIC-to-IL tables. Now we've advanced to stage four (see figure 3).



(Fig. 3 — The fig leaves. But naive Eve didn't know what the apple contained!)

Now, since our translation is done with tables, why not allow the user to add to the tables - - i.e., have user-defined commands, syntax, etc.? And since we're translating into IL, why not give the user full use of all the IL subroutines - - i.e., provide BASIC[3] keywords for all of the IL "opcodes"?

* * *

You are now entering the Twilight Zone.

* * *

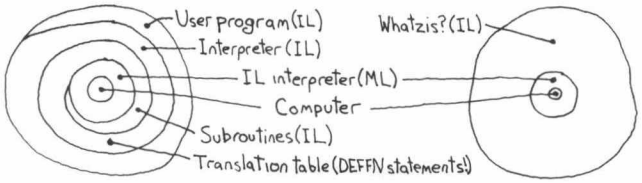
For if the user defines his own commands, are the definitions part of his program or part of the interpreter? And if every IL code translates into BASIC[3], and vice-versa, why not LIST the whole interpreter as a BASIC program? And if it In other words, with this type of comp-IL-er, what is the distinction between the interpreter and a user program in BASIC, as viewed by someone just walking in the door?

"No dee-fair-awnce!" proclams our visiting software chef. Our whatever-it-is has lost another layer. Inside, outside, they're all the same side! Shall we call it a black hole? A Klein comp-IL-er? A Cheshire Onion (which fades out, leaving only the tears)? See figure 4 and decide for yourself.

CENTRAL STANDARDS LIBRARY (CSL)

news release *Received: 77 Jul 11*

To help solve some of the standards problems in the hobbyist computer and microcomputer field, ALF Products is sponsoring a Central Standards Library. After discussions with several manufacturers in this field at the First West Coast Computer Faire, ALF has set up the CSL as a means of standards information exchange for manufacturers, consumers, hobbyists, and others interested in standards. The Library will collect submitted standards and distribute them on a non-profit basis. For more information on available standards, on how to submit standards, and on the Library's services; send $1 (to cover printing and mailing costs) to: The Central Standards Library; C/O ALF Products Inc.; 128 S. Taft; Denver, CO 80228. You will receive a copy of the first CSL Newsletter and the first submitted standard (a parallel interface standard). Manufacturers currently participating include: ALF Products, IMSAI Manufacturing, PolyMorphic Systems, Proko Electronics, Vector Graphic, and Video Terminal Technology.



(Fig.4—Fill in the Whatzis.)

Actually, someone may have already done this already. As a parallel, think of astronomy-type black holes. Even tho the idea was just recently discovered, they've been around much longer than Mankind (*if* they exist). But *I've* never even heard of the *idea* of a Klein comp-IL-er before.

So what is it good for, besides blowing minds? Well, I think I'll let that for you to figure out 0 distinguished doctors of tooth and toggle-switch. (But I definitely think it *would* have useful advantages.)

And how could this all be done? I have only an inkling of where to *begin* to begin, friends. I an but a lowly programmer who writes things like a four-voice polyphonic 8080 tone-geneRator wIth foUr OCTave RanGe aNd DrIVeR PROGRAM WHICH - - AARRGH! (Sorry, my commercial half got away with me there. All under control now. *All* under control.) SIMPLE OUTPUT CIRCCCKaaahk - - ahem. And I also really believe it CAN be done.

Thank you, ladies and gentlemen. I will now open the floor (creeak!) to questions . . . yes?

[1] Look at that word in a mirror sometime.

[2] In a neutron star, all distinction between + and - charges is lost.

[3] I'm using BASIC in the Expanded sense - - a language with the structure and commands of minimal BASIC, but with added commands, data types, etc..

●●●●●●●●●●●●

Dear Editor Persons:

Do you think this is too drawn out to have the desired effect? (That is, wide eyes with dilated pupils, soft blue glow around the head (caused by a surge in the Force), disorientation, repeated pronouncements such as "wow" and "fa-far-faarrout".)

On the other hand, if it were shortened to remove the fun and games, would it be appreciated by people who hadn't thought of the idea already?

By the way, is it original? I wrote this between 12 and 2 in the morning . . .

The APL Core Project

APL-Core
TEXAS A&M GROUP DEMONSTRATES THE FEASIBILITY OF MICROCOMPUTER FULL APL

Dale L. Schafer
APL Committee
Texas A&M Microcomputer Club
1308 E. 27th Street
Bryan, Texas 77801

Received: 77-7-7

Dear Mr. Warren,

I have enclosed a copy of our APL Character Set poster which was displayed at the APL-Core booth at the National Computer Conference in Dallas.

This article has two parts. The first is a general description of our APL-Core project describing the results of our bench-mark tests in comparison to BASIC and to other APL's. The second article is to accompany the copy of the APL Character Set. It describes the method we used to implement the APL modified character set.

Any questions or comments concerning the APL-Core project may be addressed to me at the above address.

General correspondence to the club may be addressed:

Texas A&M Microcomputer Club
P.O. Box M9
Aggieland, TX 77844

One of the advantages of participating at the NCC Booth was to meet you, Carl Helmers, and others leading in the microcomputer revolution.

One of our disappointments was that we heard very little about other microcomputer projects across the nation. We heard of an interesting ZILOG APL in San Antonio, not yet executing code as ours, but perhaps further along in some portions than ours. Our emphasis, as contrasted to some microcomputer APL's, is to implement an APL which can be expanded to a full APL on 8-bit microcomputers. As the enclosed article points out, there is very little reason, except the many lines of code required, to have a restricted subset APL.

Yours truly,
Dale L. Schafer

The APL Committee of Texas A&M Microcomputer Club exhibited a running, but as yet incomplete, APL system at the National Computer Conference. The goal of the APL-Core project is to implement on a microcomputer the Core or central portion of the full APL language and to make it available so that it may be expanded to a full microcomputer APL system.

As an interim goal, to determine the feasibility of the overall goal, the Committee implemented and demonstrated at the NCC a working subset of APL. Only a few APL functions were implemented, but among the powerful features working and demonstrated were:

1) Names of any reasonable length (to 30 characters).
2) Compact data representation as: logical data stored as bits, character data as bytes, two byte integers, and even an index data type which provides extreme data compaction as compared to most APL implementations.
3) Automatic conversion between numeric data types, including logical to integer.
4) Multidimensional matrices (up to 15 allowed).

The APL-Core project members (Don Ader; Charles Bell; Dale L. Schafer, chairman; and Randolph Wilhoit) began the project with serious doubts concerning memory requirements and execution speed with the current generation of microcomputers, but they were determined not to take an approach which would compromise the APL language. The results dispelled any doubts by the committee about the desirability of microcomputer APL.

Tests of Execution Speed

Would you believe 1000% faster than BASIC and twice as fast as APL on an IBM 5100? In fact limited tests show results many times better, so the above may be conservative estimates of speed to be expected over a large cross-section of problems. Actual test results on a selected problem are shown in the table. Of course the AMDAHL is not likely to be a hobby computer, however connect time costs for a large computer must be added to the worth of execution time to calculate a cost/performance ratio. An additional cost for memory was allowed for APL in comparison to BASIC on the same machine.

SELECTED PROBLEM SPEED COMPARISON

Computer	Typical System Purchase Cost	Language/version	Execution time
MITS Altair 8800	$800-$3000	BASIC / MITS	500 seconds
MITS Altair 8800	$1000-$3200	APL / APL-Core	14 seconds
IBM 5100	$10,000-$20,000	APL / IBM APLSV	82 seconds
AMDAHL 470V/6	$4,000,000-$10,000,000	APL / IBM APLSV	0.22 seconds

Test Problem: Generate 1000 consecutive integers, then reverse the sign of the entire array 58 times, a total of 58000 sign reversals plus loop overhead.

Memory Requirements

The present version of APL-Core uses approximately 8K plus the memory allocated for user workspace. The requirement for a very powerful APL would appear to be 12K to 16K bytes. Because a problem written in BASIC requires approximately 5 to 10 times the bulk of an APL program, it seems likely that the overall memory requirement for larger problems may even have an advantage for APL.

In summary, the APL Committee began the project prepared to pay substantial penalties in memory requirement and execution speed to obtain the programming power of APL and were astounded to find instead of an execution time penalty an execution time bonus. We expected to of course run slower than the successful desktop IBM computer, but were willing to tolerate the delay to obtain APL power at 1/5 the cost; instead APL-Core runs faster at 1/5 the cost. We anticipated that the near compiler efficiency of APL might make it nearly as fast as the simple BASIC when using large operands, instead APL ended up in a different class, a magnitude faster or more.

APL-CORE NOT YET USEFUL

The only APL functions working to date are (1) Assignment, (2) Reshape, (3) Ravel, (4) Logical-not, (5) Index-generator, (6) Size-of, (7) Monadic minus, and (8) Monadic plus. All variable types are implemented with the exception of floating point, which is of course planned. The only working implementation to date is for 8080 micros. Several months of work are ahead before we will have a useful APL available, but our progress to date and the reception of our NCC exhibit have encouraged the committee to proceed with the work ahead. We no longer have any doubt that APL belongs in Microcomputers.

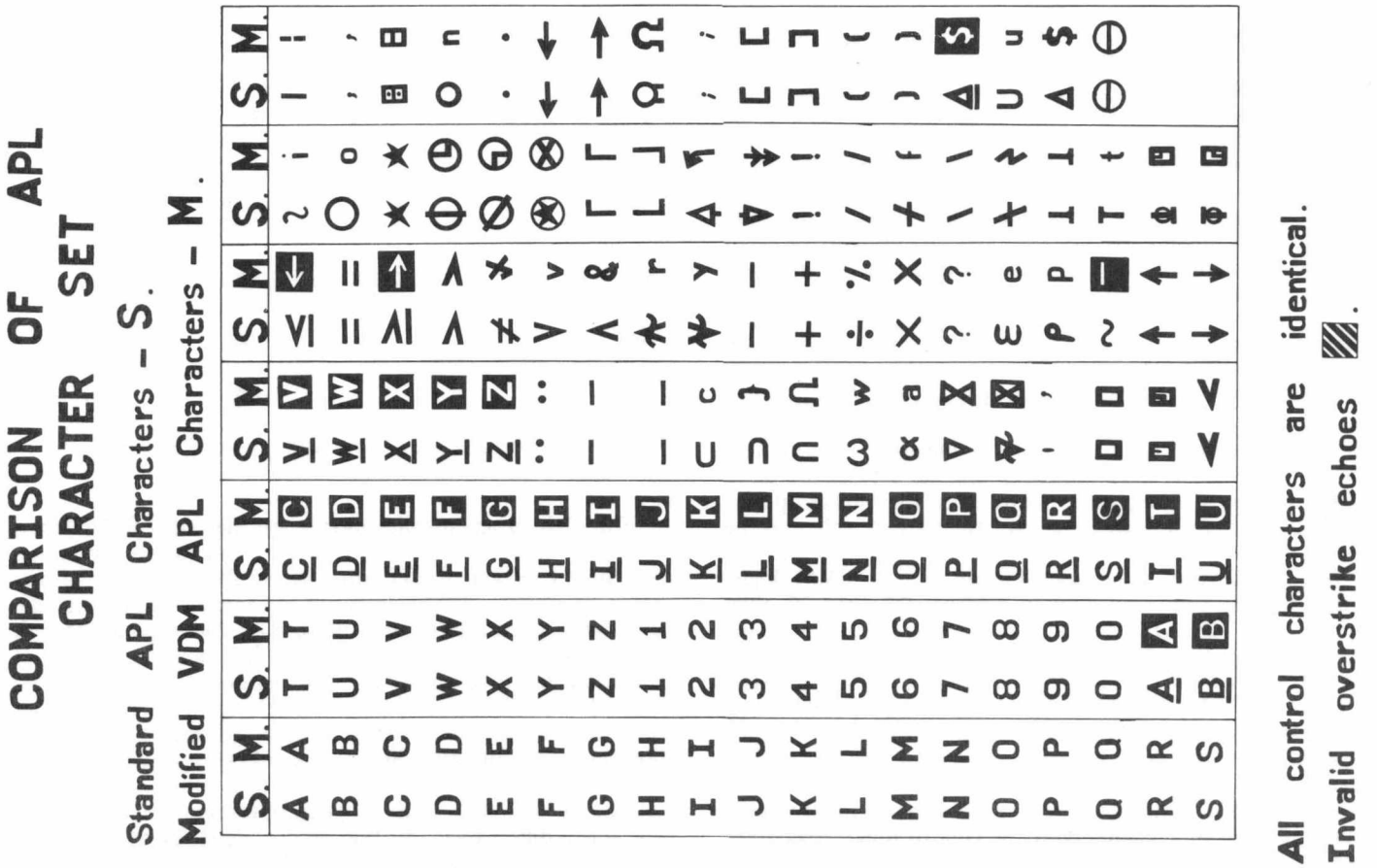
APL-CORE VIDEO DISPLAY OF APL CHARACTERS

An attraction of the APL-Core exhibit at the National Computer Conference (NCC) was a large poster showing a modified APL character set (using the VDM-I, an ASCII Video Display by Processor Technology) which had been prepared by Vanessa Holyoak for the APL Committee of the Texas A&M Microprocessor Club.

Dale Schafer, committee chairman, stated, "We had always thought that the essentially hardware problem of the display of APL characters was not our problem, and was surely being solved by someone else out there, but in the meantime Charles Bell of our committee worked up a backspace-overstrike program for the VDM which works in essence by table lookup, then display of the nearest approximation when a backspace-overstrike combination is entered on the keyboard." This display program allowed the APL Committee to use a video monitor for demonstrations at the NCC Personal Computing display booth without the purchase of any special output equipment for demonstrations of APL. Input to the APL program was made using an ASCII keyboard with labels stuck on the front of the keys, a conventional solution for those who have a keyboard without special APL character keytops.

As an example of the method, the APL logarithm symbol is normally input on an APL terminal as a circular (upper shift O), backspace, then an overstrike with the asterisk (upper shift P). With this same keyboard sequence, Charles's program would display the lower case "o" for the circular (although entered with an upper shift O), then when a backspace is followed by the upper case P, the VDM character which looks like a circle with a cross is substituted on the screen (as shown by the accompanying reduction of the NCC poster). The APL display program will also allow alternate methods which will result in display of the same character as, a reversed sequence (upper shift P, backspace, upper shift O) or, an O followed by several characters, then backspaces to the same character, then the overstruck upper shift P.

With the full 128 ASCII characters, plus the same characters with reversed cursor, close approximations could be found for the majority of APL characters (for example "i" used for iota). Some substitutions were only fair, and some could only become familiar after use over a period of time, however, for those hobbyists who cannot afford a special terminal for each language on his system, but has one of the video display boards with 128 ASCII characters, one more obstacle to the use of APL has been removed. In the meantime, the APL committee is working on APL-Core, an APL system for microcomputers but without the restrictions found on some microcomputer APL's.



DECODING 650X opcodes

H. T. Gordon,
Dept. of Etomology
University of California
Berkeley, CA. 94720

Dear Mr. Warren, *Aug. 2, 1977*

This is a write-up on 650X opcodes. The two programs in it are system-independent and so not right for KIM-1 USER NOTES (that doesn't like subroutines anyhow).

By the way, my earlier note on a STRINGOUT revision (that you published) had a few typos in the program. I've not bothered to send corrigenda because anyone who knows what he's doing will see them right away and few novices will read DDJ. In the programs I send herewith, with more complex logic, typos might give potential users a few headaches. I agree that direct reproduction of teletype output would be better, but some of us churchmice still don't have them.

Sincerely,
H. T. Gordon

Three recently-published programs (a debugger by Larry Fish in the Aug. '77 Kilobaud, and relocators by Ralph Sherman in the April '77 *DDJ* and by Jim Butterfield in the #4 '77 Kim-1 user notes) include routines for calculating the number of bytes required by a 650X opcode. All use quite different logic, and none is richly documented or coded as an independent subroutine. This decoding operation is a wheel that has probably been reinvented many times (I did it as an early programming exercise long ago, as a not-very-efficient 51-byte subroutine). The following table shows the intricacy of the problem. It lists the 16 opcode types from XØ to XF, roughly in order of usage frequency in programs (650X programmers try to avoid using 3-byte codes!). Decoding execution time will be shorter if common codes are the earliest decoded, when this is compatible with an efficient bit-sifting routine. Types XØ and X9 are unusual in that the number of bytes is determined by X (the term X_o means an even number and X_1 an odd number). Although the last 4 types are all illegal, coding errors may cause them; since they make up 60% of all illegal opcodes and are easy to sift out, this may be worth doing (but only the Butterfield program does it).

The Sherman program uses mostly (AND, CMP) logic. It sifts out all 1-byte opcodes in 4 steps: ØØ, then 2Ø, then (4,6)Ø, then X(8,A), then all 3-byte opcodes in 3 steps: X_1 (C,D,E,F), then X_1 (9,B), then X_o (C,D,E,F). Residuals are 2-byte opcodes. The Butterfield program uses a sequence of seven (AND,EOR) siftings in an indexed loop, addressing a 22-byte table of operands: first the illegals X(3,7,B,F), then 2Ø, then the X_1Ø branches, then (Ø,4,6)Ø and (Ø-7)8, then (8-F)8 and XA, then $X_1$9, then X(C,D,E). Residuals are 2-byte opcodes. Although ingenious and powerful, the program optimizes byte-economy at the cost of longer execution time.

Type	X	Bytes	# legal	# illegal
XØ	Ø, 4, 6	1	3	Ø
XØ*	X_1	2	8	Ø
XØ	X_o > 7	2	3	1
XØ	2	3	1	7
X8		1	16	Ø
XA		1	10	6
X1		2	16	Ø
X5		2	16	Ø
X6		2	16	Ø
X9	X_o	2	7	1
X9	X_1	3	8	Ø
X4		2	7	9
X2		2	1	15
XC		3	8	8
XD		3	16	Ø
XE		3	16	Ø
X3			Ø	16
X7			Ø	16
XB			Ø	16
XF			Ø	16

*includes all branch opcodes

The Fish program relies mostly on the 650X BIT instruction. Although suboptimally coded, it heightened my awareness of the power of BIT, not merely for detecting the presence or absence of single bits but (equally important) the simultaneous absence of 2 or more bits. The original program required 6 bit-masks in zero-page and had one error (that was corrected in a much more efficient revision sent to me by the author). I shall not analyze his bit-sifting operations, except to note that the very clever idea of splitting them into 2 branches (one for types X(Ø-7), the other for X(8-F)) was his. The following revision (further optimized and coded as a subroutine by me) saves both program bytes and execution time. The subroutine expects to find an opcode in the accumulator, and returns the correct number of bytes in the X register.

```
Ø21Ø  A2 Ø1     BYTNUM LDX #$Ø1 (sets 1-byte exit)
Ø212  2C 25 Ø2 BYTNOX BIT TRICK+4 (tests bit 3)
Ø215  DØ ØF            BNE HALFOP  (all X(8-F))
Ø217  2C 22 Ø2         BIT TRICK+1 (tests bits Ø-4, 7)
Ø21A  DØ 15            BNE 2BYTE   (all but (Ø,2,4,6)Ø)
Ø21C  C9 2Ø            CMP #$2Ø    (compare $2Ø)
Ø21E  FØ 1Ø            BEQ 3BYTE   (3-byte if =)
Ø22Ø  6Ø               RTS    (3 residuals 1-byte)
Ø221  A2 9F      TRICK LDX #$9F  (hidden data *)
Ø223  Ø5 14            ORA DUMMY      "    "
Ø225  Ø8               PHP            "    "
Ø226  2C 23 Ø2 HALFOP BIT TRICK+2 (tests bits Ø,2)
Ø229  FØ Ø7            BEQ 1BYTE   (all X(8,A))
Ø22B  2C 24 Ø2         BIT TRICK+3 (tests bits 2,4)
Ø22E  FØ Ø1            BEQ 2BYTE   (all X0(9,B))
            (3-byte residuals X1(9,B) and X(C,D,E,F))
Ø23Ø  E8       3BYTE INX
Ø231  E8       2BYTE INX
Ø232  6Ø       1BYTE RTS
```

*These are valid instructions that cannot be reached in program execution, but 4 of the 5 bytes serve as data operands for the BIT instructions, eliminating a data table. This trickery (suggested by a novel step in the Butterfield program, branching to a ØØ operand as a BRK instruction) would hopefully pass inspection by simple assemblers or debuggers!

The operation should be fast since neither of its 2 branches involves more than 3 bit-tests and 3 branchings; most of the common opcodes are decoded even faster. Of its 35 bytes, the 5 "trick" bytes serve to make it self-contained and functional in any 560X system. In any actual system, however, not all of them may be necessary, since most of them have large ROM programs that are a treasurehouse of bytes, at fixed addresses that make them usable as BIT masks. The subroutine would then become system-dependent; e.g., in a KIM-1 system there is an Ø8 at 1EB3 and a 14 at 1C95, so one could save 3 bytes by using only Ø5 9F in the TRICK sequence. If one can find all required mask bytes in ROM, the program will need only 30 bytes and become fully relocatable.

The main program can set the X register (e.g., to ØØ or FF) and bypass the BYTNUM setting by using a JSR BYTNOX. Operation affects only the X and status registers, e.g. the Z flag is set only by X(8,A) and is = bit 3 if both bits Ø and 2 are = Ø, while the V flag (unused by BYTNUM) is always = bit 6. The main program can add any or all of the special operations of the Sherman and Butterfield programs. The special handling of ØØ would be invoked by a BEQ after loading the opcode. Isolation of branch opcodes would be done after the return by 6 bytes: AND #$1F, CMP #$1Ø, BEQ BRANCH. I am less enthusiastic about the screening-out of 64 of the 104 illegals, and I have therefore developed an independent legality-testing subroutine.

There are some special problems in legality testing. E.g., early versions of the 650X lacked the ROR instruction and had only 147 legal opcodes instead of the 152 in the current version. There are 2 kinds of "illegals": many are interpreted as valid instructions and are executed by the 650X, while others seem to be blind alleys that halt further operations. E.g., "valid illegals" such as XF cause execution of *both* of the legals XD and XE, while "invalids" such as X2 (where X ≠ A) fail to execute. (I sent a note on this to *BYTE* long ago, that was accepted but has not yet been printed.) Also, there is added logical complexity in decoding the 6 types whose legality is determined by X, as shown in the following table of legal X values:

- X2 only X = A
- XØ all X except 8
- X9 all X except 8
- XA all X_0, plus 9 and B
- XC all X_0 (except Ø), plus B
- X4 all X_0 (except Ø,4,6), plus 9 and B

My attempts to program this using the BIT, that was so effective in BYTNUM, were so inefficient that I changed to a somewhat unusual logic, relying on a sequence of LSRs (that right-shift the opcode, lowest bit into the carry flag) to create extensive branch decisions. Like most first tries, the program must be suboptimal, especially since I have not had the advantage of seeing other legality programs (although the specs for the ECD MicroMind imply that such testing is done in their loading from tape cassettes).

The program assumes that an opcode is in the accumulator. It acts as a filter, causing a program break if the code is illegal. Although operation destroys the byte in the accumulator, it is preserved intact in the X register, so that it can be restored by a TXA in the main program after the return.

```
Ø24Ø  AA       OPLEGL TAX
Ø241  4A              LSR A  (bit Ø → carry)
Ø242  9Ø Ø9           BCC TYPEØ2  (all evens)
Ø244  4A              LSR A  (odds, bit 1 → carry)
Ø245  BØ 14           BCS ILLEGA  (all X(3,7,B,F)
Ø247  8A              TXA  (restore opcode)
Ø248  C9 89           CMP #$89 (compare to 89)
Ø24A  FØ ØF           BEQ ILLEGA (89 is illegal)
Ø24C  6Ø              RTS  (all other X(1,5,9,D))
Ø24D  4A       TYPEØ2 LSR A (evens, bit 1 → carry)
Ø24E  9Ø 17           BCC TYPEØ (all X(Ø,4,8,C))
Ø25Ø  4A              LSR A  (bit 2 → carry)
Ø251  9Ø Ø1           BCC TYPE2A  (all X(2,A))
Ø253  6Ø              RTS   (all X(6,E))
Ø254  4A       TYPE2A LSR A  (bit 3 → carry)
Ø255  BØ Ø5           BCS TYP4AC  (all X(A))
Ø257  C9 ØA           CMP #$ØA (tests for X = A)
Ø259  FØ Ø4           BEQ LEGALA  (A2 is legal)
Ø25B  ØØ       ILLEGA BRK  (other X2 illegal)
Ø25C  4A       TYP4AC LSR A  (bit 4 → carry)
Ø25D  BØ Ø1           BCS ODDX  (all odd X)
Ø25F  6Ø       LEGALA RTS  (residual even X)
Ø26Ø  29 Ø6    ODDX   AND #$Ø6 (tests X = 9,B)
Ø262  C9 Ø4           CMP #$Ø4  (must = Ø,)
Ø264  DØ 15           BNE ILLEG (illegal X1)
```

```
Ø25Ø  6Ø              RTS  (legals X = 9,B)
Ø251  4A     TYPEØ    LSR A  (bit 2 → carry)
Ø252  BØ Ø3           BCS TYPE4C  (all X(4,C))
Ø254  4A              LSR A  (bit 3 → carry)
Ø255  BØ Ø4           BCS LEGIT  (all X(8))
Ø257  C9 Ø8           CMP #$Ø8  (tests 8Ø)
Ø259  FØ ØA           BEQ NOTLEG  (8Ø is illegal)
Ø25B  6Ø     LEGIT    RTS  (all XØ legals)
Ø25C  4A     TYPE4C   LSR A  (bit 3 → carry)
Ø25D  FØ Ø6           BEQ NOTLEG  (Ø4, ØC illegals)
Ø25F  9Ø Ø5           BCC TYPE4  (other X4)
Ø261  C9 Ø9           CMP #$Ø9  (tests 9C)
Ø263  DØ E1           BNE TYP4AC  (residual XC)
Ø265  ØØ     NOTLEG   BRK  (9C is illegal)
Ø266  29 ØD  TYPE4    AND #$ØD  (tests 44, 64)
Ø268  C9 Ø4           CMP #$Ø4  (must = Ø4)
Ø26A  DØ DA           BNE TYP4AC  (residual X4)
Ø26C  ØØ              BRK  (44, 64 illegals)
```

When OPLEGL was tested (on a KIM-1, with a simple program that caused each BRK to display the illegal opcode for a few seconds) all 104 illegals were correctly identified. Nearly half of the 67 program bytes are required by X(4,A,C). Minor restructuring could save a few bytes, but I have not bothered because other programmers may now feel challenged to create a subroutine that will be both more byte- and time-efficient.

I have noted with regret the common tendency to bury complex logic inside special-purpose main programs instead of coding it as subroutines. This seems desirable to me *only* when it is vital to attain the absolute minimum execution time. The saving of 4 bytes needed by a JSR and RTS is a trivial gain. Even when its originator cannot conceive that a logic block could ever be useful in any other context (and who can be *certain* of that?), subroutining may offer greater structural flexibility, intelligibility, and ease of debugging and modification. Especially in ROMS (unalterable, but with a wonderful "always-there" character) rich internal subroutining can greatly increase the power of a system; KIM-1 users have exercised great ingenuity in accessing much of the programming in the 2K ROM, a task made more difficult by the failure of its designers to anticipate this. Furthermore, a microprocessor may be incorporated in many diverse systems (especially true of the 8080 and 650X chips), so that main programs are very often system-dependent. To the extent that they use system-independent subroutines, their adaptation to systems other than the one for which they were developed is facilitated.

Dear Dr. Warren, *August 5, 1977*

Enclosed is a one-page, one-paragraph addition to the MS I sent you a few days ago. It is an afterthought prompted by reading Stork's simulation program, in the issue of KILOBAUD I received after sending you my MS. Like Adam Osborne, I find instruction sets fascinating. They are where the real power resides. Although a primitive set, used by experts and provided with powerful auxiliaries, will outperform a superior design that lacks these enrichments, in the long run class will tell.

Sincerely,
H.T. Gordon

Like everyone else's, most of my main programs are system-dependent and involve routine operations. Whatever elegance there is must reside in the subroutines, codable in countless ways. Separately publishing these makes them available to any program in any 650X system, and may also focus attention on some elements of software design in all systems (in this instance, the advantages of branched- vs. linear-sequence sift/sort operations). Opcode decoding can be useful in non-650X systems; e.g., a debugging program-execution-simulator by Lee Stork in the Sept. '77 KILOBAUD has an opcode-byte-count routine in 8080 assembly language, using a linear sequence of 14 bit-tests (6 ANI and 8 CPI) and 14 jump-on-conditions. It is likely that this decoding existed previously, hidden in the mass of 8080 software. The absence of relative-branch instructions in the 8080 set seems strange to users of later designs of microprocessors (although I suppose 8080/Z80 users would feel handicapped by their limited range!). Still, minicomputers (and their micro copies) do without them, and the creation of a status register and a flock of jump-on-condition instructions was one of many brilliant innovations by Intel designers in the evolution of the 8008/8080 chip. One wonders what heights the Z80 might have reached, had these same designers not felt constrained to maintain software-compatibility with the 8080. When one sees how willing users are to rewrite logic blocks, instead of hunting for them in older software, the compatibility argument looks very weak! Although BASIC interpreters are not cheap, many versions exist for the 8080 and even for the 650X.

S-100 BUS COMPATIBLE MUSIC BOARD

news release *Received: 77 Jun 30*

Newtech Computer Systems' low-cost Model 6 Music Board enables anyone with an S-100 bus computer to produce music and sound effects. Applications include generating melodies, rhythms, sound effects, Morse code, touch-tone synthesis, and much more.

The Newtech Model 6 S-100 bus compatible Music Board comes fully assembled and tested. Its features include selectable output port address decoding, a latched 6-bit digital-to-analog converter, audio amplifier, speaker, volume control and RCA phono jack for convenient connection to your home audio system. It employs a glass epoxy printed circuit board with plated-through holes, gold-plated fingers and top quality components.

A complete Users Manual, supplied with the Model 6 Music Board, includes a BASIC language program for writing musical scores and an 8080 Assembly Language routine for playing them.

The price of the Model 6 Music Board is $59.95 through computer stores. Delivery is currently from stock.

For information contact your local computer store, or write to:

NEWTECH COMPUTER SYSTEMS, INC.
131 Joralemon Street
Brooklyn NY 11201
(212) 625-6220

A HIGH-SPEED MEMORY TEST PROGRAM FOR THE 6502

by Jim F. Butterfield

14 Brooklyn Avenue
Toronto Ontario
Canada M4M 2X5

Received: July 1, 1977

A recent algorithm by Knaizuk and Hartmann (IEEE Transactions on Computers, April 1977) outlines an ultra-fast RAM test.

The program below is based on this algorithm, but sacrifices testing efficiency to a small degree so as to achieve program compactness and simplicity. The test will detect any single stuck-at-1 or stuck-at-0 fault in a RAM, including the memory itself, the address and data lines, and the address decoders. Its run time is dramatically short compared to most memory test programs. The speed advantage becomes more pronounced as the amount of memory tested increases.

Each 'pass' of the test follows the following pattern:

1. Value FF is stored in every location to be tested.
2. Value 00 is stored in every *third* location, giving a pattern of FF FF 00 FF FF 00 . . .
3. Memory is checked for all values.

It is important to note that the above three steps must be done as three separate iterations.

The above pass is performed three times, with the position of the 00 value changed each time. Then the whole thing is repeated, exchanging the FF and 00 values.

The program given here is written for the KIM system; the indirect address pointer is positioned so that KIM will display it upon termination. For other systems, the pointer may be relocated and output as convenient. Subroutines and stack operations have been carefully avoided to allow the test to include page 1 of memory if desired. Memory is tested as a group of 'pages' rather than between any two arbitrary addresses; this is not essential but is usually convenient and helps the speed.

Address of the first and last pages to be tested should be placed in locations 0000 and 0001 respectively. The program starts at address 0002; it will halt showing a memory address on the display. This will be either the address of a fault, or (highest location tested + 1) for no fault.

```
MEMORY TEST   June, 1977   Jim Butterfield, 14 Brooklyn Ave.,
                             Toronto, Ontario Canada  M4M 2X5

0000 xx       BEGIN  xx     starting page for memory test
0001 xx       END    xx     ending page for memory test
0002 A9 00    START  LDA #0    zero pointers..
0004 A8              TAY        into low-order
0005 85 FA           STA POINTL   addresses
0007 85 70    BIGLP  STA FLAG     =00 first time, =FF second
0009 A2 02           LDX #2
000B 86 72           STX MOD      3 tests in each major loop
000D A5 00    PASS   LDA BEGIN    set pointer to..
000F 85 FB           STA POINTH   ..start of test area
0011 A6 01           LDX END
0013 A5 70           LDA FLAG
0015 49 FF           EOR #$FF     reverse FLAG
0017 85 71           STA FLIP      =FF first time, =00 second
0019 91 FA    CLEAR  STA (POINTL),Y    write above value..
001B C8              INY                ..into all locations
001C D0 FB           BNE CLEAR
001E E6 FB           INC POINTH
0020 E4 FB           CPX POINTH
0022 B0 F5           BCS CLEAR
              ; FLIP in all locations; now change 1 in 3
0024 A6 72           LDX MOD
0026 A5 00           LDA BEGIN    set pointer..
0028 85 FB           STA POINTH      ..back to start
002A A5 70    FILL   LDA FLAG     change value
002C CA       TOP    DEX
002D 10 04           BPL SKIP     skip 2 out of 3
002F A2 02           LDX #2       restore 3-counter
0031 91 FA           STA (POINTL),Y  change 1 out of 3
0033 C8       SKIP   INY
0034 D0 F6           BNE TOP
0036 E6 FB           INC POINTH   new page
0038 A5 01           LDA END
003A C5 FB           CMP POINTH   end of test area?
003C B0 EC           BCS FILL     no, keep going
              ; memory set up - now test it
003E A5 00           LDA BEGIN    set pointer..
0040 85 FB           STA POINTH    ..back to start
0042 A6 72           LDX MOD      synchronize 3-counter
0044 A5 71    POP    LDA FLIP     test for FLIP value..
0046 CA              DEX           ..? out of 3 times
0047 10 04           BPL SLIP           - else -
0049 A2 02           LDX #2       reset 3-counter
004B A5 70           LDA FLAG     & test for FLAG value
004D D1 FA    SLIP   CMP (POINTL),Y    make the test
004F D0 15           BNE OUT        branch if failed
0051 C8              INY
0052 D0 F0           BNE POP
0054 E6 FB           INC POINTH
0056 A5 01           LDA END
0058 C5 FB           CMP POINTH
005A B0 E8           BCS POP
              ; above test OK - change & repeat
005C C6 72           DEC MOD      change 1 in 3 position
005E 10 AD           BPL PASS     .. & do next third
0060 A5 70           LDA FLAG     invert flag..
0062 49 FF           EOR #$FF       ..for part 2
0064 30 A1           BMI BIGLP
0066 84 FA    OUT    STY POINTL   low order adds to display
0068 4C 4F 1C        JMP START      ..and exit to KIM
006B                 end
```

MEMORY TEST FOR D.G. Z-80

news release *Received: 77 Jul 25*

A new Digital Group Z-80 memory test has been announced by Steiner-Parker of Salt Lake City. The test will automatically find memory limits and the manufacturer claims that it is better than any other test currently on the market. Cost is $5.00 plus $1.50 for shipping and handling. Write Steiner-Parker, 2734 So. 2700 West, SLC, UT 84119.

CONFERENCE ON COMPUTING IN THE ARTS AND HUMANITIES

Friday evening-Sunday noon, October 21-23, 1977, Warren Weaver Hall, New York University. Sponsored by ACM/SIGLASH* and NYU Departments of Computer Science, Linguistics, Art and Art Education, Music and Music Education. Program will include:

PERFORMANCES

Friday evening: Computing in the Visual Arts--presentations of video and film works.

Saturday evening: Concert of music composed with the aid of computers.

Saturday afternoon: Computer Fashion show; Dance demonstration with computer-controlled lighting.

SESSIONS

Challenge sessions in music, art and language understanding: Computing in the arts and humanities--Is it a new medium, a tool or a distraction?

And:
Music tape and talk sessions
Computer vs. human poetry
Literary analysis
Personal Computing
Other demonstrations and discussions.

For registration information contact CONFERENCE CHAIRMAN: Dr. Naomi Sager, NYU Linguistic String Project, 251 Mercer Street, New York NY 10012. Tel: (212)598-2294, 5.

*Association for Computing Machinery/Special Interest Group on Language Analysis and Studies in Humanities.

MODS TO 8080 PILOT

```
DEAR DDJ:                                                    AUGUST 11, 1977

I HAVE MADE A FEW MODIFICATIONS TO DR. STARKWEATHER'S PILOT-8080
THESE CHANGES MAY (OR MAY NOT) BE OF INTEREST TO OTHER PILOT USERS.  HOWEVER
POOR TYPISTS LIKE ME WILL PROBABLY FIND THEM INVALUABLE!!  THE MODS
ENCLOSED ECHO DELETED CHARACTERS AND DISPLAY THE CURRENT STATE OF ANY INPUT
LINE IN RESPONSE TO A CTL/R.  THE NEW CODE FITS QUITE NICELY "ON TOP" OF THE
EXISTING CODE WITHOUT USING ANY OF THE BUFFER SPACE

--MODIFICATION TO ECHO DELETED CHARACTERS ON CONSOLE DEVICE

      LOCATION  OLD CODE  NEW CODE  COMMENTS
      --------  --------  --------  ----------------
       0CBD       2B        D7        RST 2  ;CHLL PATCH

       0D1D       2B        D7        RST 2  ;CALL PATCH

       0010       -         2B        DCX H  ;ORIGINAL CODE
       0011       -         4E        MOV C,M ;DELETED CHAR  TO REG C
       0012       -         C3 81,00  JMP CH01 ;JMP TO OUTPUT CHAR  AND RET

      NOTES
            1- USES EXTRA STACK BYTES
            2- DELETED CHARACTERS ALWAYS ECHOED TO CONSOLE
            3- NO SPECIAL REASON FOR CHOOSING "RST 2" EXCEPT IT
               MADE THE CHANGE COMPACT


--MODIFICATION TO REPEAT CURRENT LINE ON CONSOLE DEVICE IN RESPONSE TO CTL/R

      LOCATION  OLD CODE  NEW CODE  COMMENTS
      --------  --------  --------  ----------------
       0C7A       FE,7F     FE,12     CPI CTLR  ;CTL/R ?
       0C7C       CA,80,0C  CA,20,00  JZ  PATCH1  ;YES!

       0CF7       FE,7F     FE,12     CPI CTLR  ;CTL/R ?
       0CF9       CA,EF,0C  CA,2E,00  JZ  PATCH2 ;YES!

       0020       -         36,01     PATCH1  MVI M,1 ;MARK END OF BUFFER
       0022       -         11,30,01  LXI  H,EBUFF  ;SET UP H,L
       0025       -         CD,64,0C  CALL CRLF     ;NEW LINE ON CO
       0028       -         CD,49,00  CALL DSPLY    ;USE EXISTING DISPLAY SUB.
       002B       -         C3,6F,0C  JMP  KIN2     ;GO BACK TO NORMAL CODE


       002E       -         E5        PATCH2  PUSH H ;SAVE H,L
       002F       -         36,01     MVI  M,1      ;MARK END OF BUFFER
       0031       -         2A,22,01  LHLD LLSAV     ;SET UP H,L
       0034       -         CD,64,0C  CALL CRLF     ;NEWLINE ON CO
       0037       -         CD 49,00  CALL DSPLY    ;USE EXISTING DISPLAY SUB.
       003A       -         E1        POP  H        ;RESTORE H,L
       003B       -         36,20     MVI  M,20H    ;REMOVE END OF BUFFER MARK
       003D       -         C3,EF,0C  JMP  INPT2    ;GO BACK TO NORMAL CODE

      NOTES
            1- USES EXTRA STACK BYTES
            2- REPEATED LINE DISPLAYED TO CO DEVICE
            3- PILOT ALLOWS YOU TO DELETE MORE CHARACTERS THAN YOU INPUT
               WHEN LOADING A PROGRAM - - WATCH OUT!!
            4- IF YOU DELETE BACK TO A PRIOR LINE, THIS CTL/R SCHEME
               BREAKS DOWN

ALSO, DR STARKWEATHER'S FORESIGHT IN USING PILOT SOURCE STATEMENTS TO
INITIALIZE THE SYSTEM HAS ALLOWED ME TO CHANGE THE START-UP DIALOGUE TO
SELECT EITHER CONSOLE OR CASSETTE INPUT FOR NEW PROGRAM LOADING.  THE "TRICK"
I USED WAS TO CHANGE THE "EDIT" ENTRY IN THE "CTLST" OF CODES TABLE TO "TAPE".
THE NEXT TWO BYTES THEN WERE CHANGED TO POINT TO MY TAPE INPUT ROUTINES.
THE NEW INITIALIZATION SOURCE GOES LIKE THIS:

      T:
        :PILOT-8080  1.1
        :LOAD A NEW PROGRAM?
      A:
      M:Y
      JN:*%
      T:TAPE OR CONSOLE?
      A:
      M:TAPE
      JY:*%%
      T:ENTER PILOT PROGRAM
        'TERMINATE INPUT WITH CTL/Z
      LOAD:
      *% IEP
      *%% T START CASSETTE / PUSH ENTER
      A:
      TAPE:
      J:*%

SINCERELY,
STEVEN SPENCE                                    14205 N. SCOTT
                                                 EDMOND, OK  73034
```

PILOT-8080 FOOTNOTE

Dear Jim: 1977 May 14

I have two brief footnotes for the PILOT description in *Dr. Dobbs.* Please include them if you can.

1. The Development of PILOT-8080 was supported by a contract from the Lister Hill National Center for Biomedical Communications, National Library of Medicine, Bethesda, MD.

2. A paper tape of object code for PILOT-8080 can be obtained for $7.00 from Processor Technology Corp., 6200 Hollis St., Emeryville CA 94608. (415) 652-8080.

It may be possible to quote a lower price and I will let you know by phone it if turns out so.

Best Regards,
John A. Starkweather
Director
Computer Center
University of California
San Francisco CA
94143
(415) 666-4528

LIFE LINES

Professional Users Group
Dr. Dobb's Journal
P.O. Box E
Menlo Park CA 94025

Dear Professionals: 77-July 5

In the April issue of *DDJ,* Dr. Schenker mentions "Superlife" as an example of innovative fantasy and refers to a $100 prize. Well, that should stir up some interest. John Horton Conway is the originator of the Life game, as everyone knows. But the game has its roots in the much broader field of cellular automata, going back to Von Neumann and others. I don't have a bibliography handy, but a couple of titles come to mind:

CODD, E.F. *Cellular Automata.*
BURKS, A.W. *Essays on Cellular Automata.*

One might also keep in mind related works such as Golomb's classic, *Polyominoes,* and the many articles such as Read's "Contributions to the Cell Growth Problem," *Canadian Journal of Mathematics*, vol. 14, no. 1, pl-20 (1962).

Sincerely,
Jim Day
17042 Gunther Street
Granada Hills CA 91344

Z–80 PATCHES FOR MITS BASIC

Dr. Dobb's Journal of c.c.&o. Dec. 30, 1976

SUBJECT: Z–80 MITS BASIC PATCHES 12k extended – acr

It is amazing how quickly you can become tired of swapping CPU boards whenever there is a basic program to run. The following patches are the result of several days work studying disassembly listings (what the world needs is a good annotated disassembler). The problem with MITS Basic is the conditional parity operations; JPO JPE CPO CPE RPO RPE.

The 12k basic seems to be a mix of the conditional jumps and returns. Fortunately, the RPE and RPO instructions follow a RST 6, allowing a simple patch of the RST 6 and avoiding the use of the only remaining one byte call (RST 7). The three byte jmps are then patched one at a time with a common macro. This works well on my 12k basic, but i understand that there are several versions around and it would not be a bad idea to check the patch locations to see that they agree with your basic.

These patches were written and assembled on the TDL Relocating Assembler and the object can be loaded anywhere there is 0055H unused bytes. The assembler puts all address fields MSB first, so remember when you key in the patches, to swap the order of these fields.

Martin D. Gray

245 W. Grandview Ave.
Sierra Madre, CA 91024

```
'DL Z80 RELOCATING ASSEMBLER VERSION 1.0                        PAGE 1
MITS 12K EXTND BASIC (ACR)    Z-80 CPU MODS

                        .TITLE 'MITS 12K EXTND BASIC (ACR)    Z-80 CPU MODS'

                        ;MACRO DEFINITIONS
                        .DEFINE PATCH(ADRS,BRANCH,OP,INSERT,%EXPND,%SKIP)=[
                                .LOC    ADRS    ;;PATCH JPO,JPE HERE
                                JMP     %EXPND  ;;JUMP TO EXPANSION
                                .RELOC          ;;BACK TO NEXT SEQ LOC
                        %EXPND: PUSH    PSW     ;;SAVE STATUS
                                'INSERT         ;;STICK OP INSERT HERE
                                ANA     A       ;;SET ACCUM PARITY
                                'OP     %SKIP   ;;TEST PARITY AND BRANCH
                                POP     PSW
                                JMP     ADRS+3  ;;COND NOT MET
                        %SKIP:  POP     PSW
                                JMP     BRANCH  ;;COND IS MET]

                        ;FIX RST 6 HERE
0035                            .LOC    35H
0035  C3 0000'                  JMP     RST6
0000'                           .RELOC
0000  3D                RST6:  DCR     A       ;DECR ACCUM THRICE
0001' 3D                        DCR     A
0002' 3D                        DCR     A
0003' F5                        PUSH    PSW     ;PSW TO STACK FOR
0004' E3                        'XTHL           ;EXAMINATION IN REG L
0005  A7                        ANA     A       ;SET PARITY
0006' EA 000E'                  JPE     ..UP1
0009  CB95                      RES     2,L     ;RESET PARITY FLAG
000B  E3                        'XTHL           ;GOOD PSW BACK TO STACK
000C' F1                        POP     PSW     ;AND PSW IS FIXED
000D  C9                        RET             ;SO QUIT
000E' CBD5              ..UP1:  SET     2,L     ;SET PARITY FLAG
0010  E3                        'XTHL           ;PSW FIXED HERE ALSO
0011  F1                        POP     PSW     ;IN COMES THE GOOD PSW
0012  C9                        RET             ;AND BACK GOES THE PROGRAM

                        ;CONDITIONAL JMPS  'JPO AND JPE' AND THEIR ASSOC.
                        ;PARITY AFFECTING PREDECESSORS ARE FIXED HERE.
                        ;HERE COME THE Z-80 MACRO PATCHES
                                .RADIX  16
                                PATCH   0CEA,0CFD,JPO[
0CEA  C3 0013    +              JMP     ..0001
0013  F5         +..0001: PUSH  PSW
0014  A7        +              ANA     A
0015' E2 001C'   +              JPO     ..0002
0018  F1         +              POP     PSW
0019' C3 0CED    +              JMP     0CEA+3
001C' F1         +..0003:  POP   PSW
001D  C3 0CFD    +              JMP     0CFD  ]
                                PATCH   10BC,10C0,JPO[
10BC  C3 0020    +              JMP     ..0003
0020  F5         +..0003: PUSH  PSW
0021  A7        +              ANA     A
0022' E2 0029'   +              JPO     ..0004
0025  F1         +              POP     PSW
0026' C3 10BF    +              JMP     10BC+3
0029  F1         +..0004:  POP   PSW
002A  C3 10C0    +              JMP     10C0  ]
                                PATCH   229A,22F4,JPE,CUI     5[
229A  C3 002D    +              JMP     ..0005
002D  F5         +..0005: PUSH  PSW
002E  D605       +              CUI     5
0030' A7         +              ANA     A
0031' EA 0038'   +              JPE     ..0006
0034  F1         +              POP     PSW
0035  C3 229D    +              JMP     229A+3
0038' F1         +..0006:  POP   PSW
0039' C3 22F4    +              JMP     22F4  ]
                                PATCH   118C,1178,JPO[
118C  C3 003C'   +              JMP     ..0007
003C' F5         +..0007: PUSH  PSW
003D' A7         +              ANA     A
003E' E2 0045'   +              JPO     ..0008
0041' F1         +              POP     PSW
0042' C3 118F    +              JMP     118C+3
0045' F1         +..0008:  POP   PSW
0046' C3 1178    +              JMP     1178  ]
                                PATCH   1A82,905,JPE[
1A82  C3 0049'   +              JMP     ..0009
0049' F5         +..0009: PUSH  PSW
004A' A7         +              ANA     A
004B' EA 0052'   +              JPE     ..0010
004E' F1         +              POP     PSW
004F' C3 1A85    +              JMP     1A82+3
0052' F1         +..0010:  POP   PSW
0053' C3 0905    +              JMP     905  ]
                                .RADIX  10
                                .END
```

INTEGRAL IMPACT PRINTER FOR $745

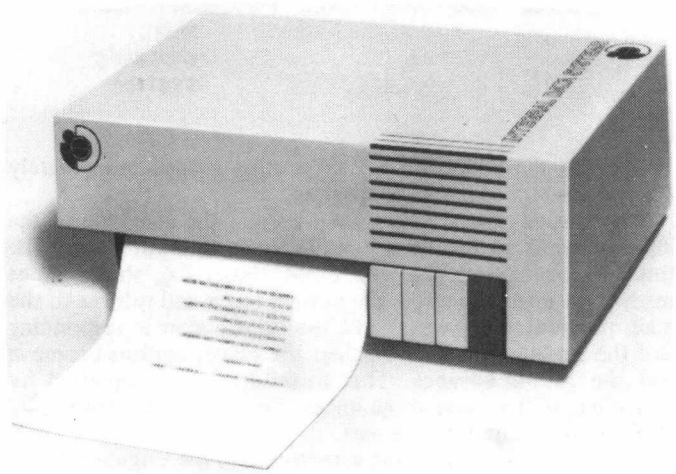
news release *Received: 77 Aug 31*

Integral Data Systems, Inc., Watertown, Mass., is now offering a new, full-performance, full-feature, dot matrix impact printer designed for use with mini or micro-computer systems. Printing at rates to 120 cps with up to 132 characters per line, the Integral Impact's standard features include an RS-232 and current loop serial interface, enhanced mode (double width) characters and selectable character and line sizes. Multiple copy capability on both fan-fold and roll paper is also a standard capability.

Housed in an attractive table top console with functional operator controls easily accessible on the front panel, the Integral Impact is a complete printer system ready to plug in and operate. Using the RS-232 serial interface, the printer can be integrated into any mini or micro-computer system by simply attaching it to a standard serial port. Serial baud rates of 110 to 1200 bits per second are selectable, and for those who prefer a parallel interface, that capability is also provided. Switch settings select character sizes and line length from 80 to 132 characters per line.

A 5 x 7 dot matrix is used to print the standard 64 character ASCII set. The print mechanism itself is very simple with few moving parts for added reliability, and incorporates a mechanism that automatically re-inks the ribbon to give an expected ribbon life of up to 10,000,000 characters. Line buffering by the microprocessor-based controller allows instantaneous print rates of more than 120 cps with sustained throughput of more than 75 cps possible.

Unit price for the Integral Impact is $745 with quantity discounts available. Delivery is 30-60 days ARO. For further information, contact: Integral Data Systems, Inc., 5 Bridge Street, Watertown, MA. 02172, (617) 926-1011.



REAL-TIME MODIFICATION FOR SANGER'S VIDEO CHASE

David J. Morgan
Assistant Professor of Geography
The University of Texas at Dallas
P.O. Box 688
Richardson, Texas 75080 August 4, 1977

Joseph Sanger gave us an exciting VDM Video Chase game in the May, 1977 *DDJ*. He mentioned planning a modification so that the robots would move toward you in real-time. That stimulated me to write the enclosed routine.

After mastering the "static" version of Video Chase I made the attached modifications for a "dynamic" system which automatically moves the robots at you if you don't move in the required time. My modification varies the required response time from about six seconds for a difficulty factor of zero (the original program runs at different difficulty levels) down to about three seconds for a difficulty factor of nine. You have to be pretty good at running the system to win at level nine.

The modification was quite simple. In Sanger's source listing there is only one place which calls the routine to input a move from the user. This occurs at location 015BH (line 1680) which has the symbolic label YMOV1. I changed the source as follows (changes shown in assembled code as well):

Change from:

```
015B CD A9 01  1680 YMOV1 CALL  INP   *GET USER MOVE
```

Change to:

```
015B CD F3 05  1680 YMOV1 CALL  INPX  *GET USER MOVE
                                       FOR
                                      *DYNAMIC
                                       SYSTEM
```

This sends the program to a subroutine loaded immediately beyond the end of Sanger's program.

My routine puts a timing loop around the checking of the input device's status. If a normal move is input before the timing loop expires (i.e. the input status flag shows input ready) the routine accepts the normal input and returns to the main program. If, however, the user is too slow in responding and the timing loop expires, then the player forfeits his move and the robots advance. This forfeiture is accomplished by returning to the main program as though an 'H' move (stay in place) were input by the user.

There appears to be a bug somewhere in the original version that I haven't been able to find. After about three games (sets of nine series of kills) the program blows up. If anyone else has solved the problem I would like to see a patch. I haven't been able to find any problems in my entry of the source code but I did relocate the origin above my operating system. It did appear to be relocatable (since the VDM is in the standard location and uses the standard I/O port).

Also, why did you reprint Raskin's report on the DZ80-80 CPU? I though I read that a while back.

I am busily typing in PILOT on my IMSAI. I have the iCOM disk system and am quite pleased with it. The power supply they sell (the ADTECH) is expensive and runs hot but the software is reliable and has served me pretty well.

ONE-FOR-ONE PATCH TO P.T. 5K BASIC

Received; 77-Aug 10

Dear Jim,

Dr. Dobb's (and others) have proposed numerous patches (and patch repairs) for the integer routine of the Processor Technology 5-K Basic. Some of them have worked. In Tucson there is Howard H. Heid, a very brilliant young fellow, who deserves a big applause and congratulations for a true fix -- not a patch, but a FIX -- for this problem. Howard has been literally 'possessed' by this problem (in my machine) and the patches presented by *Dr. Dobb's*, but 'gotta-do-it-better' Howard was not satisfied until the repair was made (get this) in THE ORIGINAL NUMBER OF BYTES.

The solution is so simple we couldn't see the trees (or something of that sort): return the routine to its original form, then change TWO BYTES --

OBB2	change C9 to	3D	(DCR A)
OBB5	change D0 to	FC	(RP)

The routine now treats any fractional value (less than 1) in exactly the same way it handles the fractional portion of any other value (greater than 1). The top part of the routine is given below, with the proper changes.

You have a beautiful paper. . . .thanks.

Morris G. Miller
Box 11496
Tucson AZ 85734

Fix for Integer Routine for Processor Technology 5-K BASIC

```
0BBA  0A          Aint   LDAX B
0BAB  D6 81              SUI 129
0BAD  F2 B3 0B           JP AINT1
0BB0  AF                 XRA A
0BB1  02                 STAX B
0BB2  3D                 DCR A
0BB3  DC 05       Aint1  SUI FPNIB-1
0BB5  F0                 RP
0BB6  57                 M0V D,A
0BB7  0B                 DCX B
0BB8  0B          Aint2  DCX B
                         .
                         .
                         .
                         ETC.
```

Morris G. Miller
Box 11496
Tucson AZ 85734

ISAACSON TO BE PC EDITOR FOR *DATAMATION*?

We hear the rumor that Dr. Portia Isaacson, the recent chief wheel for the Dallas National Computer Conference – and prime mover behind NCC'77's having a major program and exhibit activity concerning home and hobby computing – will become the Contributing Editor for Personal Computing for *Datamation*. It looks like the #1 data processing mag has finally discovered that there's lots happening in our little corner of the computing world (or, they have discovered that many advertising bucks are forthcoming from our corner).

Go get 'em, Portia.

VIDEO CHASE CONVERSION FOR A POLY VTI

Paul Holliday
4807 Arlene Street
San Diego, CA. 92117
(714) 225-6209

Received: 77-Aug-18

This is a copy of the patches needed to convert the game of VIDEO CHASE for 8080/VDM (which appeared in the May 1977 issue of Dr Dobb's) for use with the Polymorphics Video Terminal Interface (VTI) board.

I would like to make note, especially for the novice, that entering the original machine code and then the patches may turn out to be quite a tedious chore! This of course is the opinion of a novice (myself) and is based upon my very first experience with entering raw code on my Vector 1 computer just newly "Up and Running".

Frankly, I was not prepared for the amount of time that it took me to enter the code and then modify it to fit my system. For example, it took me about three hours to enter the original code and an additional six hours to patch and de-bug. This was done with the aid of the Vector 1 monitor program and using a keyboard/hex entry. I would hate to have to enter the code via front panel bit switches!

In any case, a distinct advantage was gained by using the hex dump which was so neatly provided at the end of the source code. The hex dump really speeds up entry of raw data and should be provided as part of any of the programs that you publish. The one disadvantage that I found with the hex dump entry was that I had a heck of a time staying awake! It gets pretty boring punching in numbers after thirty or forty minutes. My suggestion is to enter it in stages, perhaps even spreading it out over a few days.

A few other tips may prove handy. Enter (or code) the program exactly as it appears in the dump (or listing) the first time through, and then immediately write (or save) the original on tape. Do this even before you "try it out" for the first time. If you do this, you will always have an original, unmodified "Baseline" version (even if it has a few keypunch errors). This provides a convenient way to go back to "go" if (or should I say when?) something happens during modifications or when entering the patches. What I usually do is keep the Baseline on one side of the tape and the patched or modified versions on the other. It is generally best to keep about three iterations back if you plan on making changes in small, incremental steps.

Some notes about the patch tables enclosed. All of the patches are oriented towards my Vector 1 system with the Poly VTI board. If you have such a system or combination, all of the patches in Table 1 must be done. I have described my hardware configuration in Table 5. The patches in all the Tables are in sequential (core) order for ease of entry.

Table 1 also makes repeated reference to 'Note 1' which describes the need for bit 7 in VTI ASCII codes. However, some assemblers may not accept the source code value 'X'+80H as a valid input. If this is the case with your assembler, the exact hex value could be used instead. For example, 'X'+80H evaluates as D8H, which is the 8 bit ASCII value for the character 'X'. Actually, if you have an assembler, a better way would be to leave the 7 bit ASCII value as is and call a subroutine to insert bit 8 for the VTI. (You can use ORI 80H).

Tables 2, 3, and 4 are merely excerpts from Table 1, and are previded for convenience in changing or experimenting with various hardware/software configurations.

In particular, Table 2 lists only those patches required for video terminal and keyboard input/output. This table can be used to change only the I/O routines and is useful for those systems where "everything else fits but the I/O."

Table 3 shows the patch areas for the robots symbol only. I didn't make any changes for my configuration since the graphic symbol produced with 07H was acceptable. In fact, I preferred the white 'blob' to the somewhat nondescript symbol used in the original version. You can use this Table to experiment with different symbols and produce 'robots' to your liking.

As a final note, I would like to mention that I agree with and heartily approve of DDJ's general philosophy. That is, ample distribution of software written by non-professionals for personal use. We all gain by communication and distribution of ideas and tools. All we have to do now is convince the hardware manufacturers to standardize and help stop the proliferation of diverse systems. I have not as yet run across two systems (hobbyist) that were exactly alike!

Keep up the good work and keep Jef Raskin turned on.

○ ○ ○ ○ ○ ○

TABLE 1. Patches To Convert Video Chase V2.1 for VDM FOR USE WITH POLY VTI BOARD (ALL PATCHES, SEE ALSO INDIVIDUAL PATCHES)

LOC (HEX)	WAS	NEW	SOURCE WAS		NEW SOURCE	
				ENTER	IN START (SEE TBL 2)	
0008	01	D1	ENTER	IN DATA	LXI H, D1H (VTI)	
0014	CD	D1		LXI H, MS2	LXI H, D3H (VTI)	
001D	CF	D3	DIFF	LXI H, MS1	MVI M, 'X'+80H	(NOTE 1)
003C	58	D8		MVI M, ' ' (NOTE 1)	MVI M, ' '+80H	(NOTE 1)
0042	20	A0		MVI M, 'X' (NOTE 1)	MVI M, 'X'+80H	(NOTE 1)
0051	58	D8		MVI M, ' ' (NOTE 1)	MVI M, ' '+80H	(NOTE 1)
005D	20	A0		MVI M, 'X' (NOTE 1)	MVI M, 'X'+80H	(NOTE 1)
0063	58	D8		MVI M, 'X' (NOTE 1)	MVI M, 'X'+80H	(NOTE 1)
006F	58	D8		MVI M, 'X' (NOTE 1)	MVI M, 'X'+80H	(NOTE 1)
0097	58	D8		MVI M, 'X' (NOTE 1)	MVI M, 'X'+80H	(NOTE 1)
009E	CC	D0	ROSE	LXI H, ROBN	MVI M, D0F6H (TBL2)	
0010	54	D4		MVI M, 'T' (NOTE 1)	MVI M, 'T'+80H	
00A3	CC	D0		LXI H, ROBN+4	LXI H, D0FAH	
00A5	59	D9		MVI M, 'Y' (NOTE 1)	MVI M, 'Y'+80H	
00A8	CC	D0		LXI H, ROBN+8	LXI H, D0FEH	
00AA	55	D5		MVI M, 'U' (NOTE 1)	MVI M, 'U'+80H	
00AD	CD	D1		MVI M, ROBN+42H	MVI M, D138H	
00AF	5C	DC		MVI M, SCH (\)	MVI M, DCH (\)	
00B2	CD	D1		LXI H, ROBN+44H	LXI H, D13AH	
00B4	5E	DE		MVI M, 5EH (^)	MVI M, DEH (^)	
00B7	CD	D1		LXI H, ROBN+46H	LXI H, D13CH	
00B9	2F	AF		MVI M, 2FH (/)	MVI M, AFH (/)	
00BC	CD	D1		LXI H, ROBN+80H	LXI H, D176H	
00BE	47	C7		MVI M, 'G' (NOTE 1)	MVI M, 'G'+80H	
00C1	CD	DL		LXI H, ROBN+82H	LXI H, D178H	
00C3	0D	AD		MVI M, ODH (←)	MVI M, ADH (←)	
00C6	CD	D1		LXI H, ROBN+84H	LXI H, D17AH	
00C8	48	C8		MVI M, 'H' (NOTE 1)	MVI M, 'H'+80H	
00CB	CD	D1		LXI H, ROBN+86H	LXI H, D17CH	
00CD	09	AD	*	MVI M, 09H *	MVI M, ADH (←)	
00D0	CD	D1		LXI H, ROBN+88H	LXI H, D17EH	
00D2	4A	CA		MVI M, 'J' (NOTE 1)	MVI M, 'J'+80H	
00D5	CD	DL		LXI H, ROBN+C2H	LXI H, D188H	
00D7	2F	AF		MVI M, 2FH (/)	MVI M, AFH (/)	
00DA	CD	D1		LXI H, ROBN+C4H	LXI H, D18AH	
00DC	08	A1		MVI M, OBH (!)	MVI M, A1H (!)	
00DF	CD	D1		LXI H, ROBN+C6H	LXI H, D18CH	
00E1	5C	DC		MVI H, SCH	MVI H, DCH	
00E4	CD	D1		LXI H, ROBN+100H	LXI H, DLF6H	
00E6	42	C2		MVI M, 'B' (NOTE 1)	MVI M, 'B'+80H	

* THIS IS AN ERROR IN ORIGINAL CODE, SB '-' OR 2DH. (cont)

NOTE 1. THE VTI MUST HAVE BIT 7=1 FOR ASCII, THUS ALL DIRECT MOVES TO VTI SCREEN MUST HAVE 'A'+80H (OR USE 8 BIT ASCII)

TABLE 1. (CONTINUED)

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE
00E9	CD	D1	LXI H, RCORN+104H	LXI H, D1FAH
00EB	4E	CE	MVI M, 'N' (NOTE 1)	MVI M, 'N'+80H
00EE	CD	D1	LXI H, RCORN+108H	LXI H, D1FEH
00F0	4D	CD	MVI M, 'M' (NOTE 1)	MVI M, 'M'+80H
012C	20	A0	CPI ' ' (NOTE 1)	CPI ' '+80H
014E	20	A0	CPI ' ' (NOTE 1)	CPI ' '+80H
0153	0F	87	MVI M, 0FH (YOU)	MVI M, 87H (YOU)
0162	20	A0	MVI M, ' ' (NOTE 1)	MVI M, ' '+80H
0197	20	A0	CPI ' ' (NOTE 1)	CPI ' '+80H
019C	0F	87	MVI M, 0FH (YOU)	MVI M, 87H (YOU)
01AA	00	D1	INP IN STAT	INP IN STAT (or D1H)
01AC	40	01	ANI RDA	ANI RDA (or 01H)
01AD	CA	C2	JZ INP	JNZ INP
01B1	01	D0	IN DATA	IN DATA (or D0H)
020B	58	D8	CPI 'X' (NOTE 1)	CPI 'X'+80H
0212	0F	87	CPI 0FH (YOU)	CPI 87H (YOU)
0217	58	D8	CPI 'X' (NOTE 1)	CPI 'X'+80H
022A	04	00	MVI M, 4	MVI M, 0 (FULL BRIGHT)
022B	84	7F	MVI M, 84H	MVI M, 7F (FULL DARK)
0235	58	D8	MVI M, 'X'	MVI M, 'X'+80H
0239	20	A0	MVI M, ' ' (NOTE 1)	MVI M, ' '+80H
0286	0C	A3	DEATH MVI M, 0CH (NOTE 1)	MVI M, 'C'+80H
02F8	00	D1	WAIT IN STAT	WAIT IN STAT (or D1H)
02FA	40	01	ANI RDA	ANI RDA (or 01H)
02FB	CA	C2	JZ WAIT	JNZ WAIT
02FF	01	D0	IN DATA	IN DATA (or D0H)
0317	CD	D1	LXI H, ENDM1	LXI H, D110H
0322	30	B0	ADI 30H	ADI 30H+80H
0325	CD	D1	STA 0CD27H	STA D127H
032A	30	B0	ADI 30H	ADI 30H+80H
032D	CD	D1	STA 0CD29H	STA D129H
0339	CD	D1	LXI H, ENDM1	LXI H, D110H
0344	30	B0	ADI 30H	ADI 30H+80H
0347	CD	D1	STA 0CD27H	STA CD27H
034C	30	B0	ADI 30H	ADI 30H+80H
034F	CD	D1	STA 0CD29H	STA D129H
0352	CD	D1	END2 LXI H, ENDM3	END2 LXI H, D190H
0389	CC	D0	PLOT LXI H, 0CC00H	LXI H, D000H
0450	CC	D0	CLEAR LXI H, 0CC00H	LXI H, D000H
0452	20	A0	MVI M, ' ' (NOTE 1)	LXI H, ' '+80H
0456	D0	D4	CPI 0D0H	CPI D4H (NOTE 2)
045C	CB	CB	OUT CBH (NO CHANGE FOR VTI)	OUT CBH

(cont.)

NOTE 2. BOTTOM OF VTI SCREEN = D000 + 1024 = D400

TABLE 1 (CONTINUED)

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE
0462	CF	D3	LXI H, MS1	LXI H, D3CCH
046E	CF	D3	MSG2 LXI H, MS1	MSG2 LXI H, D3CCH
0478	CF	D3	WIN LXI H, MS1	WIN LXI H, D3CCH
0494	CF	D3	SCORE LXI H, SRE	SCORE LXI H, D3CCH
049F	30	B0	ADI 30H	ADI 30H+80H
04A2	CF	D3	LXI H, SCR	LXI H, D3BDH
04A8	30	B0	ADI 30H	ADI 30H+80H
04B0	CF	D3	LXI H, MSY	LXI H, D3B8H
04B9	CF	D3	LXI H, MT1	LXI H, D3F5H
04C5	CF	D3	LXI H, MS1	LXI H, D3CCH
04C9	20	A0	MCL1 MVI M, ' '	MCL1 MVI M, ' '+80H
04D2	0A	C3	MSG0 LDAX	MSG0 JMP MSGQ
04D3	FE	DC	CPI 0FFH	
04D4	FF	04	" "	

NOTE: THE JUMP AT MSG0 JUST BYPASSES ALL CALLS TO MSG0 AND DIVERTS THEM TO MSGQ WHICH IS IDENTICAL EXCEPT FOR THE "ORI 80H". THIS SETS BIT 7 = 1 WHICH IS REQUIRED FOR THE VTI TO OUTPUT AN ASCII CHARACTER.

○ ○ ○ ○ ○

TABLE 2. Patches for Keyboard IN/out only

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE AND/OR COMMENTS
0008	01	D1	IN DATA	IN STAT (NOTE 3)
01AA	00	D1	INP IN STAT	INP IN STAT (OR D1H)
01AC	40	01	ANI RDA	ANI RDA (OR 01H)
01AD	CA	C2	JZ INP	JNZ INP
01B1	01	D0	IN DATA	IN DATA (OR D0H)
02F8	00	D1	WAIT IN STAT	WAIT IN STAT (OR D1H)
02FA	40	01	ANI RDA	ANI RDA (OR 01H)
02FB	CA	C2	JZ WAIT	JNZ WAIT
02FF	01	D0	IN DATA	IN DATA (OR D0H)

(CONT.)

TABLE 2. (CONTINUED)

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE AND/OR COMMENTS
				DATA EQU D0H RDA EQU 01H STAT EQU D1H NOTE 3. THIS IS NOT REQUIRED FOR THE VTI BUT I CHANGED IT TO IN D1 FOR CORRECTNESS. IT DOESNT MATTER IF IT IS D0 OR D1 BUT BEST IF LEFT AT D1 OR NOP.

○ ○ ○ ○ ○ ○ ○ ○

TABLE 3. PATCHES FOR ROBOTS symbol only

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE
0131	07	NOT CHANGED	MVI M,7	I DIDNT CHANGE THIS BECAUSE THE VALUE 07H WILL PRODUCE THE [illegible] SYMBOL:  dark WHICH IS ACCEPTABLE AS THE ROBOT TARGET
021C	07		"	
02B9	07		"	
02BE	07		"	
02C3	07		"	
03EB	07		CPI 7	
03F2	07		"	
03F9	07		"	
0400	07		"	
0407	07		"	
0445	07		"	
ROBOTS DEATH DANCE ONLY:				
0221	04	00	MVI M,4	MVI M,00H FULL BRITE
022B	84	7F	MVI M,84H	MVI M,7FH FULL DARK

TABLE 4. Patches for YOU Symbol only

LOC (HEX)	WAS	NEW	SOURCE WAS	NEW SOURCE
0153	0F	87	MVI M,0FH	MVI M,'Θ' (or 87H)
019C	0F	87	YI MVI M,0FH	MVI M,'Θ' (or 87H)
0212	0F	87	CPI 0FH	CPI 'Θ' (or 87H) NOTE: THE VALUE 'Θ' IS THE theta SYMBOL ON MY CHARACTER GENERATOR. CAN USE 87H IF 'Θ' SYMBOL NOT VALID WITH YOUR ASSEMBLER.

○ ○ ○ ○ ○ ○ ○ ○ ○ ○

TABLE 5. SYSTEM DESCRIPTION (Paul Holliday)

SYSTEM DESCRIPTION (as of 10 Aug '77)		
HARDWARE:		
• Mainframe	Vector 1	
• CPU.	Vector 1 8080 CPU board	
• Keyboard	KB-6 (connected to VTI) (Stat bit 0=low keypres)	STATUS IN: D1* RDA(status) 01 DATA IN: D0
• TV monitor . . .	Hitachi P-05 W/Pickles & Trout Mod.	
• Cassette 1	Tarbell, 1500 bps	STATUS IN: 6E DATA IN/0: 6F
• I/O	IMSAI MIO board, cassette 2 .	CRI DATA: 40 CONTROL: 43 PIO: 41 SIO: 42
• Video interface .	Polymorphics VTI, Rev 1.2,	ADDRESS: D000
• Memory	#1=Solid State Music MB-6A 8k,	0000-1FFF
• PROM/RAM. . .	Vector 1 reset & go w/1k RAM	C000-CFFF
SOFTWARE:		
• Vector 1 monitor on Prom/ram (V1.2P) for VTI		C000-C700
*Addresses and I/O channels in hex notation.		

AN 8080 TRACER

Dear Jim:

My ambition since getting involved in personal computing two years ago has been to produce a software system that provides the user of a small system with the kind of power available on a large-scale timesharing system. What with the demands on time imposed by the necessity of making a living, progress has been slow, but at last I seem to be getting somewhere. I have basically settled on a design synthesized from MULTICS (the universal segment notion), UNIX (separation of system nucleus and privileged utilities, plus driver philosophy and the "pipe" concept), and the internal protocol from a radially-multiplexed data network I have developed at work.

The main reason for this letter, however, is unrelated to all this neat stuff. Along the road to this system, I have developed an 8080 debug monitor (we must crawl before we run) which includes a complete interpretive trace with software breakpoints, trace within range for debugging subroutines and the ability to examine or change any machine register or memory location. The memory dump command permits memory to be examined in Hex, ASCII, or 8080 instruction format. The output of the trace is very easy to read (sample attached). I haven't researched very thoroughly, but I don't know of an 8080 trace in the public domain, so if this is the first, or it seems of interest, I would be willing to prepare it for publication in your journal.**[Do! Do! -Ed.]** At present, the trace is integrated in my 4K monitor, which includes lots of other commands not of general interest. Since the monitor listing runs to some 61 pages, it's hardly practical to publish it all and let people extract the trace. If you want to publish the trace, I'll do the work to extract it and make a nice trace-only package, but I would first like to know if you're interested, since I have no need myself for such a package and otherwise won't spend the week or so of spare time to dig it loose and clean it up for publication. **[Yes - We *are* interested . . . even if it has taken 3 months to reply to your kind offer. - J.W.]**

Other than that, I now have a home-brew PL/M compiler running on a handy local timesharing system, and I'm presently extending it to support general data structures. I intend to write the operating system entirely in the extended PL/M to make it as machine-independent as possible.

Sincerely,
John Walker
Marinchip Systems
16 St. Jude Road
Mill Valley CA 94941

P.S. My guess as to the size of the extracted trace routine is something less than 2K.

Sample dump in instruction format

```
  X
.i 1000,a
1000  LXI  SP,2000
1003  MVI  A,08
1005  INR  A
1006  CALL 1100
1009  JMP  1005
.i 1100:110f
1100  PUSH PSW
1101  DAA
1102  PUSH PSW
1103  ANI  0F
1105  MOV  C,A
1106  POP  PSW
1107  ANI  F0
1109  RAR
110A  RAR
110B  RAR
110C  RAR
110D  MOV  B,A
110E  POP  PSW
110F  RET
```

John Walker
Marinchip Systems
16 St. Jude Road
Mill Valley, CA 94941
4/28/77

Sample register examine / change and Full instruction trace

```
  X
.rg
A=13 B=01 C=02 D=F7 E=9F H=3F L=B5 F=12 P=1102 SP=1FFC
.rg b=fc,c=de,
A=13 B=FC C=DE D=F7 E=9F H=3F L=B5 F=12 P=1102 SP=1FFC
.ton
.g 1000
1000  LXI  SP,2000 SP=2000
1003  MVI  A,08    A=08
1005  INR  A       A=09 F=06
1006  CALL 1100    SP=1FFE [1FFF]=10 [1FFE]=09
1100  PUSH PSW     SP=1FFC [1FFD]=09 [1FFC]=06
1101  DAA
1102  PUSH PSW     SP=1FFA [1FFB]=09 [1FFA]=06
1103  ANI  0F      F=16
1105  MOV  C,A     C=09
1106  POP  PSW     F=06 SP=1FFC [1FFB]=09 [1FFA]=06
1107  ANI  F0      A=00 F=56
1109  RAR
110A  RAR
110B  RAR
110C  RAR
110D  MOV  B,A     B=00
110E  POP  PSW     A=09 F=06 SP=1FFE [1FFD]=09 [1FFC]=06
110F  RET          SP=2000 [1FFF]=10 [1FFE]=09
1009  JMP  1005
1005  INR  A       A=0A
1006  CALL 1100    SP=1FFE [1FFF]=10 [1FFE]=09
1100  PUSH PSW     SP=1FFC [1FFD]=0A [1FFC]=06
1101  DAA          A=10 F=12
1102  PUSH PSW     SP=1FFA [1FFB]=10 [1FFA]=12
1103  ANI  0F      A=00 F=56
1105  MOV  C,A     C=00
1106  POP  PSW     A=10 F=12 SP=1FFC [1FFB]=10 [1FFA]=12
1107  ANI  F0      F=02
1109  RAR          A=08
110A  RAR          A=04
110B  RAR          A=02
110C  RAR          A=01
110D  MOV  B,A     B=01
110E  POP  PSW     A=0A F=06 SP=1FFE [1FFD]=0A [1FFC]=06
110F  RET          SP=2000 [1FFF]=10 [1FFE]=09
1009  JMP  1005
1005  INR  A       A=0B F=02
1006  CALL 1100    SP=1FFE [1FFF]=10 [1FFE]=09
1100  PUSH PSW     SP=1FFC [1FFD]=0B [1FFC]=02
1101  DAA          A=11 F=16
1102  PUSH PSW     SP=1FFA [1FFB]=11 [1FFA]=16
1103  ANI  0F      A=01 F=12
1105  MOV  C,A     C=01
1106  POP  PSW     A=11 F=16 SP=1FFC [1FFB]=11 [1FFA]=16
1107  ANI  F0      A=10 F=02
1109  RAR          A=08
110A  RAR          A=04
110B  RAR          A=02
110C  RAR          A=01
110D  MOV  B,A
110E  POP  PSW     A=0B SP=1FFE [1FFD]=0B [1FFC]=02
110F  RET          SP=2000 [1FFF]=10 [1FFE]=09
1009  JMP  1005
1005  INR  A       A=0C F=06
```

1802 HACKERS! UNITE

Dear Dr., August 9, 1977

This letter is directed to all of the readership who have built and now program with the RCA CDP-1802CD uP.

'Where is all of the software/hardware correspondence?'

Dr. Dobb, please print this letter so that we in the minority don't self-extinct. The RCA CDP-1802CD is an excellent uP.

To the readers:

Send all initial correspondence to:

National Clearinghouse CDP-1802CD
920 Terrace Drive
Glenwood, Illinois 60425

Yours very truly,

Phil Gennarelli

FRIEND FINDS FACTORIAL FLAW

Jef Raskin

It is always a pleasure to get a helpful letter from a careful reader. Mr. William T. Mitchell, 5805 Farnham Court, Hanover Park IL 60103 wrote that the example of a recursive factorial function in my article on MIT's BASIC, Polymorphic System's BASIC and NIBL in the April issue of *DDJ* was a good swiss cheese emulator. That is, it was full of holes. It didn't flag negative arguments and did not give 0! =1 as it should. He suggested:

```
100 DEF  FNF(X)
110 IF  X(0  THEN STOP
120 IF  X(2  THEN RETURN 1
130 RETURN X*FNF(X-1)
140 FNEND
```

Mr. Mitchell assumed in his letter that X was an integer, and noted that a trap should be added for non-integer X if that assumption did not obtain.

By way of apology I present a bit more carefully written routines. Publishing even two lines of code is like putting up a target. May the sharpshooting readers of *DDJ* find any errors in these routines.

Lines 100-140 are a recursive routine for X!. Lines 200 - 280 are an iteractive routine for the same thing. They are included only to test the correctness of the output from the recursive routine. Lines 10-60 is the testing program, which feeds negative, positive and non-integer values to the functions. The output is also presented. Among other things it shows the impressive 8 digit precision of Polymorphic System's BASIC.

I don't believe in the practice of having error conditions stop a subprogram. Subprograms should send flags back to the main program. The main program can then send a message or stop, or find some way around the problem.

Lastly, if a subprogram is recursive, it is best to check for correctness of the arguments outside the routine. For example, in computing X! the integer function in my program is invoked X-1 times. If the main program filtered out non-integers it would only have to be done once. Another alternative would be to have a routine that does the filtering and then calls FNF.

Thanx to Doug Wyatt for his ever helpful suggestions.

```
10 X=-5\ REM TEST A NEGATIVE ARGUMENT
15 GOSUB 70
20 X=5.6\ REM TEST A NON-INTEGER ARGUMENT
25 GOSUB 70
30 FOR X= 0 TO 13\ REM TEST INTEGER ARGUMENTS
35 GOSUB 70
40 NEXT X
45 STOP
65 REM THIS SUBROUTINE PRINTS A NUMBER AND ITS FACTORIAL
66 REM COMPUTED TWO WAYS.
70 PRINT X,FNF(X),FNG(X)
75 RETURN
>
90  REM FNF(X) RETURNS THE FACTORIAL OF A POSITIVE INTEGER X
91  REM IF X IS NEGATIVE, FNF RETURNS -1 TO INDICATE THE ERROR.
92  REM AS IS CUSTOMARY 0!=1.  IF X IS NOT AN INTEGER THAN THE
93  REM INTEGER PART OF X IS USED AS THE ARGUMENT TO FNF.
94  REM THE PROGRAM OPERATES RECURSIVELY.  IF X IS ZERO OR ONE
95  REM THEN THE VALUE 1 IS RETURNED.  OTHERWISE X! IS
96  REM CALCULATED ACCORDING TO THE FORMULA X!=X*(X-1)!
97  REM NO VARIABLES ARE CHANGED BY THIS PROGRAM
100 DEF FNF(X)
110 IF X<0 THEN RETURN -1\REM FLAG NEGATIVE VALUES OF X
120 IF X<2 THEN RETURN 1 \REM BOTH 0!=1 AND 1!=1
130 RETURN INT(X)*FNF(X-1)
140 FNEND

190 REM FNG(X) COMPUTES X FACTORIAL ITERATIVELY USING THE
191 REM FORMULA X!=2*3*4*...*X.  IF X IS NOT AN INTEGER
192 REM THE INTEGER PART OF X IS USED.  IF X IS NEGATIVE
193 REM THE ERROR IS FLAGGED BY RETURNING -1.  AS USUAL
194 REM 0!=1.  VARIABLES I AND T ARE CHANGED BY THIS FUNCTION.
200 DEF FNG(X)
210 IF X<0 THEN RETURN -1\REM FLAG NEGATIVE VALUES OF X
220 IF X<2 THEN RETURN 1\REM BOTH 0!=1 AND 1!=1
230 T=1\REM T WILL BE USED TO STORE THE PARTIAL VALUES OF X!
240 FOR I=2 TO X\REM ONLY THE INTEGER PART OF X IS USED
250 T=T*I
260 NEXT I
270 RETURN T
280 FNEND
>
>
 -5 -1 -1
 5.6 120 120
 0 1 1
 1 1 1
 2 2 2
 3 6 6
 4 24 24
 5 120 120
 6 720 720
 7 5040 5040
 8 40320 40320
 9 362880 362880
 10 3628800 3628800
 11 39916800 39916800
 12 4.790016E+08 4.790016E+08
 13 6.2270208E+09 6.2270208E+09
Stop in line 45
>
>
>
```

6800 TELEPHONE I/O SOFTWARE

Received: 77-Aug-16

Dear Sir,

Software Exchange is a newly formed company entering the field of home computing. We are developing a line of low cost software for the computer hobbyist. Our emphasis is on the practical application for the home computer.

We are now offering two telephone application programs for the 6800 microcomputer. Each program includes complete documentation, with schematic diagrams and instructions.

Program 1: 6800 automatic telephone dialer-$9.95 postpaid. Includes object code and punched paper tape in Mikbug* format, and instructions for adapting to other 6800 systems.

Program 2: 6800 telephone answering device - $4.95 postpaid. Includes assembly listing and object code. Compatible with any 6800 system.

Mikbug is a registered trademark of Motorola Inc.

Sincerely,
Howard Berenson, Owner
Software Exchange
2681 Peterboro
W. Bloomfield MI 48033

$100-$150 FOR 8080/Z-80 WORD PROCESSING SOFTWARE PACK

News Release *Received: 77 Aug 8*

We are happy to announce the completion of our newest software package, THE ELECTRIC PENCIL by Michael Shrayer. It is a character-oriented word processing system and is available in four versions. It is currently available from the author or at many computer stores. Mail orders should be addressed to: Michael Shrayer, 3901 Los Feliz Blvd., No. 210, Los Angeles CA 90027. Standard versions are $100.00 and Diablo versions are $150.00 (Calif. add 6% tax).

For those in the Los Angeles area, we would be delighted to demonstrate The Electric Pencil. We can be reached any day or evening at (213) 665-7756.

Late Flash! The pencil is now available for the Sol-20/Solos/Cuter for standard printer. Retail: $100.

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 18 **September 1977** **Volume 2, Issue 8**

A REFERENCE JOURNAL FOR USERS OF HOME COMPUTERS

CONSUMER COMMENTS

About Those Elusive PILOT Tapes – *Ronald C. Fryxell*, reply by *Gregory Yob* 4

Some Follow-Ups and A Few New Tidbits from Readers – *Jef Raskin* 5

Computer Applications for the Handicapped – *Warren J. Dunning* 7

SOFTWARE

AN INTERACTIVE PROGRAMMING LANGUAGE FOR CONTROL OF ROBOTS – *Lichen Wang* 8

complete background, documentation, examples, & source listing

A MICROPROCESSOR OPERATING SYSTEM: THE KERNEL – *Max K. Agoston, Ph.D.* 20

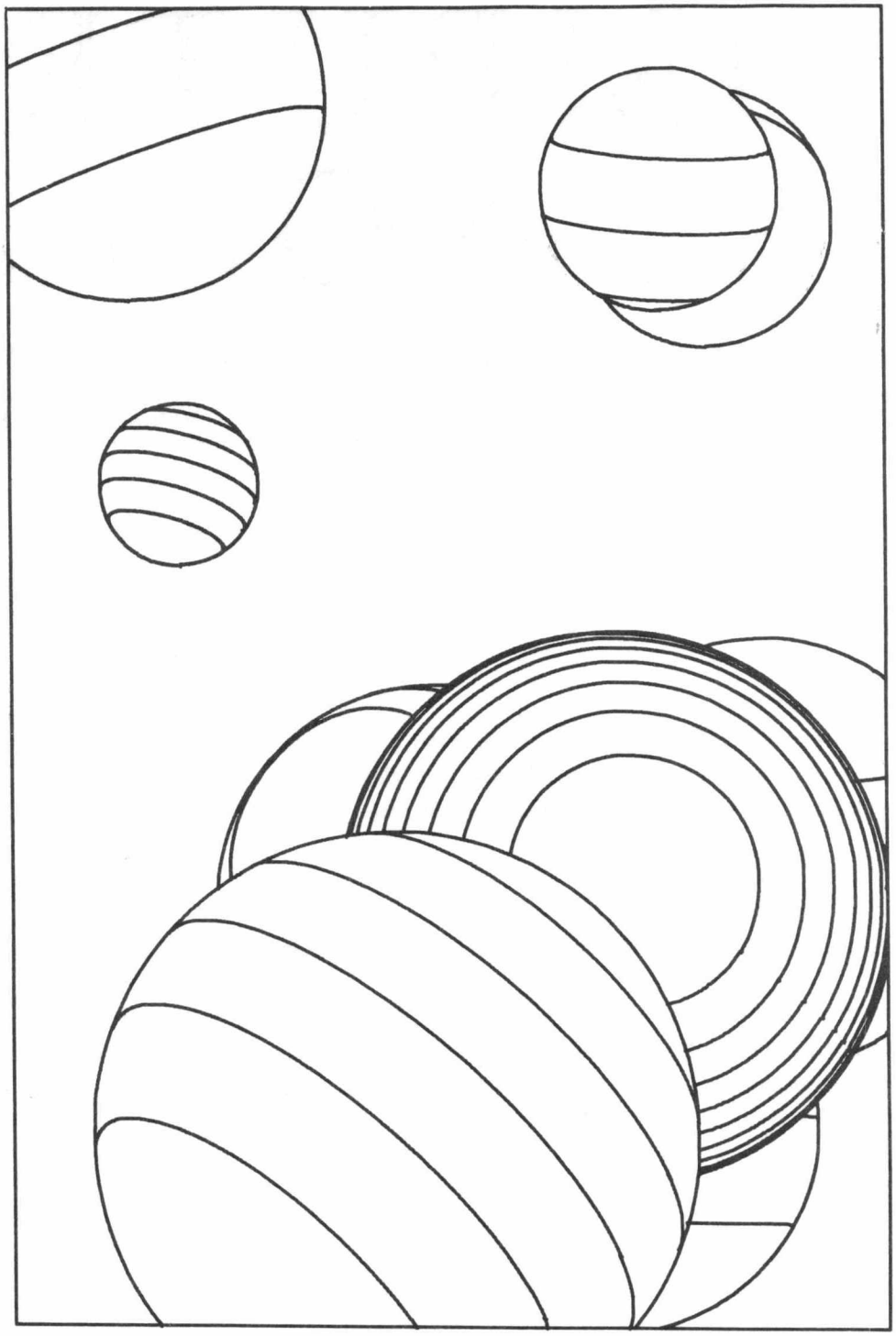
complete documentation, data structure schematics & source listing

Basic-Coded Memory Dumper – *John B. Palmer* 41

SOFT PATCHES

Patching Microsoft's 4.0 Basic on P.T.'s SOL – *Jack L. Calaway* 42

Wee Improvements to MSI's FD8 Operating System – *Gary D. Gaugler* 44



© David Dameron 1979

ABOUT THOSE ELUSIVE PILOT TAPES

HELP!!!!! *Received: 77-Aug-24*

On page 18 of the May 1977 issue there is an address to which one may direct requests for PILOT-8080 in machine readable form. I made such a request several months before I received that issue of *DDJ*, and to this date I have received no reply??

Ronald C. Fryxell
926 Maple Street
Albion, MI 49224

Editorial note: Dr. Starkweather says that a number of people have written to that address and received at best very delayed service. We contacted Gregory Yob of the PILOT Exchange and received the following. – TRW

WHAT THE %*/&%* HAPPENED TO THOSE PILOT 8080 PAPER TAPES

After much fubar, the PILOT Exchange is doing the PILOT 8080 paper tapes. The source tape is some 5 inches thick and the hex format tape is a more reasonable size. I am interfacing a punch to my IMSAI, and when that heroic struggle is completed, I will be able to punch those tapes, and, WILL!! Be assured that they will arrive, probably in 4 – 8 weeks. If that is a problem, give me a call.

Gregory Yob
The PILOT Exchange
P O Box 354
Palo Alto, CA 94301
(415) 326-4039

oldie but goodie available

8K PDP-8/I WITH QUAD DECTAPES & PUNCH FOR $4K

Our glorious(?) Editor spent much of a decade (no pun intended) in mutually supportive relationships with the 8-Family of Maynard, and has long had a warm spot in his heart for these quaint machines. (For the younger generation and the uninitiated, this eccentric discussion concerns one of the first - - and, for many years, the most popular - - minicomputers: the PDP-8 manufactured by Digital Equipment Corporation (DEC) in Maynard, MA.)

Back in the Summer of '76, when he was idling along doing nothing but editing *DDJ* and consulting hither and yon, he stumbled upon a PDP-8/I system that was being put out to pasture (sold on the used equipment market), and - - with heart throbbing and eyes watery - - he grabbed it up. To hold close, spend long wintery evenings with as he had for so many years, and keep him warm at night (with *its* power supply, it can keep a small room quite comfy on even the coldest days).

Then, along came PC'76, the blossoming of *DDJ,* the Computer Faire, the *Silicon Gulch Gazette*, the ACM, teaching at Stanford, NCC in Dallas, PC'77, . . . and it goes on and on. An' our Editor has had little time to do more than turn his old friend on from time to time, and watch it glow and blink. Lest this old 8/I feel spurned and lonely - - for it is an honorable machine with a strong Puritan work ethic, not accustomed to having its diodes idle - - Jim is seeking a good home for it.

It's mounted in two 6' high, 19" racks, has 8K 12-bit words (enough to run DEC's OS-8, DECtape DIBOL, etc.), a TC-01 DECtape controller, four TU-55 DECtape transports, a 20ma current-loop interface for a TTY, a paper-tape punch, schematics & documentation, and comes with a set of diagnostic programs . . . and maybe some interesting DECtapes. When last powered up and connected to an ASR33, a friend ran X-8 (DEC's super whing-ding diagnostic & exercisor software system) on it for about 12 hours with nary a glitch. The version of X-8 being run tested the 8K of memory, TC-01, and all four TU-55's, but was not configured to test the punch. Price for the whole system, FOB Menlo Park, California, is $4,000. Contact Jim Warren, (415) 851-7075, 851-7664.

STORE PRAISED

Dear DDJ; *Received: 77 Sept 16*

Please tell your readers about a *super* computer store that I deal with. It is Computer Hobbies Unlimited, 9601 Kendrick Road, Richmond, VA 23235. Phone is 804-276-5056.

Here are some examples of the service I have received from the proprietors, Terry and Tina:

A factory sealed kit was missing 44 LEDS – the LEDs were replaced immediately by the store.

A factory sealed kit was missing a bag of resistors, capacitors, crystal and an IC. A replacement was mailed in two days by the store. The replacement was never received. Another replacement was provided in about a week.

A CPU board I assembled was checked out by Terry. He found and repaired a solder bridge in 5 minutes that would have taken me an hour or more to find.

I highly recommend the store to anyone who wants service and honesty from their computer store.

David L. Johnson
4106 Montreal Ave
Prince George, VA 23875

IMPERFECTIONS REPORTED RE SOLID STATE'S VB-1 & RAECO RDR

Sirs: *Received: 77-Aug-17*

I am currently working on correcting bugs in the AMSAT satellite monitor and adding VDM support to it. I plan on starting a linking loader using MOSTEK's object standards (*BYTE*, 7/77) and a text editor using the random-access capabilities of the VDM-1 or VB-1 video boards. Please send me information about publishing in *DDJ* as I would be glad to share my work with your readers. *Please see inside front cover. Delighted to have you share with the community. – JW*

I have several comments about my Solid State Music VB–1 card. While it went together without any problems, I found two design deficiencies. A 74193 used as a modulo-15 line counter has a set of gates to clear it whenever its outputs are 1 1 1 1 (decimal 15). Fine, except that they also use the load/reset pulse to trigger another counter and a flip-flop. Since this pulse is very short, depending on the speed of the 74193, I had to try several before finding one that worked. They were aware of this problem, but I don't think that the board should have been designed in this way to begin with. The other problem is that the video output is not blanked when the 2102 display page is switched from the refresh counters to the 8080 address bus. The result is "snow" that appears on the screen whenever the processor is accessing the display memory. I have not been able to devise a fix for this problem. Except for these two problems, the board works fine. The assembly instructions aren't Heathkit quality, but they are quite satisfactory.

I also had problems with my RAECO paper tape reader when trying to read yellow tape. They claim that "buff-colored tape can be read with a little care", but I was unable to read it at all. I've come up with these changes, which have worked quite well: change R9 to 47K and remove R10. This makes the feed hole phototransistor match the data sensor in gain and added hysteresis, apparently to insure valid data whenever a STROBE occurred. However, because of the transmittivity of yellow tape, if enough light were provided to trigger the sprocket hole sensor then the data sensors would see through the tape. Unfortunately, my changes require the use of a D.C.-powered lamp, as the reader is now sensitive to hum on an A.C. lamp.

Phil R Karn
WA3RVL/WB2AJX
112 Edgemoor Lane
Ithaca, NY 14850

SOME FOLLOW-UPS and A FEW NEW TIDBITS FROM READERS

Jef Raskin 77-Sept-20

Thanxs to Ray Atnip for pointing out DEC's use of PEEK and POKE (in BASIC under RT-11) before Micro Soft brought those useful extensions to personal computer land. I apologize to our readers and to DEC for missing this point. I *do* have a copy of DEC's offerings of BASIC under RT-11 and (by way of proclaiming innocence) point out that the manuals *do not* mention PEEK and POKE. If a reader can send me a copy of a DEC manual with this information for my files I would appreciate it. I now have about 40 BASIC manuals including a mimeo'd one from Dartmouth back when BASIC was first invented.

I have gotten two interesting letters from Bob Hoover of 1875 Monte Vista Dr., in Vista CA 92083. He writes that he has had some very bad experience with a Jade Co. (of Lawndale, CA) memory board. Among the problems were a missing chip and badly mangled sockets. The heat sinks did not fit the PC board and even if they had, they would not have fit into the Altair or IMSAI card frame. At last hearing Mr. Hoover had returned the board and asked for a refund. Mr. Hoover found a whopping problem with the Type-away Selectric conversion board for the S-100 bus. In his words: "Ground plane in lower right corner of board *solidly* attached to 5V bus. Results predictable." Once the smoke had cleared and the traces cut apart, the board worked fine. Compliments to the computer for not self-destructing its power supply on that one! Readers ordering that board be warned. Lastly he reports on an interesting experiment. Mr. Hoover bought a "just overhauled" Selectric from the San Diego Byte Shop ($450– a good price). He immediately sent it to IBM to be tested. They found that "This machine has been cleaned (unprofessionally) within the last week or so but I assure you it has *never* been overhauled." A service contract on a Selectric is only $60 per year from IBM. It may well be worth it – if only to know if your equipment has really been overhauled.

Preliminary Reviews:

The Heuristics Speechlab has got to be one of the best products I have seen. The board was given to a beginner (not even any soldering experience) and the instructions led her to a perfectly completed board. This board has a lot of discrete components, and the manual does a superb job of having the builder check and double check everything. The documentation is incredibly good, with many experiments and detailed explanations. The product is called "SPEECH-LAB" and not only provides a speech recognizer for your S-100 computer, but does indeed provide a laboratory course on speech and sound recognition. Amazing, and a very good value. It even includes a microphone.

There will a more complete review coming along soon. But there is a *lot* to test and learn. I can heartily recommend this product. I chose one at random from Heuristic's stock and in spite of the multitude of components there were no missing parts. If you are at all interest in speech recognition there is no need to wonder if this product is for real. It is!

Another product that is very complex is the North Star disk system. I am in the middle of evaluating this blend of hardware and software. First of all, the kit works as advertised. It is a solid design electronically, and the Shugart disk is a wonder. The price is right. And the BASIC software is one of the better hobbyist packages. I am disappointed in the building instructions and the documentation which is extremely weak. North Star expects the buyer to have an oscilloscope handy. The check-out procedure requires a scope. If everything goes well, and you skip the check-out, an oscilloscope is *not* required. There are no adjustments to be made. The experienced hobbyist will find the product excellent. The beginner will want to wait until North Star re-writes the manuals.

Also under review is the Extensys 6 4K memory board. I was given no choice but to take a selected board, as they didn't have any boards in stock when I visited their plant. A more complete review is coming up but I can say that this particular board works perfectly. No hassles. It is the first four-layered printed circuit board that I have seen in a hobbyist product. There is a certain hardware slickness here that is quite advanced. The documentation is, again, slick. But it is not nearly as complete as one might wish.

pseudo-contest

MANUFACTURERS' DELIVERY DELAY

Tom Williams, Assistant Editor, *DDJ* *Dated: 77-Oct-24*

Our "pseudo-contest" for the manufacturer with the longest delivery delay, announced in the June/July issue of DDJ, has received moderate response. However, a number of the people responding have supplied some very well-documented "dossiers", which have allowed us to discuss some quite specific problems with the manufacturers, most of whom have been co-operative in furnishing information.

All this can be a bit time-consuming as we are taking care to see that we talk to the proper people – usually the presidents of the companies. What we are finding out will be the subject of an article that is now taking shape around the material that has been received to date.

So, this is a "last call" to anyone who wishes to send in *documented* evidence of what they consider unreasonable delay in delivery.

A major article will appear in the next issue on this matter.

OUR INDEPENDENT PRODUCT REVIEWER IS STILL INDEPENDENT

Dear Jim: *Received: 77-Sep-23*

Your *Journal* has established a reputation that is recognized by equipment manufacturers and consumers as being impartial. In an area such as data processing, where confusion/distrust is never more than a rumor away, your *Journal* has served as a rock of clear thinking and open exchange.

Mr. Jef Raskin, who heads your Product Evaluation Group, has given considerable support to the Poly-88 computer system in his article in the Nov/Dec 76 issue of your *Journal*, ("Poly-88–An Excellent System", pg 16).

Would it not be more in keeping with the image of the *Journal* to point out that Mr. Raskin is a Poly Morphic dealer?

Robert S Hoover
1875 Monte Vista Dr
Vista, CA 92083

Editorial Note: Jef, when contacted on this point responded as follows: 1) He was not a dealer when he wrote the article; 2) When he discovers something good in computers he believes in making it available to customers; 3) He is NOT a Polymorphics dealer and never has been. His name was listed in error, and 4) The firm he works for, Bannister and Crun, has sold Poly computers and Altair and IMSAI . . .

Mr. Brian Wilcox, Vice President of Polymorphic, when contacted by phone, confirmed Jef's statement. – TRW

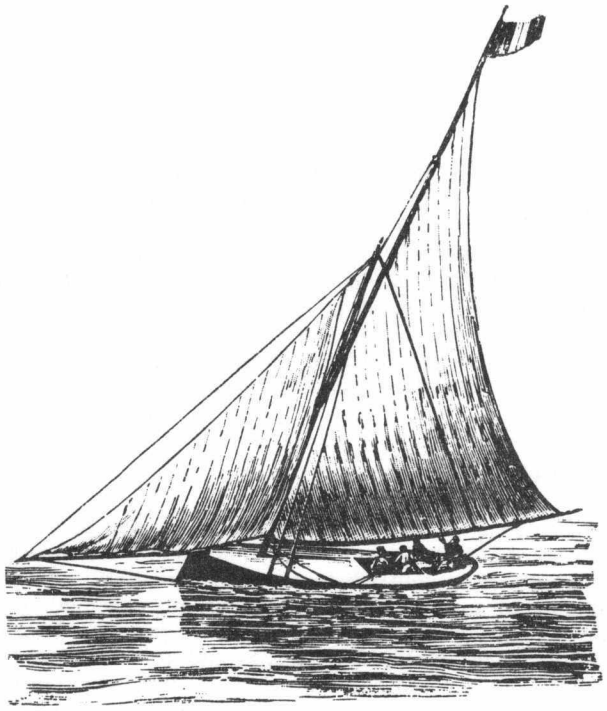
COMPUTER APPLICATIONS FOR THE HANDICAPPED

Greetings,

Received: 77-Oct-3

So, you finally got your computer up and running. What now? Have you spent a small fortune on your computer only to impress the neighbors or play Star Trek? How about putting your computer and your expertise in hardware design and/or programming to practical and worthwhile use? What about a project for your neighborhood computer club? The challenge is this: to develop computer-based systems to aid the handicapped. The microprocessor makes possible many devices that were only dreams several years ago. Talking computer terminals and voice output reading machines would be very valuable to the blind. A speech recognition system and readout would be invaluable to the deaf. Control systems for artificial limbs, voice input telephone dialers and personal communications devices for remote computer access are all possible. The literature, especially the personal computing conferences and publications, abounds with good and realizable ideas. It is up to us to implement these ideas and get them into the hands of someone who can really use them. Why not seek out some handicapped people or some organization for the handicapped in your community and see what *they* need? If your device has sufficient general demand, there are several not-for-profit organizations which could manufacture it at the lowest possible cost to the handicapped. Probably the best way to find out what's already been done, what's needed, and how to do it, is to communicate our ideas with each other. A small group of interested people, under the leadership of Dr. Robert Suding (The Digital Group), met at PCC 77 in Atlantic City to discuss some problems of the handicapped and how we can help. We will be meeting at most major personal computing affairs and invite your participation. We also invite your ideas, problems and suggestions which may be of help in this project. To this end, there will be an ongoing column in BYTE to promulgate these ideas. Please send your suggestions and ideas to:

Computers for the Handicapped
c/o Warren J. Dunning
5939 Woodbine Ave
Philadelphia, PA 19131



Mr John Ingram
3301 Don Tomas Dr
Carlsbad, CA 92008 714-229-6925

Dr Murray Turrof
Dept of C.I.S.
New Jersey Inst of Tech
323 High St
Newark, NJ 07102 201-645-5503
(Doing research in this area)

Mr Ira Laefsky
8808 Patton Rd
Philadelphia, PA 19118 215-836-4486
(Dept of C.I.S.-U. of Penna. Eye movement control of prosthetic devices)

Mr Richard Jackson
19880 Cranbrook No. 120
Detroit, MI 48221 313-341-6304
(Will act as convention coordinator)

Mr Marty Glassman
9601 Ashton Rd Apt 0-28
Philadelphia, PA 19114
(Will write/edit magazine articles)

Mr T.F. Bogart
Southern Station – Box 246
Hattiesburg, MS 39401 601-266-4292

Ms Kathleen Murphy
601 E 12th St
Kansas City, MO 816-374-5081

Mr Joseph Wise Jr
3804 Parkmont Ave
Baltimore, MD 21206 301-426-5169

Ms Barbara Froding
RD 1
Bridgeton, NJ

Mr Frank Whitescarver
391 Ski Trail
Kinnelon, NJ 07405

Mr Douglas Cook
3006 Hazelhurst St Rd
Pittsburg, PA 15227

Mr Phillip L.S. Lum
339 Treasure Boatway
Sarasota, FL 33581

Warren J. Dunning
5939 Woodbine Ave
Philadelphia, PA 19131 215-879-9383
(Will publish newsletter and coordinate things)

Mr Milton E. Sussman
16 W Gowen Ave
Philadelphia, PA 19119 215-CH7-3181
(will coordinate magazine articles)

Mr Dave Day
Box 29
Maynard, MA 01754

Ms Helen Baltzer
P.O. Box 143
West Swanzey, NH 03469

Mr Gordon French
614 18th Ave
Menlo Park, CA 94025 415-325-4209

Dr Robert Suding
c/o The Digital Group
P.O. Box 6528
Denver, CO 80206

Mr Burchenal Green
Managing Editor, Creative Computing
P.O. Box 789M
Morristown, NJ 07960

****

An INTERACTIVE PROGRAMMING LANGUAGE for CONTROL of ROBOTS

by Lichen Wang

Editorial Note:

In our haste to get Dr. Dobb's *back on schedule, we made a horrible error. In* DDJ *no. 18, our layout department pasted up a whole section of text out of sequence in Lichen Wang's article. This mistake got past everybody whose job it is to catch such things including myself and I take ultimate responsibility for it. In an effort to make things right we are reprinting the text of the article (for the listings, see* DDJ *no. 18) with our sincere apologies to Lichen Wang and our readers whom we left scratching their heads. –TRW*

Introduction

This is a simple-minded "Tiny" language. The basic idea is to use an ASCII keyboard as the command input device for a microprocessor controlled robot. Rudimentary commands consisting of a single ASCII character are executed by the robot as soon as they are typed. Some primitive forms of IF-THEN-ELSE, REPEAT-n-TIMES, and run-time-macro are included. Paired parentheses are used to group any number of commands into a single command and enable one to construct complex structures. The run-time-macro is probably the most important feature that makes a simple language such as this one powerful and easy to use.

This language is open-ended and is not limited to any specific robot. To illustrate this point, a version of this language is implemented on an 8080-based microprocessor to control a robot who has a "mind" but no "body". Later in this article, two different "bodies" are attached to the robot's "mind" as examples. Revisions and implementations on other microprocessors can be made if there are such interests.

The Mind

In this implementation, one uses the VDM-1 interface and a video display to represent the robot's mind – so that one can always read what is in the robot's mind. Other hardware required are: an ASCII keyboard with interface, and an 8080 based microprocessor system with 8K of RAM at 0000-1FFF. The major part of the memory is used for the stack. The assembler listing and object code of the program are included following this article.

The robot's mind consists of an input line buffer of 58 characters long, 15 lines of macro definition space (also of 58 characters each) and a 4-decimal-digit accumulator. When one starts the program, the entire screen of the video display is cleared, the characters "A=0000" are displayed at the top right corner, and a cursor is shown at the top left corner (i.e., the beginning of the input line buffer). Whenever the cursor shows in the input line buffer, it means the robot is not doing anything but waiting for input from the keyboard. When the cursor disappears, the robot is executing a command.

robot. More will be added when a body exists.

- ␣ (space, blank) No operation
- \+ Increment the accumulator
- – Decrement the accumulator

The no-op command is sometimes needed in IF-THEN-ELSE commands discussed later. When one types the character "+" or "–", it replaces the cursor on the screen, the accumulator is changed accordingly, the "+" or "–" is erased, and then the cursor reappears. (But all these happen in a flash.) Attempts to increment beyond 9999 or decrement below 0000 will not change the content of the accumulator.

If one types the sequence of characters "(++–+)", the accumulator will be changed after the ")" character is typed, and not while the individual "+" and "–" are typed. The sequence is stored and displayed on the input line buffer as entered, and the cursor is moved to the right at the same pace. If a RUB-OUT (or DEL) character is typed before the closing ")" is typed, the input buffer is erased and the partial command is not executed at all.

The bodyless robot has two IF-THEN-ELSE commands. One of them tests the accumulator. The other is a random decision maker:

T [then command] [else command] Test if A>0

? [then command] [else command] Randomly choose one

All IF-THEN-ELSE commands have this common syntax. The leading character, such as "T" or "?", specifies the IF condition, [then command] is the command to be executed by the robot when the IF condition is met, and [else command] is the alternate command to be executed if the IF condition is not met. Note that since the IF-THEN-ELSE itself is a commmand, it too can be used as the [then command] or [else command]. Thus: "??(case1) (case 2)?(case 3) (case 4)" has the same effect as "?(?(case 1) (case 2))(?(case 3) (case 4))" and will both cause the robot to randomly choose to execute one of the four cases.

The bodyless robot has two REPEAT-n-TIMES commands. One repeats a fixed number of times as specified, and the other uses the current value of the accumulator as the repeat count.

324 [command] repeats the [command] 324 times

A [command] repeats the [command] A number of times.

For example, "A–" will zero the accumulator, and "A+" will double the content of the accumulator. More complex sequence can be constructed using parentheses, e.g., 34 (6+12(+–+)A+).

Last, and by no means the least, is the macro definition command. It is used to define any character, which is not one of the basic commands, to be the equivalent of a simple or complex command.

D [ch] [command] Define [ch] to be [command]

For example, "A–" clears the accumulator, one can type "DZA–" and from there on simply type "Z" to clear the accumulator. One can also have "DX(Z34(6+12(+–+)A+))" for whatever purpose.

When one types in the definition, the sequence of characters are entered into the input buffer, and the cursor is

moved to the right. When the last character (which completes the definition) is typed, the robot will search its mind to find if an old macro definition of that same character exists. In case an old definition exists, it will be marked to be deleted later. The robot then searches its mind again to find an empty line and copies the definition there. The examples above will be shown as "Z=A–" and "X=(Z34(6+12(+–+A+))", respectively.

If one attempts to define a character which already has a meaning in this language, such as "ø", "+", "–", "T", "?", "A", "D", or any digit, the robot will erase the input buffer and will not do anything else. The RUB-OUT (or DEL) key can also be used to abort the definition without damaging the old definition if any. To erase an old macro definition without creating a new one, one types "D[ch]ø". This is consistent with the convention that any undefined character is also a no-op.

The Body

One can construct a very simple robot similar to the well known "Turtle". Its moving parts consist of two stepping motors driving two independent wheels plus an idler wheel to keep its balance. A micro-switch attached to a bumper serving as its sensor and other "bells and whistles" can be added on later. The two driven wheels are arranged like the wheels on a barrow cart, and not like a bicycle. When both stepping motors are driven forward or backward, so is the robot. But when the motors are driven in opposite directions, the robot rotates.

The microprocessor interface consists of one parallel input port and one parallel output port. One bit of the input port is connected to a flip-flop which in turn can be set by touching the micro-switch on the robot. Two other bits of the input port tell the micro-processor whether the stepping motors are ready to be pulsed. Five bits of the output port are used; one of them used to reset the sensor flip-flop, and two bits each – direction and step, for the stepping motors. Other bits of the input/output ports are for further expansion.

To control the body of this robot, one adds the following command to what was discussed in the previous section:

F Move forward one step
B Move backward one step
R Rotate right one unit angle
L Rotate left one unit angle
S [then command] [else command] Test & reset the sensor

The "unit angle" is the smallest angle the robot can rotate. It is determined by the step size and the distance between the two driven wheels. The "S" command is an IF-THEN-ELSE command similar to the "T" and "?" commands discussed in the last section. The assembler listing and the object code of the modifications needed to implement these new commands are presented below.

Another Body

Another robot of a sort can be even easier to construct – if the micro-system has a DAZZLER. One uses software to control an imaginary turtle on another video display connected to the DAZZLER. The black & white mode is used to get a 128 x 128 grid. The new commands added are:

F Move forward one grid cell
R Rotate 45 degrees right
H Go home (i.e., the center of screen)
N Face North (i.e., up)
W Leave white trace
B Leave black trace
C Clear screen

Note that the "B" command here is in conflict with the one in Section III. But since the robot has only one of these two alternative bodies, this does not cause any problem. The "C" command clears the screen to the opposite of the "trace". That is, if "W" is in effect, "C" clears the screen to all black. If "B" is in effect, "C" clears the screen to all white. When one marches the turtle off the boundary, it will re-enter the boundary on the opposite side of the screen. The assembler listing and the object code of the modifications needed to implement these new commands are presented below.

Programming Examples

The following are a few examples of programs written in this language. These examples are heavily biased toward graphics and recursive macro definitions. This should not be interpreted as representing the strength or limitation of this language. One reason for using graphic examples is that the action of a mechanical robot such as the one described in Section III is more difficult to represent here than the graphics. Another reason is that this language has not been tested on a mechanical robot yet.

There are very few programming languages that allow recursive definitions, and many hobbyists think recursive routines are difficult to understand. There are many indications that the language in which one learns to express one's ideas profoundly influences one's process of thinking. It can be shown that recursion is really not any more difficult to understand than iteration.

Example 1: A walking measuring tape

Using the robot described earlier, one can define the macros:

```
DM (Sø(FM)B+)
DZ (A–Søø9999R)
```

Now if one types the character "Z", the robot will clear its accumulator, reset its sensor, and start to rotate. When the robot is facing the direction that one wants to measure, one types the character "M" and the robot will stop rotating and start to execute the macro "M". It first tests its sensor. If it did not touch anything, it will move forward one step and recur to macro "M". This process will repeat until the robot finally touches something, it will go back one step, increment the accumulator, and return from the current level of macro "M". Since the macro "M" has been nested, the robot will step back and increment accumulator as many times as it had stepped forward. This means the robot will find its way back to where it started from, and the accumulator will show the distance to the obstacle in units of the robot's step size.

Example 2: Windmill

Using the robot described above, one can define macros to draw all kinds of windmills:

```
DX (CA-8(HNARP+))
DP (. . . . . )
```

Where "P" is a macro defined by the user to draw some curve. "P" should use only the "F", "R", and fixed number repeat commands and paired parentheses. After one defines "P", one can type "X" to draw the windmill. The robot will first clear the screen, zero the accumulator, and then repeat 8 times drawing the pattern "P", each time starting at the home position but facing a different direction. The accumulator will be incremented each time after "P" is drawn, and will be used to determine the direction of the next iteration.

Example 3: Dragon curve [1]

The dragon curve is a family of very interesting curves. Figure 1A shows the 3rd order dragon curve and Fig. 1B shows the 8th order one. These curves can be constructed by

folding a strip of paper. One keeps doubling the strip and folds it into half the current length for n times – n being the order of the curve. One then opens all creases (there are 2**n–1 of them) to 90 degrees each, and a dragon curve is formed. The robot of Section IV can also draw these curves – up to the 12th order due to the 128 x 128 resolution. One defines:

```
DLT (–L6RJ+) G
DJT (–L2RJ+) G
DG4F
DQ (HN6R30FCF4RARL)
```

After these are defined, one sets the accumulator to the desired order (0 through 8) and types the character "Q". Recall that one can zero the accumulator by typing "A–", and that one can then set the accumulator to n by typing "n+". The macro "G" defines the segment size of the dragon. One can redefine "G" to be a smaller number of "F"s so that higher order dragons can fit the screen. With "DGF", up to the 12th order dragon can be drawn.

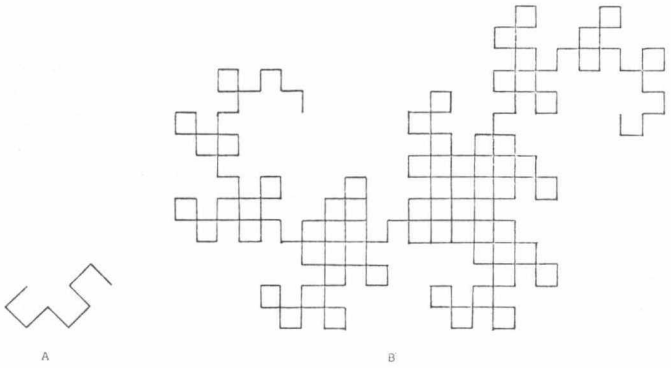


Figure 1. Dragon curve of (A) order 3, and (B) order 8.

The dragon curve is defined by the macro "L", while macro "J" defines the reversed dragon curve. When folding the paper strip, one may observe the fact that the nth order dragon consists of a (n–1)th order dragon, a 90 degree fold, and another (n–1)th order dragon in reverse. The definition of "L" and "J" then becomes obvious when one adds the fact that the 0th order dragon is a plain strip of paper which is defined as "G". The macro "Q" is used to pick a starting position, clear the screen and pick a starting direction.

Example 4: Sierpinski curve [2]

Sierpinski curves of order 1 and 3 are shown in Fig. 2A and 2B respectively. Like Example 3, one may define:

```
DIT (–I2FI5RG5RI2FI+) 2R
DG4F
DY (HN63F2R61FRC4 (2FI))
```

After these are defined, one sets the accumulator to the desired order and types "Y" to draw the curve. Again, "G" can be redefined to a smaller number of "F"s to draw higher order "Y"s. For "DG0F", the accumulator can be set to 5.

The Sierpinski curve is closed and consists of four identical sides arranged in different directions and connected by a short line "2F". The sides are defined (recursively again) in the macro "I". and the closed curve itself is defined in the last part of the macro "Y", namely: "4(2FI)".

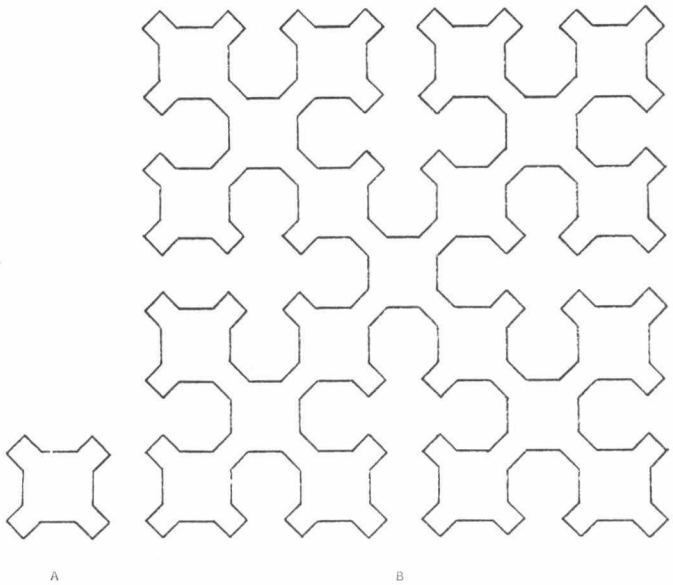


Figure 2. Sierpinski curve of (A) order 1, and (B) order 3.

Example 5: Hilbert curve [2]

Hilbert curves of 1st and 5th order are shown in Fig. 3A and 3B respectively. The macro definitions are presented without further explanation:

```
DUT (–VG6RU2RGUG6RV+) 6R
DVT (–U2RGVG6RV2RGU+) 2R
DG4F
D$ (HN63F2R63FC2RU)
```

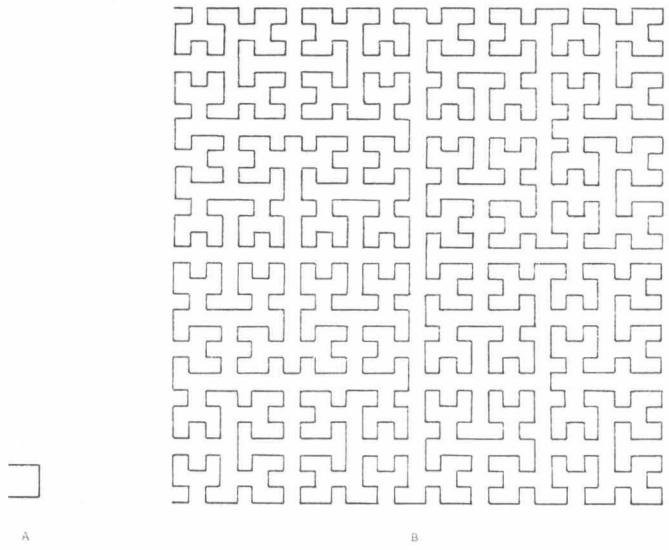


Figure 3. Hilbert curve of (A) order 1, and (B) order 5.

Discussion

In Section II, only a very limited number of basic commands, a total of 8, are included and yet it has been demonstrated that quite complicated tasks can be accomplished by this simple language. In practice, one would put in more basic commands to make the language even easier to use. One would also want to include more facilities, e.g., variables or a stack for the accumulator, etc. As is evident when one glances through the program listings which follow, these additions can be easily made.

Footnotes:

1. See Scientific American April 1967 pages 116-123.
2. See Scientific American December 1976 pages 124-133.

One question that may arise is the limited number of characters in the ASCII code. If the keyboard is equipped with SHIFT and CNTRL keys, there are 128 different characters; and if the parity bit is not used and can be controlled from the keyboard, there are 256 different codes. One can further expand this by defining a few characters to be special escape characters. When the character table lookup routine encounters one of these escape characters, it will get the next character and scan a different table.

As the title of this article implies, the scope currently is limited to "interactive control"; there is no provision for artificial intelligence. Another limitation of this language in its current state is that there is no way to handle concurrent actions and real time interrupts from the robot. The author looks forward to feedbacks from the readers. He especially welcomes ideas which would enlarge the scope of the language as opposed to what specific command should be included or deleted.

The author wishes to thank Harvey A. Cohen and Marvin R. Winzenread for many interesting discussions and their pioneer works and publications in related topics that led to the development of this language.

□□□□

SOME SUPPLEMENTARY OBSERVATIONS FROM GREG YOB

To Lichen Wang

Re Tiny Robot Language (TINMAN) *Received:77-Oct-19*

Your design is hard to beat for conciseness. I found it a bit difficult to follow the examples, but that would be clearer with a live machine in front of me. All in all, a very clear and clean approach.

My first thoughts were mostly cosmetic in nature, ie, I prefer to be able to give multicharacter names, and locating the various tests embedded in a command string was sometimes confusing. Of course, tests can have parentheses appropriately located to mark them, and multicharacter command names or macro names could be done with a delimiter convention, ie

TURTLE

Tests could be either:

TEST (first) (second)

or

T! (first) (second)! note that (xxx) stands for whatever goes there.. As I said, these are cosmetic in nature . . .

In pondering the addition of bodies, I noted that the commands for the bodies outnumbered those for the mind. Since the macro facility is essential to the language, why not use it to implement each body rather than have each body eat up more characters in the ASCII set . . . The beginnings of such an approach would look like this (bear in mind I have not run it all the way through to be consistent with other aspects of TINMAN). Two new commands should take care of it:

I - Input from robot & place value in Accumulator

O (number or A) Output value to Robot

Of course, a stack gets pretty important, so some kind of push & pop commands may help out as well . . . Bear in mind I am not just adding some more commands, I am using these to remove all commands that ultimately are input and output commands.

With I, O, push & pop, then the other commands for a body may be written as macros. While we are at it, it might be neat to "store" screenfuls of macros in other sections of memory or on an external device, so that those used for a body would not clutter the VDM display.

A hint of an idea towards interactive & real-time uses would be to have the robot's input actually be a macro call. That is, the value returned by the robot would be a defined macro which is then executed. This would either be an interrupt (god forbid!!) or a polling kind of thing where after each operation, the inputs are polled, and if one needs treatment, the state of the language processor is stacked, and the required macro is called. A suitable form might look like:

P (robot #) (macro or command) or A	Poll robot and do command if ready, command can then input whatever value from robot, etc.

Of course, if robots appear, so must numbers for I and O.

Two methods may be used, one is to have the polling after input of a command from the keyboard, and the other is to have it at intervals within the program, say after each 20 executions of a mind primitive routine from the commands dispatcher, and after each 1000 passes through the keyboard input wait loop.

A naggy detail is that a robot may want some attention from the keyboard. This could be done, and would allow keyboard input during a run of a command by a slightly different version of I and O:

I -	Input number from keyboard & put in accumulator
I number or A	Input from Robot number FOO (The keyboard is Robot 0)
O value or A	Write value on VDM in commands input buffer spot or, preferably, bottom line
O value or A (blank) value or A	Write first argument to Robot indicated by second argument.

This gets me into more tangles than I started with. However, I offer these as suggestions towards some of the problems you mentioned in your document.

I am of course interested in getting Source & Object for playing with. Meanwhile, thanks for an interesting afternoon of thought. . .

Gregory Yob
PO Box 354
Palo Alto, CA 94301
(415) 326-4039)

```
ASSEMBLER LISTING OF "THE MIND"
 ********************************************
 *                                          *
 *          W S F N                         *
 *    (Which Stands For Nothing)            *
 *    ------------------------              *
 *  An Interactive Programming Language     *
 *     for Control of Robots                *
 *    Version 0.1, September 1977           *
 *       By Lichen Wang                     *
 *                                          *
 ********************************************
 *
 * We have 8K of RAM at 0000-1FFF, a VDM
 * at CC00-CFFF, and an ASCII keyboard.
 *
                    ORG   00000H
029D  DANGER EQU   LAST+64     *Don't let stack
FD63  STKLMT EQU   -DANGER     *eat up program.
1FFE  STACK  EQU   01FFEH
1FFE  SEED   EQU   STACK       *Seeds for RNDB
00C8  VDM    EQU   0C8H        *Standard VDM-1
CC00  SCREEN EQU   0CC00H      *port & addr.
0000  KEYSTA EQU   000H        *Status port,
0080  KEYDAV EQU   080H        *key ready bit,
0001  KEYDAT EQU   001H        *& data port.
0000
```

```
                        *   *** In the beginning ***
                        *
0000 310020                 LXI  SP,SEED+2   *Seeding RNDB
0003 CD5C02                 CALL INIT        *Baptize the
0006 AF                     XRA  A           *robot, and
0007 D3C8                   OUT  VDM         *brain wash it
0009 2100CC                 LXI  H,SCREEN
000C 3ED0                   MVI  A,(SCREEN+1024)/256
000E 3620               I1  MVI  M,' '
0010 23                     INX  H
0011 BC                     CMP  H
0012 C20E00                 JNZ  I1
0015 21413D                 LXI  H,'A='      *Put 'A=0000'
0018 223ACC                 SHLD SCREEN+58   *in his mind
001B 213030                 LXI  H,'00'      *(at the top-
001E 223CCC                 SHLD SCREEN+60   *right corner).
0021 223ECC                 SHLD SCREEN+62
                        *
                        *   *** DIRECT ***
                        *
                        * This is the main loop that clears
                        * garbage (old definitions), reads a
                        * command from the keyboard, puts it in
                        * the robot's input buffer (which is the
                        * top line on the VDM), and then let the
                        * robot execute the command.
                        *
0024 31FE1F             DIRECT LXI  SP,STACK
0027 2100CC             D1     LXI  H,SCREEN     *Clear the input
002A 063A               D2     MVI  B,58         *buffer,
002C 3620               D3     MVI  M,' '
002E 23                        INX  H
002F 05                        DCR  B
0030 C22C00                    JNZ  D3
0033 0E7F                      MVI  C,@DEL       *and other lines
0035 CDAE00                    CALL FIND         *that has been
0038 CA2A00                    JZ   D2           *marked.
003B 60                        MOV  H,B          *B=0, just copy
003C 68                        MOV  L,B          *From keyboard
003D 1100CC                    LXI  D,SCREEN     *To input buffer
0040 CD5000                    CALL RCMD
0043 2100CC                    LXI  H,SCREEN     *From buffer
0046 50                        MOV  D,B          *To nowhere
0047 58                        MOV  E,B          *(DE=0)
0048 06FF                      MVI  B,0FFH       *Now execute it
004A CD5000                    CALL RCMD
004D C32700                    JMP  D1           *Repeat all this
                        *
                        *   *** RCMD *** & *** CMD ***
                        *
                        * This subroutine is the syntax scanner.
                        * If B reg. is 0, it scans for one
                        * command and returns.  If B reg. is not
                        * 0, it scans and executes one command.
                        *
                        * CMD is another entry, which assumes
                        * that the first character of the
                        * command is already in C reg.
                        *
0050 CD8100             RCMD   CALL GETCH        *Get character
0053 DB00               CMD    IN   KEYSTA       *But did the
0055 E680                      ANI  KEYDAV       *master have
0057 C22400                    JNZ  DIRECT       *other wishes?
005A E5                        PUSH H
005B CDC600                    CALL RESRV        *Table lookup
005E 78                        MOV  A,B          *for reserved
005F C26500                    JNZ  C1           *Not reserved
0062 B7                        ORA  A            *Reserved
0063 E3                        XTHL             *Go do that
0064 C9                        RET              *B is in Z flag
0065 B7                 C1     ORA  A            *Not reserved
0066 E1                        POP  H
0067 C8                        RZ               *Don't execute
0068 E5                        PUSH H            *Excecute user
0069 CDAE00                    CALL FIND         *defined command
006C C27F00                    JNZ  C2           *Undefined
006F E5                        PUSH H            *Defined
0070 2163FD                    LXI  H,STKLMT     *But let's see
0073 39                        DAD  SP           *if there's more
0074 7C                        MOV  A,H          *room in stack.
0075 B7                        ORA  A
0076 FA2400                    JM   DIRECT       *Forget it
0079 E1                        POP  H            *OK, restore HL
007A 23                        INX  H            *points to user
007B 23                        INX  H            *definition
007C CD5000                    CALL RCMD         *Scan that
007F E1                 C2     POP  H            *Undefined ch.
0080 C9                        RET              *is a No-op
                        *
                        *   *** GETCH ***
                        *
                        * This subroutine gets a character to C
                        * reg., and optionally make a copy.  HL
                        * points to where to get the character
                        * from.  But if HL=0, we input from the
                        * keyboard.  DE points to where to make
                        * the copy.  But if DE=0, no copy is
                        * made.
                        *
0081 7B                 GETCH  MOV  A,E          *Pointer to buff
0082 E63F                      ANI  63
0084 FE39                      CPI  57           *End of buffer?
0086 D22400                    JNC  DIRECT       *Buffer overflow
0089 B2                        ORA  D            *Make copy?
008A CA9000                    JZ   G1           *No copy, fine
008D 3E7F                      MVI  A,@DEL       *Copy, show
008F 12                        STAX D            *cursor
0090 7C                 G1     MOV  A,H          *Where from?
0091 B5                        ORA  L
0092 C2A500                    JNZ  G3           *From memory
0095 CD4601             G2     CALL RNDB         *(Randomlize)
0098 DB00                      IN   KEYSTA       *From keyboard
009A E680                      ANI  KEYDAV
009C CA9500                    JZ   G2
009F DB01                      IN   KEYDAT
00A1 4F                        MOV  C,A
00A2 C3A700                    JMP  G4
00A5 4E                 G3     MOV  C,M          *From memory
```

```
00A6 23                 INX  H
00A7 7A          G4     MOV  A,D          *Copy?
00A8 B3                 ORA  E
00A9 C8                 RZ                *No copy, done
00AA 79                 MOV  A,C          *Copy it
00AB 12                 STAX D
00AC 13                 INX  D
00AD C9                 RET
                 *
                 *   *** FIND ***
                 *
                 * This subroutine searches the robot's
                 * memory to find a char.  It compares
                 * the C reg. with the first character
                 * of the 2nd through 16th line on the
                 * VDM screen.  If a match is found, the
                 * Z flag is set, and HL points to where
                 * it found it.  If no match, Z is reset.
                 *
00AE D5          FIND   PUSH D
00AF C5                 PUSH B
00B0 2140CC             LXI  H,SCREEN+64  *Ch.1 of line 2
00B3 114000             LXI  D,64         *Line length
00B6 060F               MVI  B,15         *Total lines
00B8 79                 MOV  A,C          *Character
00B9 BE          F1     CMP  M
00BA CAC300             JZ   F2           *Match found
00BD 19                 DAD  D            *No, next line
00BE 05                 DCR  B
00BF C2B900             JNZ  F1           *More lines
00C2 04                 INR  B            *No, reset Z
00C3 C1          F2     POP  B
00C4 D1                 POP  D
00C5 C9                 RET
                 *
                 *   *** RESRV *** & *** DIGIT ***
                 *
                 * This subroutine checks the character
                 * in C reg. to see if it is one of the
                 * reserved characters.  If it is, the Z
                 * flag is set and the HL reg. is loaded
                 * with the address of the handler of
                 * that reserved character.  If C reg.
                 * does not contain one of the reserved
                 * characters, the Z flag is reset.
                 *
                 * The entry DIGIT will check if C reg.
                 * contains a digit.  It sets/resets Z
                 * flag the same way but does not change
                 * HL reg.
                 *
00C6 C5          RESRV  PUSH B
00C7 060A               MVI  B,TBLSZ      *# of entries
00C9 213E02             LXI  H,TABL       *& addr of table
00CC 79                 MOV  A,C          *Character
00CD BE          V1     CMP  M
00CE 23                 INX  H
00CF CAE600             JZ   V2           *Match found
00D2 23                 INX  H            *No
00D3 23                 INX  H            *Skip addr
00D4 05                 DCR  B
00D5 C2CD00             JNZ  V1           *More in table
00D8 C1                 POP  B            *No, not there
00D9 211702             LXI  H,CMD09      *May be a digit
00DC 3E39        DIGIT  MVI  A,'9'        *Digit 0-9?
00DE B9                 CMP  C
00DF D8                 RC                *No
00E0 79                 MOV  A,C          *May be
00E1 D630               SUI  '0'
00E3 D8                 RC                *No
00E4 BF                 CMP  A            *Yes, set Z
00E5 C9                 RET
00E6 7E          V2     MOV  A,M          *Matched table
00E7 23                 INX  H
00E8 66                 MOV  H,M          *Load HL from
00E9 6F                 MOV  L,A          *table
00EA C1                 POP  B
00EB C9                 RET
                 *
                 ****************************************
                 *
                 * From here on are the routines that
                 * handle each reserved characters, and
                 * a table of these characters.
                 *
                 * RCMD or CMD will use the table and
                 * may jump to one of these routines.
                 * The Z flag is set if we are in the
                 * non-execute mode.  Reg. B also will
                 * be 0 in this case.  Reg. C has the
                 * character to be handled.  HL points
                 * to where the command string is from.
                 * If HL=0, command is from keyboard.
                 * DE points to where a copy of the
                 * command string is to be made.
                 * If DE=0, no copy is going to be made.
                 *
                 * Each handler should end with a RET and
                 * reg. BDEHL should not be destroyed.
                 *
                 ****************************************
                 *
                 *   *** blank ***
                 *
                 * A blank (space) is a no-op.
                 *
00EC C9          CMDBL  RET               *No-op
                 *
                 *   *** D ***
                 *
                 * D[ch][command]
                 *
                 * This command defines the [ch] to be
                 * the equivalence of the [command].
                 * [ch] must not be one of the reserved
                 * characters.
                 *
                 * If old definition exists, the old one
```

```
                        * will be marked and will be deleted
                        * by DIRECT.  If the new definition
                        * is No-op ' ', it will not be stored
                        * in the robot's mind.  Any undefind
                        * character is a No-op anyway.
                        *
00ED CD8100             CMDD   CALL GETCH
00F0 E5                        PUSH H
00F1 CDC600                    CALL RESRV       *Is ch reserved?
00F4 CA2400                    JZ   DIRECT      *Yes, forget it
00F7 78                        MOV  A,B         *No
00F8 B7                        ORA  A           *Execute?
00F9 C20001                    JNZ  CD1         *Yes
00FC E1                        POP  H           *No, non-execute
00FD C35000                    JMP  RCMD        *Skip it
0100 CDAE00             CD1    CALL FIND        *Execute
0103 C20801                    JNZ  CD2         *No old def.
0106 367F                      MVI  M,@DEL      *Mark old def.
0108 C5                 CD2    PUSH B           *Now, new def.
0109 0E20                      MVI  C,' '       *Find space in
010B CDAE00                    CALL FIND        *robot's mind
010E C22400                    JNZ  DIRECT      *No luck
0111 C1                        POP  B           *Found room
0112 71                        MOV  M,C         *Tell him the ch
0113 23                        INX  H
0114 363D                      MVI  M,'='       *is defind as
0116 23                        INX  H
0117 EB                        XCHG
0118 E3                        XTHL
0119 0600                      MVI  B,0
011B CD8100                    CALL GETCH       *Make copy
011E 3E20                      MVI  A,' '       *But ' ' means
0120 B9                        CMP  C           *erase old one
0121 C22901                    JNZ  CD3         *It's not ' '
0124 1B                        DCX  D           *It is ' '
0125 1B                        DCX  D           *erase the '='
0126 12                        STAX D
0127 1B                        DCX  D           *and the ch
0128 12                        STAX D
0129 CD5300             CD3    CALL CMD         *Continue scan
012C D1                        POP  D           *for command
012D 06FF                      MVI  B,0FFH      *Restore B
012F C9                        RET
                        *
                        *   *** ( ***
                        *
                        * ([command]....)
                        *
                        * Any number of commands can be grouped
                        * together to be treated as one command
                        * by a pair of parentheses.  This can
                        * also be nested.
                        *
0130 CD8100             CMDLP  CALL GETCH
0133 79                        MOV  A,C
0134 FE29                      CPI  ')'
0136 C8                        RZ
0137 CD5300                    CALL CMD
013A C33001                    JMP  CMDLP
                        *
                        *   *** ? ***
                        *
                        * ?[then command][else command]
                        *
                        * This command tells the robot to
                        * randomly pick between the [then
                        * command] and the [else command].
                        *
                        * RNDB is a random bit generator.
                        *
013D CA5C01             CMDQM  JZ   SKIP         *Non-execute
0140 CD4601                    CALL RNDB        *Execute
0143 C37001                    JMP  TEST
0146 E5                 RNDB   PUSH H           *Random bit, see
0147 2AFE1F                    LHLD SEED        *Tausworthe,
014A 29                        DAD  H           *Math. Comp. 19
014B 7D                        MOV  A,L         *(1965) 201-209
014C D25201                    JNC  CQ1
014F EE2D                      XRI  02DH        *Watson et.al.,
0151 6F                        MOV  L,A         *Math. Comp. 16
0152 22FE1F             CQ1    SHLD SEED        *(1962) 368-369
0155 E1                        POP  H
0156 E601                      ANI  1
0158 C9                        RET
                        *
                        *   *** T ***
                        *
                        * T[then command][else command]
                        *
                        * This command tests the robot's acc.
                        * If it is not 0, then execute the
                        * [then command], else execute the
                        * [else command]
                        *
                        * The SKIP and the TEST entries can
                        * be used by other commands of the
                        * same syntax.
                        *
0159 C26201             CMDT   JNZ  CT1          *Execute
015C CD5000             SKIP   CALL RCMD         *Non-execute
015F C35000                    JMP  RCMD
0162 E5                 CT1    PUSH H           *Execute
0163 2A3CCC                    LHLD SCREEN+60   *Get acc.
0166 7C                        MOV  A,H         *Or all 4 digits
0167 B5                        ORA  L
0168 2A3ECC                    LHLD SCREEN+62
016B B4                        ORA  H
016C B5                        ORA  L
016D E1                        POP  H
016E FE30                      CPI  '0'         *All '0'?
0170 CA7E01             TEST   JZ   T1          *Test flag
0173 CD5000                    CALL RCMD        *Do "Then"
0176 0600                      MVI  B,0
0178 CD5000                    CALL RCMD        *Skip "Else"
017B 06FF                      MVI  B,0FFH
017D C9                        RET              *Finished
017E 0600               T1     MVI  B,0         *Otherwise ...
0180 CD5000                    CALL RCMD        *Skip "Then"
```

```
0183 06FF               MVI  B,0FFH
0185 C35000             JMP  RCMD        *Do "Else"
                        *
                        *   *** + *** and *** - ***
                        *
                        * These are single character commands
                        * which adds/subtracts 1 to/from the
                        * accumulator respectively.
                        *
0188 C8         CMDPS   RZ               *Non-execute
0189 E5                 PUSH H           *Execute
018A CDE501             CALL GETA        *Get accumulator
018D E1                 POP  H           *Value of acc
018E 3E27               MVI  A,9999/256  *Is it 9999?
0190 BC                 CMP  H
0191 C29C01             JNZ  CPS1        *No
0194 3E0F               MVI  A,9999 MOD 256
0196 BD                 CMP  L
0197 C29C01             JNZ  CPS1        *No
019A E1                 POP  H           *Yes, don't add
019B C9                 RET
019C 23         CPS1    INX  H           *Not 9999, add
019D 0E08       PUTA    MVI  C,8         *Put acc back
019F 29         P1      DAD  H           *Binary->decimal
01A0 7C                 MOV  A,H
01A1 D664               SUI  100         *Divide by 100
01A3 DAA801             JC   P2
01A6 67                 MOV  H,A
01A7 23                 INX  H
01A8 0D         P2      DCR  C
01A9 C29F01             JNZ  P1          *Divide loop
01AC 4C                 MOV  C,H         *Remainder
01AD 7D                 MOV  A,L         *Quotient
01AE 213CCC             LXI  H,SCREEN+60 *Hi-order digit
01B1 CDBB01             CALL TWOD        *Put 2 digits
01B4 23                 INX  H
01B5 79                 MOV  A,C         *Low order
01B6 CDBB01             CALL TWOD        *2 more digits
01B9 E1                 POP  H
01BA C9                 RET
01BB 362F       TWOD    MVI  M,'0'-1     *One before '0'
01BD 34         P3      INR  M           *Make it '0'
01BE D60A               SUI  10
01C0 D2BD01             JNC  P3          *Count in 10's
01C3 C63A               ADI  '0'+10
01C5 23                 INX  H
01C6 77                 MOV  M,A         *and 1's
01C7 C9                 RET
01C8 C8         CMDMS   RZ               *Non-execute
01C9 E5                 PUSH H           *Execute
01CA CDE501             CALL GETA        *Get acc
01CD E1                 POP  H
01CE 7C                 MOV  A,H         *Is it 0?
01CF B5                 ORA  L
01D0 2B                 DCX  H
01D1 C29D01             JNZ  PUTA        *No, do -1
01D4 E1                 POP  H           *Yes, don't -1
01D5 C9                 RET
                        *
                        *   *** A ***
                        *
                        * A[repeated command]
                        *
                        * This command repeats the [repeated
                        * command] A times.  A is the current
                        * value of the robot's accumulator.
                        *
01D6 CA5000     CMDA    JZ   RCMD        *Non-execute
01D9 E5                 PUSH H           *Execute
01DA CDE501             CALL GETA        *Get acc
01DD E1                 POP  H           *Value of acc
01DE E3                 XTHL             *Old HL restored
01DF CD8100             CALL GETCH       *Next character
01E2 C32302             JMP  ITER        *Go repeat it
                        *
                        *   *** GETA *** & *** NUMB ***
                        *
                        * Subroutines to get the value of acc/
                        * numbers from input.
                        * HL points to where numbers are from.
                        * Value is returned on top of the
                        * stack. (Yes, top of the stack.)
                        *
                        * NUMB assumes the first digit is
                        * already in C reg.
                        *
01E5 213CCC     GETA    LXI  H,SCREEN+60 *Hi-order digit
01E8 CD8100             CALL GETCH       *of accumulator
01EB E5         NUMB    PUSH H           *Save old HL
01EC 210000             LXI  H,0         *HL=value now
01EF D5         N1      PUSH D           *Decimal->binary
01F0 3E18               MVI  A,24
01F2 BC                 CMP  H           *H too big?
01F3 D2F701             JNC  N2          *No, OK
01F6 67                 MOV  H,A         *Yes, set it
01F7 54         N2      MOV  D,H         *Save HL in DE
01F8 5D                 MOV  E,L
01F9 29                 DAD  H           *to do *10
01FA 29                 DAD  H
01FB 19                 DAD  D
01FC 29                 DAD  H           *Got it
01FD 1600               MVI  D,0
01FF 79                 MOV  A,C         *C = next digit
0200 D630               SUI  '0'
0202 5F                 MOV  E,A
0203 19                 DAD  D           *Add next digit
0204 D1                 POP  D
0205 E3                 XTHL             *Old HL back
0206 CD8100             CALL GETCH
0209 CDDC00             CALL DIGIT       *More digits?
020C E3                 XTHL             *HL=value again
020D CAEF01             JZ   N1          *Yes, continue
0210 33                 INX  SP          *No, dig out
0211 33                 INX  SP          *return addr.
0212 E3                 XTHL             *Put value there
0213 3B                 DCX  SP          *Bury it
0214 3B                 DCX  SP
0215 E3                 XTHL             *Get old HL back
```

```
0216 C9                 RET
                        *
                        *   *** number ***
                        *
                        * [number][repeated command]
                        *
                        * This command will repeat the [repeated
                        * command] any number of times as
                        * specified.
                        *
                        * The entry ITER may be called by other
                        * handlers of the same syntax.
                        *
0217 CDEB01     CMD09   CALL NUMB          *Get value
021A 78                 MOV  A,B          *Execute?
021B B7                 ORA  A
021C C22302             JNZ  ITER         *Yes, iterate
021F F1                 POP  PSW          *No, pop value
0220 C35300             JMP  CMD          *Skip command
0223 E3         ITER    XTHL              *Iterate
0224 7C                 MOV  A,H          *HL has value
0225 B5                 ORA  L
0226 2B                 DCX  H
0227 C23302             JNZ  IT1          *Not 0, go on
022A E1                 POP  H            *It's 0
022B 0600               MVI  B,0          *Skip command
022D CD5300             CALL CMD
0230 06FF               MVI  B,0FFH       *Restore B
0232 C9                 RET
0233 E3         IT1     XTHL              *Iterations
0234 E5                 PUSH H
0235 C5                 PUSH B
0236 CD5300             CALL CMD
0239 C1                 POP  B
023A E1                 POP  H
023B C32302             JMP  ITER
                        *
                        *   *** TABL ***
                        *
                        * This is a table of all reserved
                        * chracters (except digits, which are
                        * also reserved).  Each reserved char.
                        * is followed by the addr. of the
                        * routine that handles the character.
                        *
                        * TBLSZ is the number of entries in
                        * the table.
                        *
023E 20         TABL    DB   ' '          *No-operation
023F EC00               DW   CMDBL
0241 44                 DB   'D'          *Define
0242 ED00               DW   CMDD
0244 28                 DB   '('          *Begin
0245 3001               DW   CMDLP
0247 29                 DB   ')'          *End
0248 2400               DW   DIRECT
024A 7F                 DB   @DEL         *Abort
024B 2400               DW   DIRECT
024D 3F                 DB   '?'          *Random choice
024E 3D01               DW   CMDQM
0250 54                 DB   'T'          *Test acc.
0251 5901               DW   CMDT
0253 2B                 DB   '+'          *Add one
0254 8801               DW   CMDPS
0256 2D                 DB   '-'          *Subtract one
0257 C801               DW   CMDMS
0259 41                 DB   'A'          *Accumulator
025A D601               DW   CMDA
            000A    TBLSZ   EQU  (*-TABL)/3
                    *
                    *   *** INIT ***
                    *
                    * Initilize the robot.
                    *
025C C9             INIT    RET           *Baptize
            025D    LAST    EQU  *        *the robot
                            END
```

OBJECT CODE OF "THE MIND"

```
0000 :  3100 20CD  5C02 AFD3  C821 00CC  3ED0 3620
0010 :  23BC C20E  0021 413D  223A CC21  3030 223C
0020 :  CC22 3ECC  31FE 1F21  00CC 063A  3620 2305
0030 :  C22C 000E  7FCD AE00  CA2A 0060  6811 00CC
0040 :  CD50 0021  00CC 5058  06FF CD50  00C3 2700
0050 :  CD81 00DB  00E6 80C2  2400 E5CD  C600 78C2
0060 :  6500 B7E3  C9B7 E1C8  E5CD AE00  C27F 00E5
0070 :  2163 FD39  7CB7 FA24  00E1 2323  CD50 00E1
0080 :  C97B E63F  FE39 D224  00B2 CA90  003E 7F12
0090 :  7CB5 C2A5  00CD 4601  DB00 E680  CA95 00DB
00A0 :  014F C3A7  004E 237A  B3C8 7912  13C9 D5C5
00B0 :  2140 CC11  4000 060F  79BE CAC3  0019 05C2
00C0 :  B900 04C1  D1C9 C506  0A21 3E02  79BE 23CA
00D0 :  E600 2323  05C2 CD00  C121 1702  3E39 B9D8
00E0 :  79D6 30D8  BFC9 7E23  666F C1C9  C9CD 8100
00F0 :  E5CD C600  CA24 0078  B7C2 0001  E1C3 5000
0100 :  CDAE 00C2  0801 367F  C50E 20CD  AE00 C224
0110 :  00C1 7123  363D 23EB  E306 00CD  8100 3E20
0120 :  B9C2 2901  1B1B 121B  12CD 5300  D106 FFC9
0130 :  CD81 0079  FE29 C8CD  5300 C330  01CA 5C01
0140 :  CD46 01C3  7001 E52A  FE1F 297D  D252 01EE
0150 :  2D6F 22FE  1FE1 E601  C9C2 6201  CD50 00C3
0160 :  5000 E52A  3CCC 7CB5  2A3E CCB4  B5E1 FE30
0170 :  CA7E 01CD  5000 0600  CD50 0006  FFC9 0600
0180 :  CD50 0006  FFC3 5000  C8E5 CDE5  01E1 3E27
0190 :  BCC2 9C01  3E0F BDC2  9C01 E1C9  230E 0829
01A0 :  7CD6 64DA  A801 6723  0DC2 9F01  4C7D 213C
01B0 :  CCCD BB01  2379 CDBB  01E1 C936  2F34 D60A
01C0 :  D2BD 01C6  3A23 77C9  C8E5 CDE5  01E1 7CB5
01D0 :  2BC2 9D01  E1C9 CA50  00E5 CDE5  01E1 E3CD
01E0 :  8100 C323  0221 3CCC  CD81 00E5  2100 00D5
01F0 :  3E18 BCD2  F701 6754  5D29 2919  2916 0079
0200 :  D630 5F19  D1E3 CD81  00CD DC00  E3CA EF01
0210 :  3333 E33B  3BE3 C9CD  EB01 78B7  C223 02F1
0220 :  C353 00E3  7CB5 2BC2  3302 E106  00CD 5300
0230 :  06FF C9E3  E5C5 CD53  00C1 E1C3  2302 20EC
0240 :  0044 ED00  2830 0129  2400 7F24  003F 3D01
0250 :  5459 012B  8801 2DC8  0141 D601  C9
```

ASSEMBLER LISTING OF "THE BODY"

```
                        ****************************************
                        *
                        *   The robot's body
                        *
                        * Input port #2 and output port #2
                        * are interfaced to the robot's body.
                        *   IN  2, Bit 7: Sensor flip-flop
                        *   IN  2, Bit 6: Right motor ready
                        *   IN  2, Bit 4: Left motor ready
                        *   OUT 2, Bit 7: Reset sensor flip-flop
                        *   OUT 2, Bit 6: Step right motor
                        *   OUT 2, Bit 5: Right motor direction
                        *   OUT 2, Bit 4: Step left motor
                        *   OUT 2, Bit 3: Left motor direction
                        *
                        * *** Patches ***
                        *
0003                            ORG  00003H
0003 CD9702                     CALL INIT       *New location
0070                            ORG  00070H
0003 2124FD                     LXI  H,STKLMT   *New limit
00C7                            ORG  000C7H
00C7 060F                       MVI  B,TBLSZ    *# of entries
                        *
                        * *** Extend the command table ***
                        *
025C                            ORG  0025CH
025C 46                         DB   'F'        *Forward robot
025D 6B02                       DW   CMDF
025F 42                         DB   'B'        *Backward
0260 7002                       DW   CMDB
0262 52                         DB   'R'        *Right rotation
0263 7502                       DW   CMDR
0265 4C                         DB   'L'        *Left rotation
0266 7A02                       DW   CMDL
0268 53                         DB   'S'        *Sensor test
0269 8B02                       DW   CMDS
            000F        TBLSZ   EQU  (*-TABL)/3
                        *
                        *   *** F, B, R, & L ***
                        *
                        * These are single character commands to
                        * move the robot.
                        *
026B 3E0A               CMDF    MVI  A,00AH     *Forward robot
026D C37C02                     JMP  DOIT
0270 3E1E               CMDB    MVI  A,01EH     *Backward
0272 C37C02                     JMP  DOIT
0275 3E0E               CMDR    MVI  A,00EH     *Right rotation
0277 C37C02                     JMP  DOIT
027A 3E1A               CMDL    MVI  A,01AH     *Left rotation
027C C8                 DOIT    RZ              *Non-execute
027D F5                         PUSH PSW        *Execute
027E DB02               DI1     IN   2
0280 E60A                       ANI  00AH       *Both motors
0282 FE0A                       CPI  00AH       *ready?
0284 CA7E02                     JZ   DI1        *No
0287 F1                         POP  PSW        *Yes
0288 D302                       OUT  2          *Do it
028A C9                         RET
                        *
                        *   *** S ***
                        *
                        * S[then command][else command]
                        *
                        * This command tests the robot's sensor.
                        * If sensor is set, then execute the
                        * [then command], else execute the
                        * [else command]
                        *
028B CA5C01             CMDS    JZ   SKIP       *Non-execute
028E DB02                       IN   2          *Execute
0290 E601                       ANI  001H       *Sensor bit
0292 D302                       OUT  2          *Reset it
0294 C37001                     JMP  TEST
                        *
                        *   *** INIT ***
                        *
                        * Initilize the robot.
                        *
0297 3E01               INIT    MVI  A,001H     *Baptize
0299 D302                       OUT  2          *the robot
029B C9                         RET
            029C        LAST    EQU  *
                                END
```

OBJECT CODE OF "THE BODY"

```
0003 : CD97 02
0070 : 2124 FD
00C7 : 060F
025C : 466B 0242
0260 : 7002 5202  4C7A 02B1  538B 023E  0AC3 7C02
0270 : 3E1E C37C  023E 0EC3  7C02 3E1A  C8F5 DBE6
0280 : 0AFE 0AFD  CA7E 02F1  D302 C9CA  5C01 DB02
0290 : E601 D302  C370 013E  01D3 02C9
```

ASSEMBLER LISTING OF "ANOTHER BODY"

```
                        ****************************************
                        *
                        *   Another Robot's Body
                        *
                        * Here the DAZZLER board and a vidio
                        * display are used as the playgrounds of
                        * another robot.  The last 2K of the 8K
                        * RAM is used by the DAZZLER.
                        *
            17F9        STACK   EQU  017F9H      *Make room
            17F9        SEED    EQU  STACK       *for DZL etc.
            17FB        BORW    EQU  017FBH      *Black or white
            17FC        X       EQU  017FCH      *Robot position
            17FD        Y       EQU  017FDH
            17FE        DX      EQU  017FEH      *& direction
            17FF        DY      EQU  017FFH
```

```
                1800    DZL     EQU  01800H       *DAZZLER RAM
                        *
                        * *** Patches ***
                        *
0000                            ORG  00000H
0000 31FB17             DIRECT  LXI  SP,SEED+2    *New location
0003 CD2103                     CALL INIT
0024                            ORG  00024H
0024 31F917             DIRECT  LXI  SP,STACK
0070                            ORG  00070H
0070 2191FC                     LXI  H,STKLMT     *New limit
00C7                            ORG  000C7H
00C7 0611                       MVI  B,TBLSZ      *# of entries
0147                            ORG  00147H
0147 2AF917                     LHLD SEED         *New location
0152                            ORG  00152H
0152 22F917             CQ1     SHLD SEED
                        *
                        * *** Extend the command table ***
                        *
025C 43                         DB   'C'          *Clear screen
025D 7102                       DW   CLEAR
025F 42                         DB   'B'          *Black trace
0260 8902                       DW   BLACK
0262 57                         DB   'W'          *White trace
0263 8F02                       DW   WHITE
0265 48                         DB   'H'          *Go home
0266 9602                       DW   HOME
0268 4E                         DB   'N'          *Face North
0269 9F02                       DW   NORTH
026B 52                         DB   'R'          *Right 45 degr.
026C A902                       DW   RIGHT
026E 46                         DB   'F'          *Forward 1 step
026F CB02                       DW   FORWRD
                0011    TBLSZ   EQU  (*-TABL)/3
                        *
                        * *** C *** Clear the screen
                        *
0271 C8                 CLEAR   RZ                *Non-execute
0272 E5                         PUSH H            *Execute
0273 C5                         PUSH B
0274 210018                     LXI  H,DZL
0277 010008                     LXI  B,2048
027A 3AFB17             CL1     LDA  BORW         *Black or white?
027D 2F                         CMA
027E 77                         MOV  M,A
027F 23                         INX  H
0280 0B                         DCX  B
0281 78                         MOV  A,B
0282 B1                         ORA  C
0283 C27A02                     JNZ  CL1
0286 C1                         POP  B
0287 E1                         POP  H
0288 C9                         RET               *Finished
                        *
                        * *** B *** Black trace
                        *
0289 C8                 BLACK   RZ                *Non-execute
028A AF                         XRA  A            *Execute
028B 32FB17                     STA  BORW
028E C9                         RET
                        *
                        * *** W *** White trace
                        *
028F C8                 WHITE   RZ                *Non-execute
0290 3EFF                       MVI  A,0FFH       *Execute
0292 32FB17                     STA  BORW
0295 C9                         RET
                        *
                        * *** H *** Go home
                        *
0296 C8                 HOME    RZ                *Non-execute
0297 E5                         PUSH H            *Execute
0298 D5                         PUSH D
0299 214040                     LXI  H,04040H     *Center position
029C C3DB02                     JMP  DISP         *Display it
                        *
                        * *** N *** Face North
                        *
029F C8                 NORTH   RZ                *Non-execute
02A0 AF                         XRA  A            *Execute
02A1 32FE17                     STA  DX
02A4 3C                         INR  A
02A5 32FF17                     STA  DY
02A8 C9                         RET
                        *
                        * *** R *** Rotate right 45 degrees
                        *
02A9 C8                 RIGHT   RZ                *Non-execute
02AA E5                         PUSH H            *Execute
02AB 2AFE17                     LHLD DX           *Old direction
02AE 7C                         MOV  A,H          *DX'=(DY
02AF 85                         ADD  L            *+DX)
02B0 CDC002                     CALL R1           */SQRT(2)
02B3 4F                         MOV  C,A          *New DX
02B4 7C                         MOV  A,H          *DY'=(DY
02B5 95                         SUB  L            *+DX)
02B6 CDC002                     CALL R1           */SQRT(2)
02B9 67                         MOV  H,A          *New DY
02BA 69                         MOV  L,C          *New DX
02BB 22FE17                     SHLD DX           *Save them
02BE E1                         POP  H
02BF C9                         RET
02C0 FE02               R1      CPI  2            *Subroutine
02C2 C2C602                     JNZ  R2           *to divide
02C5 3D                         DCR  A            *by SQRT(2)
02C6 FEFE               R2      CPI  -2           *approximately
02C8 C0                         RNZ
02C9 3C                         INR  A
02CA C9                         RET
                        *
                        * *** F *** Forward 1 step
                        *
02CB C8                 FORWRD  RZ                *Non-execute
02CC E5                         PUSH H            *Execute
02CD D5                         PUSH D
02CE 2AFC17                     LHLD X            *Old position
02D1 EB                         XCHG
```

```
02D2 2AFE17            LHLD DX          *Direction
02D5 7C                MOV  A,H
02D6 82                ADD  D
02D7 67                MOV  H,A         *New Y
02D8 7D                MOV  A,L
02D9 83                ADD  E
02DA 6F                MOV  L,A         *New X
02DB 22FC17      DISP  SHLD X           *Save XY &
02DE 7C                MOV  A,H         *display
02DF 2F                CMA              *Y is up
02E0 07                RLC              *Map XY to RAM
02E1 07                RLC
02E2 67                MOV  H,A
02E3 07                RLC
02E4 E6F0              ANI  0F0H
02E6 5F                MOV  E,A
02E7 7D                MOV  A,L
02E8 0F                RRC
02E9 0F                RRC
02EA E60F              ANI  00FH
02EC B3                ORA  E
02ED 5F                MOV  E,A         *Half done
02EE 7C                MOV  A,H
02EF 17                RAL
02F0 17                RAL
02F1 E605              ANI  005H
02F3 57                MOV  D,A
02F4 7D                MOV  A,L
02F5 07                RLC
02F6 07                RLC
02F7 07                RLC
02F8 E602              ANI  002H
02FA B2                ORA  D
02FB C618              ADI  DZL/256     *Finally!
02FD 57                MOV  D,A         *DE->RAM
02FE 7D                MOV  A,L         *Find the bit
02FF E603              ANI  003H
0301 6F                MOV  L,A
0302 7C                MOV  A,H
0303 E604              ANI  004H
0305 B5                ORA  L
0306 C619              ADI  TAB MOD 256
0308 6F                MOV  L,A         *from a table
0309 2603              MVI  H,TAB/256
030B 3AFB17            LDA  BORW        *Black or white
030E A6                ANA  M           *Pick the bit
030F 4F                MOV  C,A
0310 7E                MOV  A,M
0311 2F                CMA              *Reverse mask
0312 EB                XCHG             *HL->DZL RAM
0313 A6                ANA  M           *Reverse mask
0314 B1                ORA  C           *Pick the bit
0315 77                MOV  M,A         *Back to RAM
0316 D1                POP  D
0317 E1                POP  H
0318 C9                RET
0319 01          TAB   DB   001H        *Bit map of
031A 02                DB   002H        *DAZZLER
031B 10                DB   010H
031C 20                DB   020H
031D 04                DB   004H
031E 08                DB   008H
031F 40                DB   040H
0320 80                DB   080H
                       *
                       * *** INIT ***
                       *
0321 3E8C        INIT  MVI  A,080H+(DZL/512)
0323 D30E              OUT  00EH        *Turn on DZL
0325 3E6D              MVI  A,06DH      *Black & white
0327 D30F              OUT  00FH        *2K Hi res.
0329 CD9002            CALL WHITE+1
032C C3A002            JMP  NORTH+1
           032F  LAST  EQU  *
                       END
```

OBJECT CODE OF "ANOTHER BODY"

```
0000 :  31FB 17CD  2103
0024 :  31F9 17
0070 :  2191 FC
00C7 :  0611
0147 :  2AF9 17
0152 :  22F9 17
025C :  4371 0242
0260 :  8902 578F  0248 9602  4E9F 0252  A902 46CB
0270 :  02C8 E5C5  2100 1801  0008 3AFB  172F 7723
0280 :  0B78 B1C2  7A02 C1E1  C9C8 AF32  FB17 C9C8
0290 :  3EFF 32FB  17C9 C8E5  D521 4040  C3DB 02C8
02A0 :  AF32 FE17  3C32 FF17  C9C8 E52A  FE17 7C85
02B0 :  CDC0 024F  7C95 CDC0  0267 6922  FE17 E1C9
02C0 :  FE02 C2C6  023D FEFE  C03C C9C8  E5D5 2AFC
02D0 :  17EB 2AFE  177C 8267  7D83 6F22  FC17 7C2F
02E0 :  0707 6707  E6F0 5F7D  0F0F E60F  B35F 7C17
02F0 :  17E6 0557  7D07 0707  E602 B2C6  1857 7DE6
0300 :  036F 7CE6  04B5 C619  6F26 033A  FB17 A64F
0310 :  7E2F EBA6  B177 D1E1  C901 0210  2004 0840
0320 :  803E 8CD3  0E3E 6DD3  0FCD 9002  C3A0 02
```

****

A Microprocessor Operating System: The Kernel

by Max K Agoston, PhD
Dept of Mathematics
San Jose State University
San Jose, CA 95192

This article describes in detail the kernel for a real-time interactive multiprocess operating system (RIMOS for short). RIMOS was intended to extend the INTEL MDS system and because of this the kernel subroutines make use of INTEL's ISIS. (The INTEL monitor was used only at the start to load ISIS.) However, the source code that will be listed later is basically usable on any computer using the INTEL 8080 microprocessor, except that one would have to write a file system and make some simple changes to eliminate reference to ISIS.

An Overview of the Total System

The basic structure of RIMOS was derived largely from concepts for operating systems as discussed in [2] and [3]. RIMOS was to consist of a hierarchy of seven levels. Each level would make use of the operations and data structures made available to it by the level which was immediately below it but would not concern itself with their implementation. Levels 1 and 2 correspond to what I shall call the kernel, whose subroutines are listed in Table 1. The purpose of the kernel is to create an environment in which program execution and I/O operations are handled uniformly as cooperating concurrent processes.

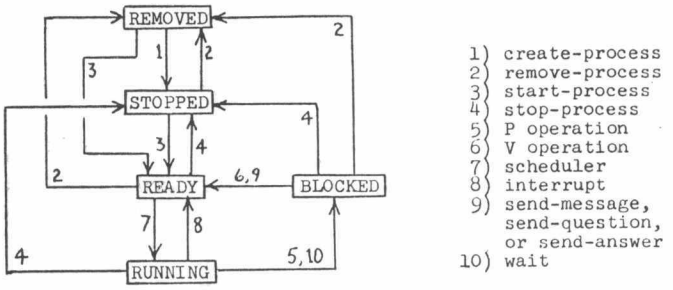
Level 0:
The hardware or basic machine.

Level 1:
This level consists of eleven routines and the appropriate data structure. There are eight interrupt-handlers which service each of the eight hardware generated interrupts.

The scheduler is a routine that starts processes executing and also determines their order of execution. In general, the order is determined by their priority and their position in the queue of active processes. Processes with lower priority do not get to execute until all higher priority processes have finished. Processes of the same priority are serviced in a round-robin fashion with each process being allotted a fixed length of time to run.

Finally, the P and V synchronization primitives are routines which provide the operating system with a mechanism for preventing processes from interfering with each other in critical sections.

Level 2:
This level consists of eight basic routines dealing with the primitive operations on processes and the messages between them (see Figure 1). Figure 1 shows the states of a process and how the routines transform into the other. It should be pointed out that even though interrupts are not disabled these routines are essentially "non-interruptable" in the sense that if any process is executing one of them then no other process can execute any of them, since any such attempt will get it stuck on a semaphore.



The states of a process and the primitives that cause transitions between them

FIGURE 1

The create-process routine loads a process from the disk using ISIS and creates for the operating system a record or Process Control Block (PCB) of this process.

The start-process routine takes a stopped process and inserts it into its appropriate place in the queue of active PCBs depending on its priority. If the routine is asked to start a process which does not exist yet, it will create the process first and then proceed as before.

The stop-process routine removes a process from the queue of active processes and puts it into a "stopped" state. Nothing else is changed though, so that if the process is later restarted, then it will begin exactly where it left off.

The remove-process routine returns the message system resources of a process Q to the operating system as appropriate. If Q is not a "permanent resident", then its PCB is retained but flagged as being a "program". The difference between a stopped process and a program is that the former can still receive messages, whereas the latter cannot.

The send-message and send-question routines enable a process to send a message to another process. In the case of the latter, an answer is returned eventually via the send-answer routine. Each process is allowed to send a fixed number of messages or questions. An attempt to exceed this number will return an error message.

The wait routine allows a process Q to wait for "data". It provides Q with the capability of specifying exactly what it wants to wait for, whether it is to be a message, question, answer, or interrupt.

Level 3:
This level would consist of the shared TTY and CRT process calls on the I/O driver processes. Processes and their interaction are now seen from a global point of view.

Level 4:
The user processes.

Level 5:
The job control language, which is the interface between the virtual machine of the user and the internal machine. The interpreters for this language are processes on level 4.

Level 6:
The user.

Although the whole operating system RIMOS did not get finished, with the exception of the interrupt-handlers, all of the routines of the kernel (see Table 1) have been implemented and tested. The kernel takes up approximately 1800 bytes of memory and the 8080 assembly language code for it is listed at the end of this documentation. As one can see from the list of user callable subroutines in Table 1, the kernel provides the user with a lot of power, especially with its message system.

The Kernel Data Structure

In order to understand how the subroutines of the kernel are implemented one must understand the underlying data structure. This is described next.

THE SUBROUTINES	DESCRIPTION OF PARAMETERS
Level 1:	
Interrupt handlers	
Scheduler	ID: indicates local or global semaphore
P P(ID, SEM)	SEM: address of semaphore
V V(SEM)	NAME: address of process's name PRIORITY: specifies priority MESSAGE ANSWER: 2 bytes each
Level 2:	ADDRESS: address of 3 byte block – 2 bytes for the question and 1 byte for identification purposes
Create-process RESULT := CREAT(NAME)	BID: an integer used for identification purposes.
Start-process RESULT := START(NAME, PRIORITY)	WTBLK: address of a 10 byte block whose contents are returned by the routine and which describe the data that was received
Stop-process RESULT := STOP(NAME)	MASK: 1 byte binary word which specifies the data that one is waiting for
Remove-process RESULT := REMOV(NAME)	RESULT: an integer which is returned and which specifies the error, if any, that occurred while attempting to execute the routine.
Send-message RESULT := SENDM(MESSAGE, NAME)	
Send-question RESULT := SENDQ(ADDRESS,NAME)	
Send-answer RESULT := SENDA(ANSWER,BID)	
Wait RESULT := WAIT(WTBLK, MASK)	

The subroutines of the kernel and their parameters in PL/M notation if intended to be user callable

TABLE 1

GLOBAL POINTERS (each consist of 1 byte):

Free-PCB-pointer (FP) : The free PCBs form a stack using the NP field in the PCB. The FP points to the head of this stack.

All-PCB-pointer (AP) : This pointer points to the first process in the doubly-linked list of all PCBs corresponding to current processes.

Current-process-pointer (CPP) : This pointer points to the PCN of the current running process.

Free-message-buffer-pointer (FMBP) : This pointer points to the first buffer in the stack of free message buffers for all processes.

Priority-i-pointer (PRP [i]) : This is a pointer to the "first" process in the doubly-linked circular list of all active (that is, non-stopped) processes of priority i.

Number-ready-priority-i-processes (NMR [i]) : This byte specifies the number of ready, that is, non-blocked and active processes of priority i.

DEFINITION: A semaphore is a 2 byte block of memory which is divided into the following fields:

VAL (Value): 1 byte which holds in integer 1, $0 \leqslant 1 < 256$.

FWP (First-waiter-pointer) : 1 byte which holds the "index" of the PCB of the first process waiting on the semaphore are in a queue linked together via the NSWPs.

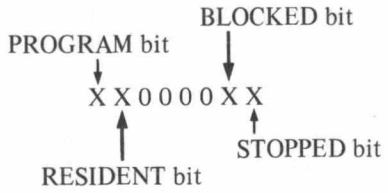
The global semaphore used by the kernel (2 bytes).

All-PCB-semaphore (AS) : This semaphore is used not only to prevent more than one process from tinkering with any one of the doubly-linked PCB lists, but also to allow only one process to execute any one of the level 2 routines at a time. Next, each process has a unique Process Control Block (PCB) associated to it which specifies the process completely:

DEFINITION: A PCB consists of 37 bytes of memory which are divided into the following fields:

NAME (NAM) : 6 bytes holding up to 6 alphanumeric characters with a blank after the last character if there are less than 6.

Execution-state (EXC) : 1 byte whose relevant bits are shown below:



bit	interpretation when set
PROGRAM	a permanent resident program which presently is not a process
RESIDENT	a system program whose code is never removed from memory
BLOCKED	process is waiting on a semaphore
STOPPED	process is in the "stopped" state

Priority (PR) : 1 byte specifying the priority of this process.

0 clock
1 I/O drivers
2 command interpreter
3 real-time user
4 background processes

Message-buffer-counter (MBC) : (1 byte) This field is initially set to a fixed number MBPP and specifies the number of buffers currently available to this process for messages.

Waiting-data-mask (WDMK) : 1 byte which specifies what type of "data" this process is waiting for because of a call to the wait routine. The bits have the same interpretation as for the parameter MASK in that routine.

Received-data-mask (RDMK) : 1 byte whose first three bits show what type of "data" is waiting to be processed by this routine. The bits have the same interpretation as for returned value EVENT in the wait routine.

First-message-pointer (FMP) : 1 byte which points to the first message in the message queue.

First-question-pointer (FQP) : 1 byte which points to the first question in the question queue.

First-answer-pointer (FAP) : 1 byte which points to the first answer in the answer queue.

Answer-buffer-pointer (ABP) : 1 byte which points to the queue of buffers that are to be used when sending back answers to questions asked by other processes. This queue is created as a protection for the system, so that the only buffers used are those which were actually intended for holding answers.

Next-semaphore-waiter-pointer (NSWP) : 1 byte which is used to link together processes that are waiting on a particular semaphore. The semaphore itself contains a pointer which points to the first process in the queue of waiting processes.

Semaphore-pointer (SEMP) : 1 byte which indicates whether or not the process is waiting on a semaphore. Its

value has the following interpretation:

(SEMP)	Semaphore on which process waits
0	none or local semaphore which is private to process
2	AS

Last-active-PCB-pointer (LAP) : (1 byte) Whenever this process is not STOPPED, this byte holds the "index" of the preceding PCB in the queue of all PCBs.

Next-active-PCB-pointer- (NAP) : (1 byte) Whenever this process is not STOPPED, this byte holds the "index" of the next PCB in the doubly-linked circular list of active processes with the same priority as this process.

Last-PCB-pointer (LP) : 1 byte which holds the "index" of the preceding PCB in the queue of all PCBs.

Next-PCB-pointer (NP) : 1 byte which holds the "index" of the next PCB in the doubly-linked queue of all PCBs.

Data-semaphore (DSM) : (2 bytes) This semaphore, initially set to 0, is used by the wait routine to enable a process to wait for the "data", that is, messages, answers, or interrupts, that it wants.

Starting-address (STAD) : 2 bytes which are the initial starting address of this process.

Stack-pointer (STKP) : 2 bytes which hold the address of the stack for this process.

Temporary-stack (STACK) : 10 bytes which serve as a temporary stack until the user has set up his own. There is space for storing the program location counter (PLC), the program status word (PSW) (which consists of the accumulator and flag bits), and the B, C, D, E, H, and L registers.

NOTE: The PCBs are actually implemented in memory by means of 20 arrays, one for each field. In this way, a PCB, and hence a process, is completely specified by means of a single integer corresponding to the index of the arrays.

Finally, in order to be able to communicate with other processe each process has access to a certain number of message buffers which it can use to pass the messages. If a process sends too many messages and uses up its allotment of buffers, then it will be unable to send any more messages and will have to wait until one of its buffers has been returned. All message buffers are eventually returned to their sender (unless they are sent by mistake to a process which does not expect messages, so that the latter will never act on them); hence, there should not be any problem normally. The routines which handle the sending of messages are described in detail elsewhere.

DEFINITION: A message buffer (MB) consists of a 5 byte block of memory which is divided into the following fields:

Message-buffer-sender-pointer (MBSP) : 1 byte which normally holds the index of the process which sent the buffer. Whenever a process Q is removed via the REMOV routine, this field is checked in all MBs. Those that contain Qs index are set to 0. In this way it is possible to prevent an answer being returned to a process that has long since been removed.

Next-message-buffer-pointer (NMBP) : 1 byte which holds the "index" of the message buffer that is next in the relevant queue.

Answer-status (ASTAT) : 1 byte whose function is to enable the WAIT routine to determine the status of an answer. If the field is non-zero, then this means that this answer is a dummy, that is, the process which was supposed to supply the answer was removed before it could finish its job. (The REMOV routine entered a non-zero value in the answer-status field to indicate this before returning the "answer".)

Message (MESS) : 2 bytes.

NOTE: The MBs (like the PCBs) are also implemented in memory by means of arrays, one for each field. In this way, a MB is completely specified to the system by means of a single integer corresponding to the index of the arrays. These indices are the integers BUFID and BID which are used by the message routines.

ABBREVIATIONS USED

PCB	Process control block
MB	Message buffer
AS	All-PCB-semaphore
DSM	Data-semaphore
FP	Free-PCB-pointer
AP	All-PCB-pointer
CPP	Current-process-pointer
NP	Next-PCB-pointer
LP	Last-PCB-pointer
NAP	Next-active-PCB-pointer
LAP	Last-active-PCB-pointer
NSWP	Next-semaphore-waiter-pointer
STKP	Stack-pointer
FWP	First-waiter-pointer in semaphore
FMBP	Free-message-buffer-pointer
FMP	First-message pointer
FQP	First-question-pointer
FAP	First-answer-pointer
ABP	Answer-buffer-pointer
PRP [i]	Priority-i-pointer
PLC	Program-location-counter
MBC	Message-buffer-counter
WDMK	Waiting-data-mask
RDMK	Received-data-mask
NMPR	Number of possible priorities for processes
MBPP	Number of message buffers allotted to each process
NMR [i]	Number of active non-blocked processes of priority i
STAD	Starting-address
MBSP	Message-buffer-sender-pointer
NMBP	Next-message-buffer-pointer
ASTAT	Answer-status
MESS	Message
NAM	Name-field
PR	Priority-field
EXC	Execution-state
VAL	Value-field of semaphore

active = non-stopped

Initialization

Inititialization of the kernel is an easy task. The initialization routine (which can be overwritten after it is finished) has

to do the following:

1. Load the kernel subroutines. (Record the address of CROO, LBLK, START, and SCH00.)
2. Set the PRP and NMR fields to 0.
3. Set the message buffer links (the NMBP fields).
4. Set the appropriate PCB links (the NP fields).
5. Create those processes which are to be permanent residents. Call these processes PROC1, PROC2, . . . , PROCk. The special entry point CR00 to the CREAT routine has to be used during initialization. The following code accomplishes this step:

```
LXI     H, adri   <-
SHLD    LBLK          \
CALL    CR00           |
 .       .             |  adri is the address of where
 .       .             |  the name "PROCi" is stored
LXI     H, adrk   <-  /
SHLD    LBLK
CALL    CR00
```

6. Call the process which is to be started first PROC. If it has not already been created in step 5, then repeat the procedure of step 5 for PROC.
7. Set the permanent resident bits of processes PROC1, PROC2, . . . , PROCk. This can be accomplished by setting the EXC field of the processes to 41H.
8. Call the START routine for those permanent resident processes that one wants in the START state (see the instructions for using START).
9. Call START for PROC.
10. Get PROC running as follows (assume that PROC was the j'th process created above):

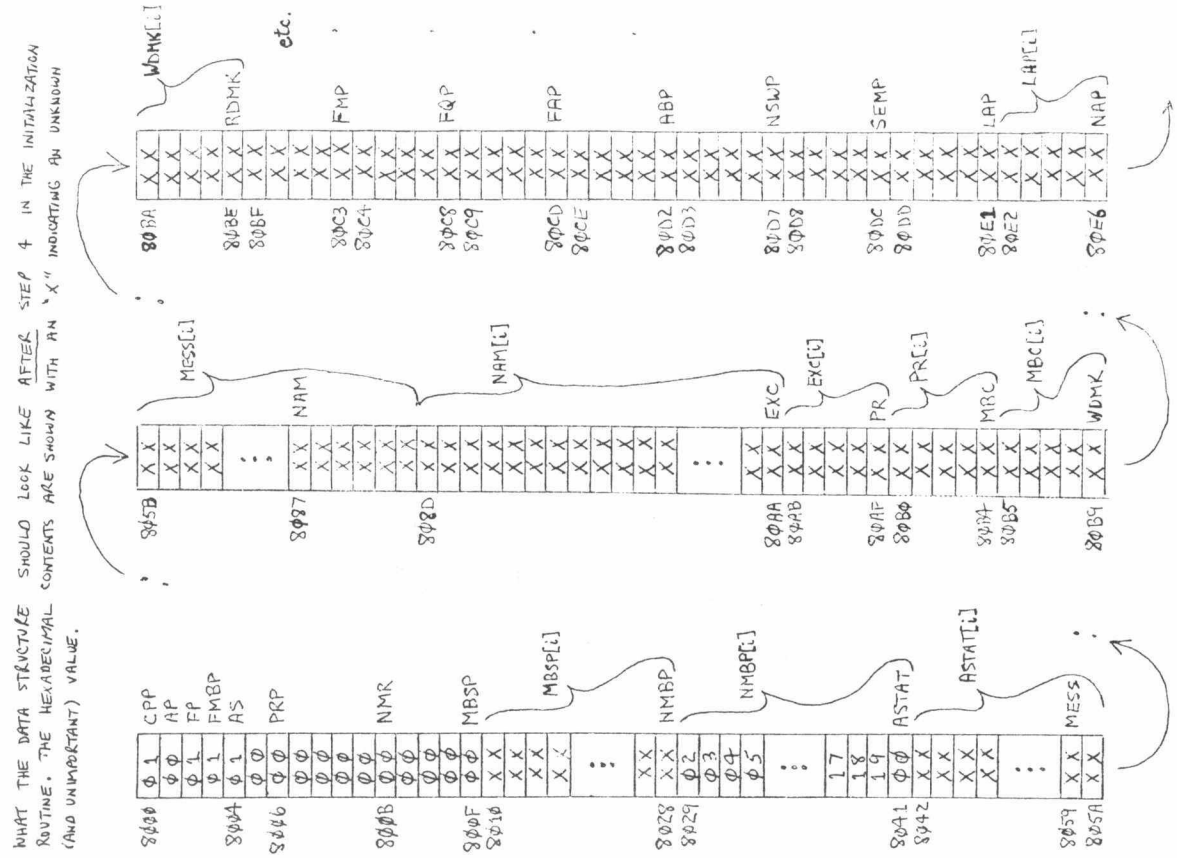
```
MVI     C, j
MVI     B, 0
MVI     A, j
DI
JMP     SCH00
```

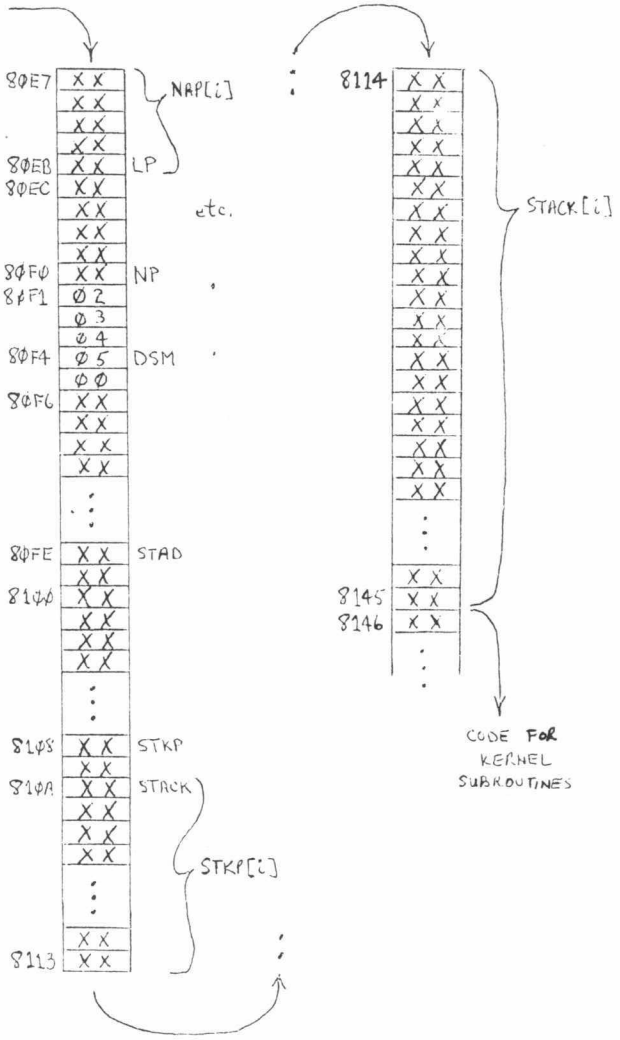
IMPORTANT NOTE: It is essential that the priority of the first process point is at least as high as the priorities of all the processes which are started in step 8 and 9 above (otherwise, some processes will be started running before the initialization routine is finished).

A Short Trace

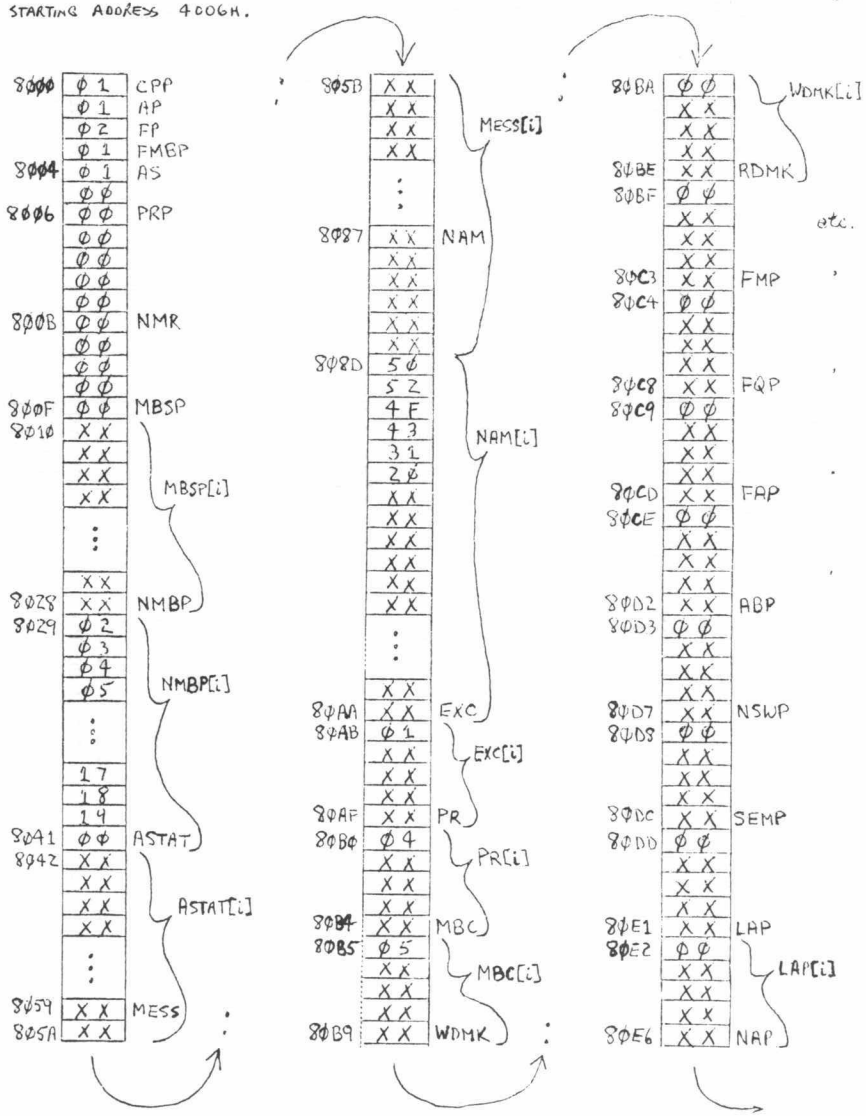
In order to help the reader understand the data structure and initialization better, we give a short trace of the sequence of changes in the data structure during and after initialization in a simple situation. However, keep in mind the general comments about the code which follow below:

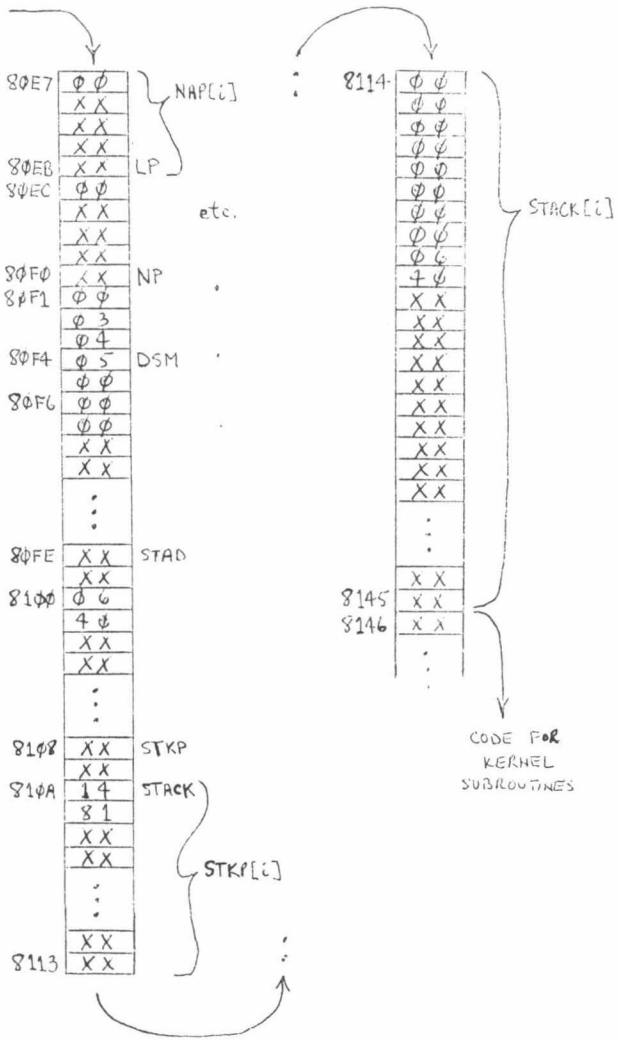
1. Only 5 processes and 5 message buffers per process are allowed, but these numbers can be changed simply by changing the values of NMPCB, MBPP, and NMPR (see the first page of the code).
2. The origin is set arbitrarily at 8000H (see the top of page 4 of my code), but this also can be changed by simply replacing that ORG statement with another.
3. As things stand, after the code is assembled one has to record certain addresses. These are used by the initialization routine and by the user. For one thing, the user would not be able to use the kernel subroutines unless he knows their addresses. One could easily get around this by setting aside some fixed locations in memory to be used as entry points to the appropriate subroutines.

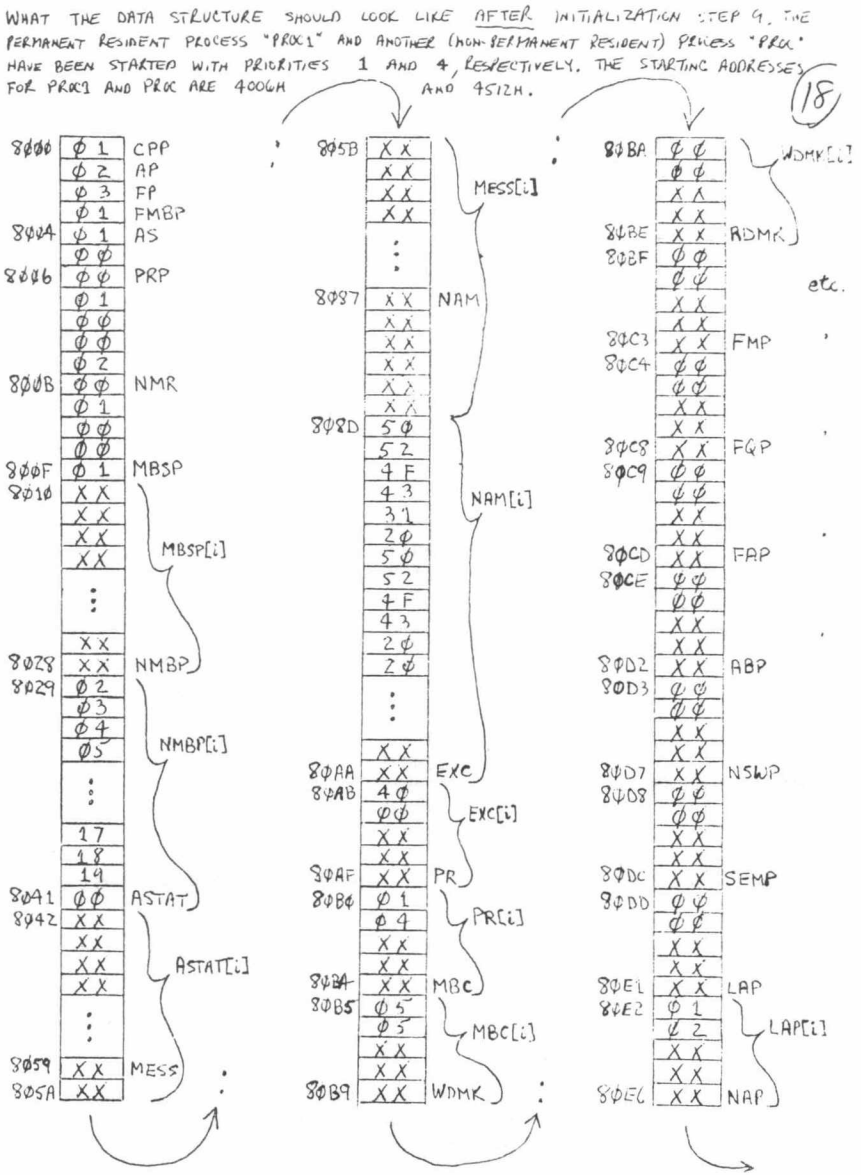


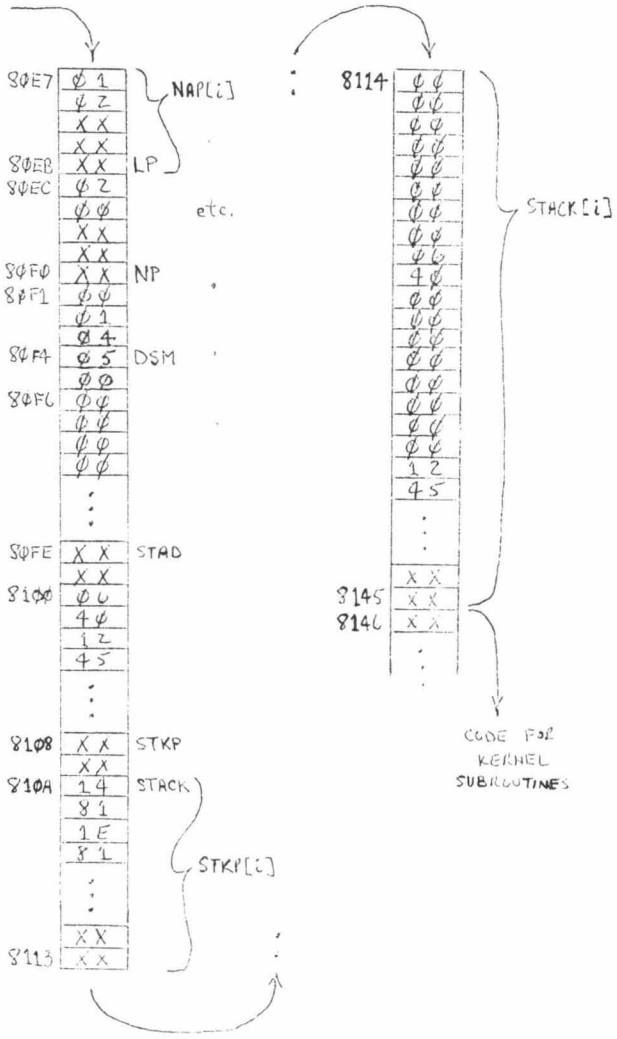


WHAT THE DATA STRUCTURE SHOULD LOOK LIKE AFTER STEP 5 IN THE INITIALIZATION ROUTINE ASSUMING THAT THERE IS TO BE ONE PERMANENT RESIDENT PROCESS "PROC1" WITH STARTING ADDRESS 400H.




8ØE7
NAP[i]
LP
etc.
NP
DSM
STAD
STKP
STACK
STK[i]
8113
8114
STACK[i]
8145
8146
CODE FOR KERNEL SUBROUTINES


WHAT THE DATA STRUCTURE SHOULD LOOK LIKE AFTER INITIALIZATION STEP 4, THE PERMANENT RESIDENT PROCESS "PROC1" AND ANOTHER (NON-PERMANENT RESIDENT) PROCESS "PROC" HAVE BEEN STARTED WITH PRIORITIES 1 AND 4, RESPECTIVELY. THE STARTING ADDRESSES FOR PROC1 AND PROC ARE 4006H AND 4512H.
18
CPP
AP
FP
FMBP
AS
PRP
NMR
MBSP
MBSP[i]
NMBP
NMBP[i]
ASTAT
ASTAT[i]
MESS
MESS[i]
NAM
NAM[i]
EXC
EXC[i]
PR
PR[i]
MBC
MBC[i]
WDMK
WDMK[i]
RDMK
FMP
FQP
FAP
ABP
NSWP
SEMP
LAP
LAP[i]
NAP
etc.



What the data structure should look like *after* PROC has been started running and *after* it has done the following:

1. Started another process PROC3 with priority 4.
   (nm1 and nm3 below are the addresses of where the names "PROC1" and "PROC3" are stored, e.g.,

```
nml:    50
        52
        4F
        43
        31
        20
```

2. Sent 2 messages to PROC1 via

```
LXI     B,0651H
LXI     D,nml
CALL    SENDM
LXI     B,0652H
LXI     D,nml
CALL    SENDM
```

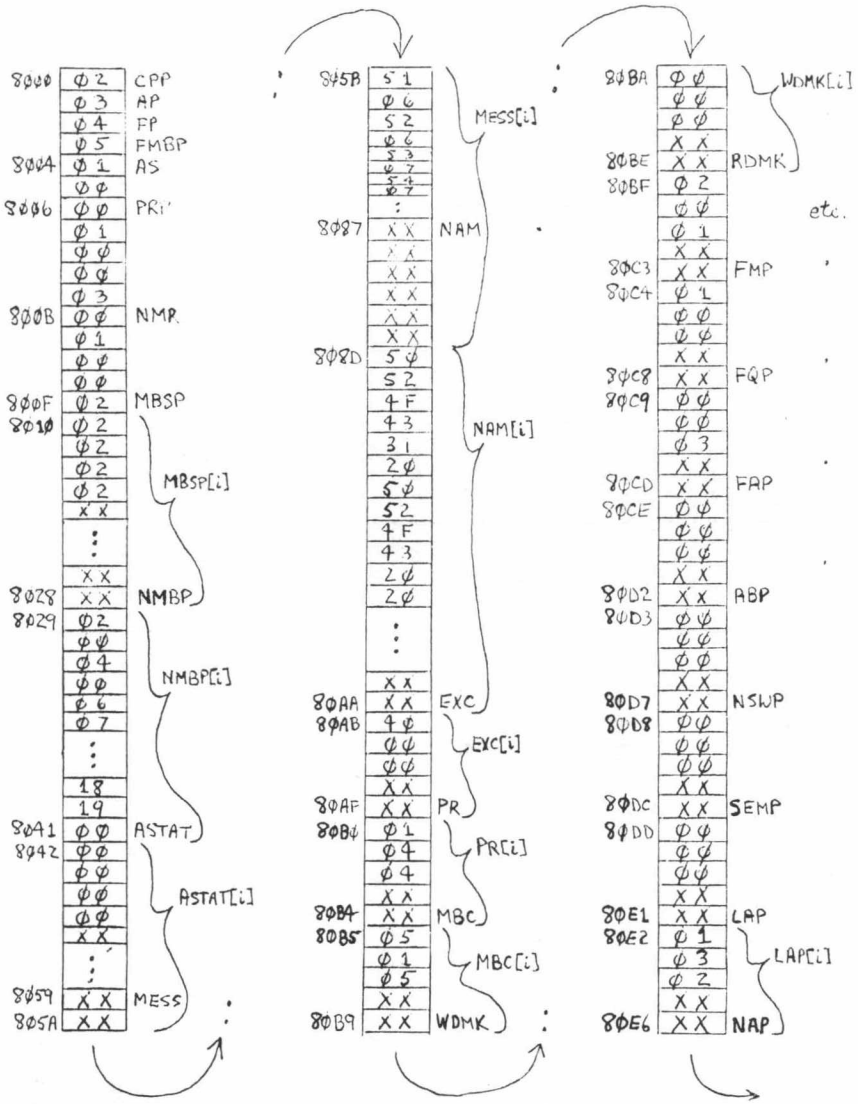
   arbitrary messages

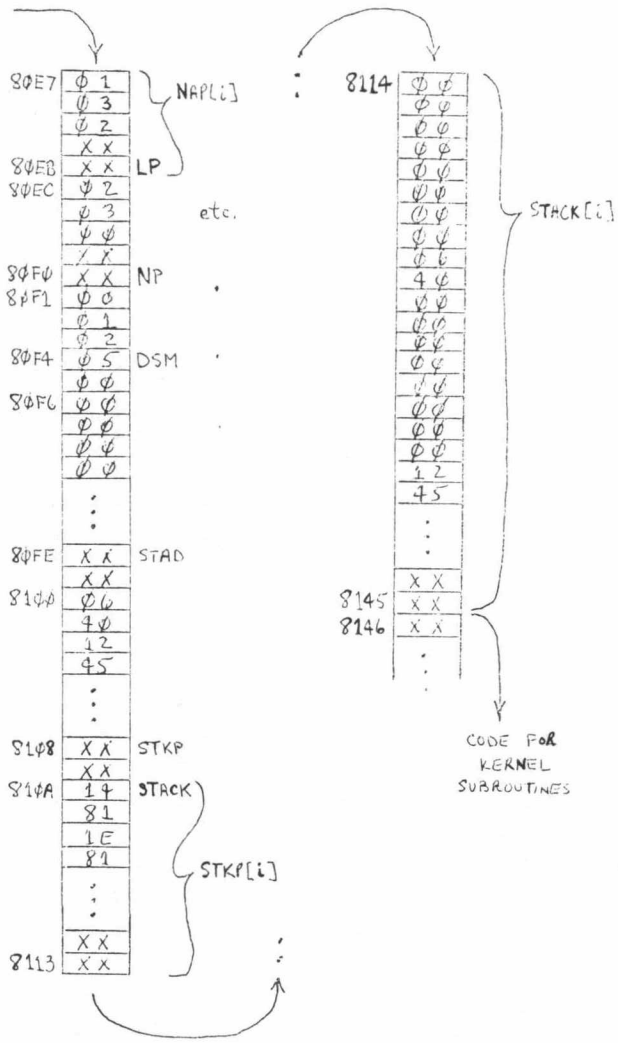
3. Sent 2 questions to PROC3 via

```
LXI     B,QBLK1
LXI     D,nm3
CALL    SENDQ
LXI     B,QBLK2
CALL    SENDQ
```

where

```
QBLK1:  DS      1
        DB      53H,7H
QBLK2:  DS      1
        DB      54H,7H
```





Acknowledgements and References are on the next page.

Complete source listing begins on the page following the Acknowledgements and References.

$25 FOR 6502 EDITOR & ASSEMBLER HEX LISTING & MANUAL

Dear Jim, *Received: 77-Oct-4*

I have just developed the following 6502 software.

MCS 6502 Resident Assembler/Text Editor
ASSM/TED

ASSM/TED. coresident in less than 4K (0200-1200, 0400-1400, 1000-2000, 2000-3000 please specify version desired). Produces relocatable object code on tape and can store executable code in memory during assembly. Using the relocating loader program, one can reload the relocatable object code at practically any location, can assemble from tape or memory. 17 commands are provided which direct the TED to load from tape (LO), assemble (AS), run a program (RU), create a relocatable object file (PU), record text on tape (RE) and other operations.

ASSM has 16 pseudo ops, provides listing consisting of source, object code, and symbol table. The user specifies the upper and lower boundaries of source file and symbol table (practically no limit to number of labels). Labels may be up to 10 characters in length. Text file and symbol table feature variable length storage of each entry in order to conserve memory. 18 error codes pinpoint any errors. TED features auto line numbering, file formating, and a manuscript feature. This software supports up to 2 tape decks, CRT and keyboard, and an optional printer. Shipped preconfigured for TIM based sytems but info is supplied in order to modify for other systems.

Hex listing and operators manual – $25.00.

Carl W Moser
3239 Linda Dr
Winston-Salem, NC 27106

Acknowledgements

I would like to thank C. Jan and particularly J. Richardson for many helpful discussions in the planning stages of this project which was carried out at Stanford University. Richardson's advice on the actual form that the Kernel should take was invaluable and the overall structure was guided largely by his comments. Finally, thanks is also due to professor D. Allison for his supervision of this project.

References

1. E. W. Dijkstra, "The structure of the 'THE' – Multiprogramming System", Comm. ACM 11(5) 1968, 341-346.

2. K. Donovan and S. Madnick, *Operating Systems*, McGraw-Hill Book Co., New York, 1974.

3. P. Brinch Hansen, *Operating System Principles*, Prentice-Hall, Inc., New Jersey, 1973.

4. B. H. Liskov, "The design of the Venus operating system", Comm. ACM 15(3) 1972, 144-149.

5. D. M. Ritchie and K. Thompson, "The UNIX timesharing system", Comm. ACM 17(7) 1974, 365-375.

6. A. C. Shaw, *The Logical Design of Operating Systems*, Prentice-Hall, Inc., New Jersey, 1974.

7. D. C. Tsichritzis and P. A. Bernstein, *Operating Systems*, Academic Press, New York, 1974.

8. R. Watson, *Timesharing System Design Concepts*, McGraw-Hill Book Co., New York, 1970.

9. *Operating Systems Survey*, the Comtre Corp., editor: A. P. Sayers, Auerbach Publ. Inc., 1971.

NAME-THE-USERS'-GROUP CONTEST ANNOUNCED BY HEATH

news release *Received: 77 Jun*

A five-year membership is the prize in a name-the-users'-group contest sponsored by computer hobbyists exchanging information on the use of Heathkits. The more obvious choices will be ruled out by Heath Co.'s objections to use of their trademarks by an independent organization. Entries and requests for further information should be sent to Charles A. Floto, 267 Willow Street, Apt. 27, New Haven, CT. 06511.

PACE DOS SYSTEM FOR $4500

news release *Received: 77 Jul 21*

A new disk-operating system (DOS) for the PACE Micro-processor Development System substantially reduces the time to assemble, edit and execute microprocessor and microcomputer operating and application programs. Designated the IPC-16P/840 by the Microcomputer Systems group of National Semiconductor Corporation, the system includes a dual-floppy disk drive in a stand-alone enclosure, an interface circuit subsystem card, a read-only memory (ROM) card containing firmware and complete operating software on a diskette.

The PACE DOS includes a comprehensive file management capability, support for assembly programs, Editors, Linking Loaders, Utility Programs and Diagnostics. The system may be installed on any IPC-16P PACE Microprocessor Development System with 12K words of Random-access memory (RAM) and heavy-duty power supply.

Each disk drive accepts a diskette capable of storing 2.528 megabits (158K words) of unformatted data. The diskettes have a maximum of 268 hexidecimal or 616 decimal sectors each. To ensure media interchangability between drives, a single-element read/write head straddles an erase element which provides a blank area between data tracks.

The PACE DOS Interface card, and the ROM Firmware card are inserted in any available slots in the PACE Microprocessor Development System. The interface card provides protocol for the dual disk drive and an optional user-supplied cathlode-ray-tube- (CRT) terminal such as the LSI ADM-3 or the Beehive Bee. Firmware, located on the card, operates both disk mechanisms, formats data, and controls data transfers.

Other firmware handles data formatting and transfer between the PACE central processing unit and the CRT terminal. CRT Input/output is RS-232 compatible at transmission rates of 300 baud or 1200 baud. A 20 milliamp current-loop I/O, with a 110 baud rate, is also available for teletypewriters.

The PACE DOS software and firmware combines an effective Monitor, a comprehensive File Manager, and a convenient File I/O Subsystem. With the combination, users reduce development time, speed debug procedures and simplify program testing.

The ROM-resident Monitor provides the user with simplified management control over system configuration and program execution. Single-word commands supply linkage to paper-tape loader, card-reader loader, diskettes, and the DEBUG subsystem.

The PACE DOS File Manager, residing on diskette, facilitates file maintenance, handles space allocation and provides file protection. The user has symbolic naming of program and data files as well as simplified control over directories, disk file specification, and data-file manipulation to and from disk. The system provides four levels of file protection to prevent accidental program destruction.

THE ROM resident PACE DOS File I/O Subsystem contains a collection of software routines that are useful for building microcomputer or microprocessor-based programs.

Upon naming source and destination files, the PACE Assembler automatically performs the assembly operation. The DOS EDITOR speeds generation of new source-statement text and aids modification of existing text in preparation for assembly. It can also be used to edit non-assembler-formatted source such as lists, tables, and directories.

The DOS Linkage Editor (LINKEDIT) relocates and links one or more load modules produced by the assemblers into a main program. LINKEDIT is command driven accepting commands from the console and from paper tape, punched cards, or the diskette itself.

Also included in the DOS Software are the PROM paper-tape/punch utility (PACPRO) and a disk-patching utility (DPATCH). PACPRO permits transferring load modules from card or diskette to paper tape for PROM or ROM programming or from load module paper tape to PROM paper tape. DPATCH allows the user to make permanent patches to a main program file in a manner similar to the way temporary patches are made in memory with a Debug program.

The PACE DOS, IPC 16P/840 is priced at $4500 which includes the dual-floppy disk with complete electronics and power supply, the PROM card for firmware routine, the DISK/CRT interface card and cables. Delivery is 30 days ARO.

National Semiconductor Corporation, Microcomputer Systems Group; 2900 Semiconductor Drive, Santa Clara, CA., 95051.

```
        TITLE  'THE KERNEL SUBROUTINES'



;NOTE:  Small letters will denote the contents of locations referred to
;       by capital letters.  For example, cpp = (CPP), a = (A), etc.


;NMPCB = Total number of PCBs allocated to the operating system
;NMMB  = Total number of message buffers allocated to the system
;MBPP  = Number of message buffers allotted to each process
;NMPR  = Number of priorities allowed

;       0 < NMPCB , NMMB            0 < NMPR          NMPCB < 32
;       0 < cpp =< NMPCB


NMPCB  EQU    5
MBPP   EQU    5
NMMB   EQU    NMPCB*MBPP
NMPR   EQU    5

MASKQ  EQU    1
MASKM  EQU    2
MASKA  EQU    4

ISIS   EQU    64          ; ISIS entry point
LOAD   EQU    6
ERROR  EQU    12




;                            MACROS
;                            ------


;Register indexed address:
RXAD   MACRO  REGPR,BSADD       ; (H&L) <- regpr + BSADD
       LXI    H,BSADD
       DAD    REGPR
       ENDM


;Register indexed load:
RXLD   MACRO  REG,REGPR,BSADD   ; reg <- (regpr + BSADD)
       RXAD   REGPR,BSADD
       MOV    REG,M
       ENDM


;Register indexed store
RXST   MACRO  REG,REGPR,BSADD   ; (regpr + BSADD) <- reg
       RXAD   REGPR,BSADD
       MOV    M,REG
       ENDM


;Register direct load
RDLD   MACRO  REG,ADDR      ; reg <- (ADDR)
       LXI    H,ADDR
       MOV    REG,M
       ENDM


JNZR   MACRO  REG,LABEL     ; If  reg  is not 0, then jump to LABEL
       XRA    A
       CMP    REG
       JNZ    LABEL
       ENDM


JZR    MACRO  REG,LABEL     ; If  reg=0 , then jump to LABEL
       XRA    A
       CMP    REG
       JZ     LABEL
       ENDM


M6A    MACRO               ; d <- 0 , e <- 6*a
       RLC
       MOV    E,A          ; e <- 2*a
       RLC
       ADD    E            ; a <- 6*a
       MOV    E,A
       MVI    D,0
       ENDM


STDM   MACRO               ; (D&E) is stored at address in (H&L)
       MOV    M,E
       INX    H
       MOV    M,D
       ENDM


STORE  MACRO
       PUSH   PSW
       PUSH   H
       PUSH   D
       PUSH   B
       LDA    CPP             ; Now to load STKP with sp
       RLC
       MOV    E,A
       MVI    D,0             ; d <- 0
       RXAD   D,STKP
       XCHG
       LXI    H,0
       DAD    SP
       XCHG                   ; (D&E) <- sp
                              ; (H&L) <- address of STKP
       MOV    M,E
       INX    H
       MOV    M,D
       ENDM


RSTOR  MACRO
       RLC                    ; First to load sp with value from STKP
       MOV    C,A
       RXAD   B,STKP
       MOV    E,M
       INX    H
       MOV    D,M
       XCHG
       SPHL
       POP    B
       POP    D
       POP    H
       POP    PSW
       ENDM


TRNAM  MACRO
;Transfer 6 bytes starting in location indicated by H&L to location
;indicated by D&E.  Note that  b = 0  at end.
       MVI    B,6
XX:    MOV    A,M
       STAX   D
       DCR    B
       JZ     YY+3
       INX    D
```

```
        INX     H
YY:     JMP     XX
        ENDM


UNLNK   MACRO   LAST,NEXT
;Unlink a PCB with index t (which is held in B&C) from the doubly-
;linked list which uses pointers LAST and NEXT.  We assume that
;       e = NEXT[t]  and  a = t
        RXLD    C,B,LAST        ; c <- (LAST[t]) = x
        RXST    E,B,NEXT        ; (NEXT[x]) <- e
        RXST    C,D,LAST        ; (LAST[y]) <- x
        MOV     C,A             ; Restore t to C
        ENDM
        ORG     8000H           ; Origin is set to 32K


;                       THE DATA STRUCTURE
;                       ------------------


CPP:    DB      1
AP:     DB      0
FP:     DB      1
FMBP:   DB      1

AS:     DB      1
        DB      0

PRP:    ORG     $ + NMPR
NMR:    ORG     $ + NMPR

        ; The storage allocated to the message buffers
MBSP    EQU     $ - 1
        ORG     $ + NMMB
NMBP    EQU     $ - 1
        ORG     $ + NMMB
ASTAT   EQU     $ - 1
        ORG     $ + NMMB
MESS    EQU     $ - 2
        ORG     $ + 2*NMMB

        ; The storage allocated to the PCBs                   BLOCKED bit
NAM     EQU     $ - 6                   ;           PROGRAM bit   |
        ORG     $ + 6*NMPCB             ;                         :   :
EXC     EQU     $ - 1                   ;       (EXC[t]) =  XXOOOOXX
        ORG     $ + NMPCB               ;                           ^   ^
PR      EQU     $ - 1                   ;                           | STOPPED bit
        ORG     $ + NMPCB               ;           RESIDENT bit
MBC     EQU     $ - 1
        ORG     $ + NMPCB
WDMK    EQU     $ - 1
        ORG     $ + NMPCB
RDMK    EQU     $ - 1
        ORG     $ + NMPCB
FMP     EQU     $ - 1
        ORG     $ + NMPCB
FQP     EQU     $ - 1
        ORG     $ + NMPCB
FAP     EQU     $ - 1
        ORG     $ + NMPCB
ABP     EQU     $ - 1
        ORG     $ + NMPCB
NSWP    EQU     $ - 1
        ORG     $ + NMPCB
SEMP    EQU     $ - 1
        ORG     $ + NMPCB
LAP     EQU     $ - 1
        ORG     $ + NMPCB
NAP     EQU     $ - 1
        ORG     $ + NMPCB
LP      EQU     $ - 1
        ORG     $ + NMPCB
NP      EQU     $ - 1
        ORG     $ + NMPCB
DSM     EQU     $ - 2
        ORG     $ + 2*NMPCB
STAD    EQU     $ - 2
        ORG     $ + 2*NMPCB
STKP    EQU     $ - 2
        ORG     $ + 2*NMPCB
STACK   EQU     $ - 10          ; Used to store the register pairs B,
        ORG     $ + 10*NMPCB    ; D, H, and also the PSW and PLC


;                       THE KERNEL SUBROUTINES
;                       ----------------------


;                           SCHED (INDEX)

;It is assumed that interrupts were disabled before the scheduler
;was called.  The integer INDEX is passed to the routine in the B
;register and has the following interpretation:

;    INDEX          action to be taken and initial data
;  ------------------------------------------------------------
;      0        If there is another ready process of the same pri-
;               ority as the current one, then the next such after
;               the current one is scheduled.  (The current process
;               will be rescheduled if it is the only ready process
;               in its queue.)  Otherwise, we try to schedule the
;               first ready process of lower priority.  If there is
;               none, then we HALT.  We assume that
;                           q = (PR[CPP]).
;    > 0        The process corresponding to the PCB with index
;               INDEX is scheduled.  We assume that
;                           q = r = (PR[INDEX]).

;At the end we have  b = 0.


SCHED   STORE
        MOV     A,B
        MVI     B,0
        ORA     A
        JZ      ROBIN           ; Jump is INDEX = 0

SCH00:                          ; Entry point for initialization
                                ; routine
        STA     CPP             ; CPP <- INDEX
        MOV     E,A             ; e <- INDEX
        MOV     D,B             ; d <- 0
        RXLD    E,D,NAP         ; e <- x = (NAP[INDEX])
        RXST    E,B,PRP         ; (PRP[r]) <- x
        ; PRP[r] now points to the PCB following the one with
        ; index INDEX

SCH0:   RSTOR
        EI
        RET                     ; Return

ROBIN   RXLD    A,B,NMR         ; a <- (NMR[q])
        ORA     A
        JZ      SCH2            ; Jump if there is no ready process
                                ; of priority q
```

```
        ; The case where  (NMR[q]) > 0
        RXLD    E,B,PRP         ; e ← t = (PRP[q])
        MOV     D,B             ; d <- 0
SCH1:   RXLD    A,D,EXC         ; a ← (EXC[t])
        ANI     00000010B
        JNZ     NEXT            ; Jump if this process is BLOCKED
        RXLD    A,D,NAP         ; a <- (NAP[e])
        RXST    A,B,PRP         ; The priority pointer is set
                                ; to the process after the one
                                ; that is now being scheduled
        LXI     H,CPP
        MOV     M,E             ; cpp <- t
        MOV     A,E
        JMP     SCH0

NEXT:   RXLD    E,D,NAP         ; e ← (NAP[t])
        JMP     SCH1

SCH2:   INR     C               ; Case where (NMR[q]) = 0
        MVI     A,NMPR          ; a ← NMPR
        SUB     C
        JP      ROBIN           ; Jump if  q =< NMPR

        EI
HALT:   HLT                     ; Case of no ready processes



;                               P [ID,SEM]


;SEM is the address of the semaphore and is passed to the routine in
;the D and E registers.  ID is a 1 byte integer specifying which global
;semaphore SEM is referring to, if any.  It is passed to the routine
;in the C register.

;         ID                    semaphore
;      ------------------------------------------------------
;          2                          AS
;          0           DSM or any other semaphore local
;                                  to a process

;Neither  b  nor  c  may be zero at the end if the semaphore had a
;non-zero value.

P:      DI
        XCHG
        JZR     M,PZZ           ; Jump if value of semaphore is 0

        DCR     M               ; Decrease value of semaphore by 1
        EI
        RET                     ; Return

PZZ:    MOV     A,C             ; a ← ID
        XCHG
        RDLD    C,CPP           ; c <- cpp
        MVI     B,0             ; b ← 0
        RXST    A,B,SEMP        ; (SEMP[cpp]) <- ID
        XCHG
        INX     H               ; (H&L) ← address of FWP[SEM]
        MOV     E,M             ; e ← t = index of first process R
                                ; waiting on this semaphore
        MOV     D,B             ; d ← 0
        XRA     A
PXX:    CMP     E
        JNZ     PNXT            ; Jump if other processes waiting
                                ; on semaphore

        MOV     M,C             ; fwp or (NSWP[t]) ← cpp
        RXST    B,B,NSWP        ; (NSWP[cpp]) <- 0
        RXLD    A,B,EXC         ; a <- (EXC[cpp])
        ORI     00000010B
        MOV     M,A             ; BLOCKED bit of current process
                                ; has been set to 1
        RXLD    C,B,PR          ; c <- q = (PR[cpp])
        RXAD    B,NMR           ; (H&L) <- address of NMR[q]
        DCR     M               ; (NMR[q]) ← (NMR[q])-1
        JMP     SCHED           ; Note that  b = 0  and  c = q

PNXT:   RXLD    E,D,NSWP        ; e ← (NSWP[t])
        JMP     PXX


;                               V [SEM]

;SEM is  the address of the semaphore and is passed to the routine
;in the B and C registers.
;NOTE.  The D register is not used.


V:      DI
        MOV     H,B
        MOV     L,C
        SHLD    VTMP1           ; Store the address of the semaphore
        INX     H               ; (H&L) ← address of FWP[SEM]
VXX:    JNZR    M,VYY           ; Jump if at least one (other) process
                                ; is waiting on this semaphore

        LHLD    VTMP1           ; (H&L) ← address of VAL[SEM]
        INR     M               ; Increase value of the semaphore by 1
VOUT:   EI
        RET

VYY:    SHLD    VTMP2           ; Store address of the FWP or NSWP[t]
        MOV     C,M             ; c ← t = index of the next PCB whose
                                ; corresponding process R is waiting
                                ; on this semaphore
        MVI     B,0             ; b ← 0  (b may be non-zero from START)
        RXLD    A,B,EXC         ; a ← (EXC[t])
        RRC
        JNC     VZZ             ; Jump if Rs STOPPED bit is 0

        RXAD    B,NSWP          ; (H&L) ← address of NSWP[t]
        JMP     VXX

VZZ:    RLC
        ANI     01000000B
        MOV     M,A             ; EXC[t]  now has BLOCKED (and
                                ; STOPPED) bit set to 0
        RXST    B,B,SEMP        ; (SEMP[t]) <- 0
        RXLD    A,B,NSWP        ; a ← (NSWP[t])
        LHLD    VTMP2           ; Load address of FWP or previous NSWP
        MOV     M,A             ; R has now been removed from the queue
                                ; of processes waiting on semaphore
        RXLD    A,B,PR          ; a ← r = (PR[t])
        MOV     E,C             ; e ← t , i.e., store t
        MOV     C,A             ; c <- r
        RXAD    B,NMR           ; (H&L) <- address of NMR[r]
        INR     M               ; NMR[r] <- NMR[r] + 1
        RDLD    C,CPP           ; c <- cpp
        RXAD    B,PR            ; (H&L) ← address of PR[cpp]
        CMP     M
        JP      VOUT            ; Jump if  r  q = (PR[cpp])

        ; Case where  r < q
        MOV     C,A             ; c ← r
        MOV     B,E             ; b ← t
        JMP     SCHED

VTMP1:  DW      0
VTMP2:  DW      0
```

```
;                       RESULT := SCAN (NAME)

;NAME is passed to this routine in the H and L registers and is the
;address of the new name.  The routine returns an integer RESULT in
;the A register with the following interpretation:

;                  RESULT              Meaning
;                 ------------------------------------------
;                     1          name belongs to a process
;                     2          name belongs to a program
;                     3               name not found

;At the end, the C register will contain the index t of the last PCB
;scanned.  If RESULT is 1 or 2, then  b = 0 .

SCAN:  SHLD   LBLK              ; Store NAME in LBLK (from parameter
                                ; block for ISIS load routine)
       LDA    AP                ; A now holds the index t of the first
                                ; PCB to look at.  We assume that t  0.
SC1:   MOV    C,A               ; c ← t
       M6A                     ; d ← 0 , e ← 6*t
       RXAD   D,NAM
       XCHG                    ; (D&E) ← address of NAM[6*t]
       LHLD   LBLK             ; (H&L) ← NAME
       MVI    B,6
NXTCH: LDAX   D                 ; Next character of name in PCB is
                                ; loaded into A
       CMP    M
       JZ     MTCH             ; Jump if characters match

       MVI    B,0              ; Reset B to 0
       RXLD   A,B,NP           ; a ← (NP[t])
       ORA    A
       JNZ    SC1              ; Jump if there are more processes
                                ; to scan
       MVI    A,3              ; RESULT ← 3
       RET                     ; Case where name was not found

MTCH:  DCR    B                ; b ← b-1
       JZ     SC2              ; Jump if all 6 characters match

       INX    D
       INX    H
       JMP    NXTCH

       ; Case where a PCB with the right name was found.  Register C
       ; contains the index of that PCB.  Also,  b = 0.
SC2:   RXLD   A,B,EXC          ; a ← (EXC[t])
       RLC
       JC     SC3              ; Jump if this PCB corresponds to a
                               ; program
       MVI    A,1              ; RESULT ← 1
       RET

SC3:   MVI    A,2              ; RESULT ← 2
       RET



;                       RESULT := CREA

;This subroutine performs the basic function of setting up a PCB for a
;program which either is already in memory or which has been loaded
;into memory from the disk.  It is assumed that LBLK holds the address
;of the name of the process which is to be created.  The routine re-
;turns an integer RESULT in the D register which specifies any errors
;that may have occurred:

;          RESULT                Status
;         ------------------------------------------------------
;            0                      no error
;            1          error while attempting to load file from
;                        disk - probably no file with that name
;            3                 no free PCBs available


CREA:  DCR    A
       JZ     CR0              ; Jump if name corresponds to a program

CR00:  MVI    C,LOAD           ; Entry point for initialization
                                ; routine
       LXI    D,LBLK
       CALL   ISIS             ; Load program from disk using ISIS
       LDA    STAT             ; Test error status
       ORA    A
       JZ     CR1              ; Jump if no error in loading

       ; Case of error while attempting to load file from disk.  Note
       ; that ISIS specifies the error.
       MVI    C,ERROR
       LXI    D,EBLK
       CALL   ISIS             ; Type out error message on console
       MVI    D,1              ; RESULT ← 1
       RET

       ; Case where name corresponded to a program.  We assume that
       ; b = 0  and  c = t , where t is the index of the PCB for this
       ; program.
CR0:   RXLD   A,B,EXC          ; a ← (EXC[t])
       ANI    01000000B
       ORI    00000001B
       MOV    M,A              ; (EXC[t]) ← 0X000001
       ; NOTE:  Setting EXC here means that in START we have to reset
       ; the STOPPED bit to 0; however, we save fetching EXC in CREAT.
       MOV    A,C              ; a ← t
       M6A                     ; d ← 0 , e ← 6*t
       JMP    INIT

       ; Program is now loaded
CR1:   LXI    D,FP             ; (D&E) ← address of FP
       LDAX   D                ; a ← fp
       ORA    A
       JNZ    CR2              ; Jump if a free PCB is available

       ; No more free PCBs are available
       MVI    D,3              ; RESULT ← 3
       RET
CR2:   MOV    C,A              ; C now holds the index t of a free PCB
       MVI    B,0              ; b ← 0
       RXLD   A,B,NP           ; a ← (NP[t])
       STAX   D                ; The FP is updated

       ; Now to store the name of the process in the PCB name field.
       MOV    A,C              ; a ← t
       M6A                     ; d ← 0 , e ← 6*t
       RXAD   D,NAM
       XCHG                    ; (D&E) ← address of NAM[6*t]
       LHLD   LBLK             ; (H&L) ← NAME
       TRNAM                   ; Transfer name

CR4:   RXAD   B,EXC            ; (H&L) ← address of EXC[t]
       MVI    M,00000001B      ; (EXC[t]) ← 00000001, i.e., STOPPED
                               ; bit is set
       LHLD   ENTRY
       XCHG
       RXAD   B,STAD
       DAD    B                ; (H&L) <- address of STAD[2*t]
       STDM

       ; Note that  b = 0 , that the execution-state field has been set
```

```
        ; to  0X000001 ; and that STAD[2t,2t+1] contains the initial
        ; starting address of this process.
        ; Next we initialize the remaining fields of the PCB.
INIT    RXAD    B,PR
        MVI     M,4             ; (PR[t]) ← 4 ; i.e., the priority is
                                ; set to a default value of 4
        LXI     D,NMPCB         ; Note that we still have  d = 0
        DAD     D
        MVI     M,MBPP          ; (MBC[t]) ← MBPP

        ; The WDMK, RDMK, FMP, FQP, FAP, ABP, NSWP, SEMP, LAP, NAP,
        ; and LP for this process are now set to  0 .
        MVI     A,11
CR5:    DAD     D
        MVI     M,0
        DCR     A
        JNZ     CR5

        ; Now to link the PCB at the head of the all-PCB-queue and to
        ; update the AP.
        DAD     D               ; (H&L) ← address of NP[t]
        LDA     AP
        MOV     M,A             ; (NP[t]) ← ap
        ORA     A
        JZ      CR6             ; Jump if  ap = 0. This case occurs
                                ; only during initialization
        MOV     E,A             ; e ← ap
        RXST    C,D,LP          ; (LP[ap]) <- t
CR6:    MOV     A,C             ; a ← t
        STA     AP              ; ap ← t

        ; Next to set the DSM and STKP
        RLC                     ; a ← 2*t
        MOV     E,A             ; e ← 2*t
        RLC                     ; a ← 4*t
        ADD     C               ; a ← 5*t
        MOV     L,A             ; l ← 5*t
        MOV     H,B             ; h ← 0
        DAD     H
        XCHG                    ; (D&E) ← 10*t
        MOV     C,L             ; c ← 2*t
        RXST    B,B,DSM         ; The DSM field is set to 0
        INX     H               ;    "         "        "
        MOV     M,B             ;    "         "        "
        RXAD    D,STACK         ; (H&L) ← address of STACK[10*t]
        XCHG
        RXAD    B,STKP          ; (H&L) ← address of STKP[2*t]
        STDM                    ; (STKP[2*t]) ← address of STACK[10*t]
        XCHG                    ; (H&L) ← address of STACK[10*t]

        ; Now to set the top 8 bytes of the stack to 0 so that
        ; when a process is scheduled for the first time its
        ; registers will all be initialized to 0
        MVI     A,8
CR7:    MVI     M,0
        INX     H
        DCR     A
        JNZ     CR7

        XCHG                    ; (D&E) ← address of STACK[10*t+8]
        RXLD    A,B,STAD
        STAX    D
        INX     D
        INX     H
        MOV     A,M
        STAX    D               ; The starting address for the new pro-
                                ; gram is now stored at the bottom of
                                ; the stack.  This address is loaded
                                ; into the PLC when the process is
                                ; started.
        MOV     D,B             ; RESULT ← 0
        RET                     ; Note that  c = 2*t  and  b = 0 .
```

```
;                       RESULT := CREAT (NAME)

;NAME is passed to this routine in the B and C registers and is the
;address of the name of the process to be created.  The routine re-
;turns an integer RESULT in the A register with the following in-
;terpretation:

;       RESULT                  event
;       ------------------------------------------------------
;         0         routine completed its task successfully
;         1         error while attempting to load file from
;                    disk - probably no file with that name
;         2          name already used by another process
;                                or program
;         3               no more PCBs available


CREAT:  PUSH    B               ; Store NAME on the process stack
                                ; Note that an interrupt could occur
                                ; here, so that one cannot simply
                                ; store NAME in memory
        LXI     D,AS
        MVI     C,2
        CALL    P               ; P[AS]
        POP     H               ; (H&L) ← NAME
        CALL    SCAN
        SUI     3
        JM      CR10            ; Jump if name not found among current
                                ; process or program names

        MVI     D,2             ; RESULT ← 2

        ; NOTE: EXIT is a global address which is used by almost all
        ; the routines.
EXIT:   LXI     B,AS
        CALL    V               ; V[AS]
        MOV     A,D             ; Set RESULT
        RET


CR10:   CALL    CREA
        JMP     EXIT


        ; The parameter block for the ISIS load routine.  It is used by
        ; the routines SCAN, CREA, CREAT, START
LBLK:   DW      0               ; Pointer to filename
BIAS:   DW      0               ; Bias address
RETSW:  DW      0               ; Return switch
        DW      ENTRY           ; Starting address for the program
                                ; which was loaded
        DW      STAT            ; Status (returned)
ENTRY:  DS      2

EBLK:                           ; The parameter block for the ISIS
                                ; error routine (used by CREA).
STAT:   DS      2
        DW      STAT


;               RESULT := START (NAME,PRIORITY)

;NAME is passed to this routine in the B and C registers and is the
;address of the name of a process which is to be started.  The E
;register holds the integer PRIORITY which specifies the priority
;that the new process is to have.  A value of 0FFH for PRIORITY is
;interpreted to mean that the new process is to be started with
```

```
;whatever priority it already had.  RESULT is an integer which is
;returned by the routine in the A register.  It has the following
;interpretation

;      RESULT                        event
;   ------------------------------------------------------------
;        0          routine completed its task successfully
;        1          error while attempting to load file from
;                    disk - probably no file with that name
;        2            specified process was not stopped


START:  PUSH   D               ; Store PRIORITY on the stack
        PUSH   B               ; Store NAME on the stack
        LXI    D,AS
        MVI    C,2
        CALL   P               ; P[AS]
        POP    H               ; (H&L) <- NAME
        CALL   SCAN
        DCR    A
        JNZ    NTFND           ; Jump if name not found among current
                               ; process names

        ; Note that C now contains the index t of the PCB with the name
        ; we were looking for and that  b = 0 .

        ; Now to check the STOPPED bit
        RXLD   A,B,EXC         ; a <- (EXC[t])
        RRC
        JC     ST1             ; Jump if process was stopped

        MVI    D,2             ; RESULT <- 2
ST0:    POP    H               ; So as not to leave PRIORITY on
                               ; stack
        JMP    EXIT

        ; Next to store the status of the BLOCKED bit
ST1     RLC                    ; a <- (EXC[t])
        ANI    01000010B
        MOV    M,A             ; STOPPED bit in EXC[t] has now
                               ; been set to 0
        ANI    00000010B
ST2:    STA    ST4+1           ; Store the BLOCKED bit for check-
                               ; ing later

        ; Now to store PRIORITY
        POP    D               ; e <- PRIORITY
        MOV    D,B             ; d <- 0
        MVI    A,0FFH          ; a <- 0FFH
        CMP    E
        RXAD   B,PR            ; (H&L) <- address of PR[t]
        JNZ    ST7             ; Jump if priority is to be changed
        MOV    E,M             ; e <- old priority

        ; Next to compare the priority q of the new process
        ; with  r = (PR[CPP])
ST3:    MOV    A,E             ; a <- q
        RDLD   E,CPP           ; e <- CPP
        RXAD   D,PR            ; (H&L) <- address of PR[CPP]
        CMP    M
        PUSH   PSW             ; Save comparison of q and r

        ; Next to check the BLOCKED bit
        MOV    E,A             ; e <- q
ST4:    MVI    A,0             ; NOTE:  The BLOCKED bit has been
                               ; stored here by an earlier
                               ; instruction
        ORA    A               ; Test the BLOCKED bit
        JNZ    ST5             ; Jump if the new process is blocked

        RXAD   D,NMR
        INR    M               ; (NMR[q]) <- (NMR[q])+1

        ; Now to link the new PCB into the appropriate active queue
        ; at the correct place.
ST5:    RXAD   D,PRP           ; (H&L) <- address of PRP[q]
        JZR    M,ST9           ; Jump if there are no processes
                               ; of priority q

        POP    PSW             ; Restore comparison of q and r.
                               ; Note that  a <- q .
        PUSH   PSW             ; Resave comparison for later
        JNZ    ST8             ; Jump if r does not equal q .
                               ; Note that  e = q .

        ;Case where  q = r
        RDLD   E,CPP           ; e <- CPP
        CALL   LINK            ; Link the new PCB in front of the
                               ; PCB for the current process

        ; Now to check if  CPP = (PRP[q])
        MOV    E,A             ; e <- q
        LDA    CPP
        RXAD   D,PRP           ; (H&L) <- address of PRP[q]
        CMP    M
        JNZ    ST6             ; Jump if  CPP  and  (PRP[q])  are
                               ; unequal

        MOV    M,C             ; (PRP[q]) <- t
ST6     POP    PSW             ; Restore the sign bit
        JP     EXIT            ; Jump if  q >= r

        ; Case where the new process has higher priority than the
        ; current one, so that it should be scheduled to run
        MOV    B,C             ; b <- t
        MOV    C,E             ; c <- q
        PUSH   B               ; Store q and t
        LXI    B,AS
        CALL   V               ; V[AS]
        POP    B
        DI
        JMP    SCHED

ST7:    MOV    M,E             ; (PR[t]) <- PRIORITY
        JMP    ST3

        ; Case where r does not equal q
ST8:    MOV    E,M             ; e <- (PRP[q])
        CALL   LINK            ; Link the new PCB in front of the
                               ; PCB pointed to by the PRP[q]
                               ; pointer
        JMP    ST6

ST9:    MOV    M,C             ; (PRP[q]) <- t
        RXST   C,B,NAP         ; (NAP[t]) <- t
        RXST   C,B,LAP         ; (LAP[t]) <- t
        JMP    ST6


NTFND:  CALL   CREA
        JNZR   D,ST0           ; Jump if error in program loading

        MOV    A,C             ; a <- 2*t , where t is the index of
                               ; the newly created PCB
        RRC
        MOV    C,A             ; c <- t
        RXLD   A,B,EXC         ; a <- (EXC[t])
        ANI    01000000B
        MOV    M,A             ; (EXC[t]) <- 0X000000
        XRA    A               ; a <- 0
        JMP    ST2
```

```
;                              LINK

;This subroutine is used by START to link the PCB with index c in
;front of the PCB with index e into the doubly-linked ready list.

LINK:  RXST   E,B,NAP
       RXLD   E,D,LAP
       MOV    M,C
       RXST   C,D,NAP
       RXST   E,B,LAP
       RET


;                              STOP0

;The main task of this routine is to unlink a process PCB from the
;ready queue and to adjust the relevant priority pointer.  It is
;called by both the STOP and REMOV routines.  We assume that  b=0 ,
;c=t , and  a=( (EXC[t]) shifted right 1 ) , where t is the index
;of the PCB whose corresponding process is to be stopped.  Also,
;registers H and L are assumed to hold the address of EXC[t].
;We still have  b = 0  and  c = t  at the end.

STOP0: RLC
       ORI    00000001B
       MOV    M,A              ; Set STOPPED bit to 1
       ANI    00000010B
       RXLD   E,B,PR           ; e <- q = Qs priority
       MOV    D,B              ; d <- 0
       JNZ    STP01            ; Jump if Q is blocked

       RXAD   D,NMR            ; (H&L) <- address of NMR[q]
       DCR    M                ; (NMR[q] <- (NMR[q])-1

       ; Now to remove the process from its circular list of active
       ; processes
STP01: RXLD   A,B,NAP          ; a <- (NAP[t]) = y
       RXAD   D,PRP            ; (H&L) <- address of PRP[q]
       CMP    C
       JZ     STP03            ; Jump if Q was the only active
                               ; process of priority q , i.e., y = t

       MOV    E,A              ; e <- y
       MOV    A,C              ; a <- t
       CMP    M
       JNZ    STP02              Jump if  t  is unequal to  (PRP[q])

       MOV    M,E              ; (PRP[q]) <- y
STP02: UNLNK  LAP,NAP          ; Unlink Qs PCB from active list
       RET

STP03: MOV    M,B              ; (PRP[q]) <- 0
                               ; List of active priority-q processes
                               ; is now empty.
       RET


;                RESULT  = STOP (NAME)

;NAME is passed to this routine in the B and C registers and is the
;address of the name of a process which is to be stopped.  If  b = 0 ,
;then the current process is trying to stop itself.  RESULT is an
;integer which is returned by the routine in the A register.  It
;has the following interpretation:

;     RESULT                    event
;  ------------------------------------------------------
;       0          routine completed its task successfully
;       1        name not found among current process names

STOP:  PUSH   B                ; Store NAME on the stack
       LXI    D,AS
       MVI    C,2
       CALL   P                ; P[AS]
       POP    H                ; (H&L) <- NAME
       JZR    H,STP2           ; Jump if current running process is
                               ; trying to stop itself

       CALL   SCAN
       DCR    A
       JZ     STP1             ; Jump if name found among current
                               ; processes

       MVI    D,1              ; RESULT <- 1
       JMP    EXIT             ; See CREAT


       ; We now have  c = 0 and  b = t , where t is the index of the
       ; PCB with the name of the process Q that we were looking for.
STP1   RXLD   A,B,EXC          ; a <- (EXC[t])
       RRC
       JC     EXIT             ; Jump if process was STOPPED

       CALL   STOP0
       JMP    EXIT             ; Note that  d = 0

       ; Case where process is trying to stop itself.
STP2:  RDLD   C,CPP             ; c <- cpp
       MVI    B,0
       RXLD   A,B,EXC          ; a <- (EXC[cpp])
       CALL   STOP0+1
NEWP:                          ; REMOV uses this entry point
       RXLD   D,B,PR           ; d <- q = (PR[cpp])
       LXI    B,AS
       CALL   V                ; V[AS]
       MOV    C,D              ; c <- q
       MVI    B,0              ; b <- 0
       DI
       JMP    SCHED


;                              RMV

;This subroutine is called by REMOV to free the resources of the
;process that is about to be removed.

RMV:   ANI    01000000B
       JNZ    RV11             ; Jump if Q corresponds to a permanent-
                               ; ly resident program

       RXLD   E,B,NP           ; e <- (NP[t])=y
       LDA    FP
       MOV    M,A              ; (NP[t]) <- fp
       MOV    A,C
       STA    FP               ; fp <- t
       ; Qs PCB is now back on stack of free PCBs

       LDA    AP
       CMP    C
       JZ     RV12             ; Jump if  ap = t , i.e., Q is the
                               ; first process in the queue

       MOV    A,C              ; a <- t
```

```
        UNLNK  LP,NP            ; Unlink Qs PCB from the list of
                                ; all PCBs

        ; Now to return all message buffers
RV1:    RXAD   B,FMP            ; (H&L) ← address of FMP[t]
        JZR    M,RV3            ; Jump if message queue is empty

        PUSH   B                ; Store t on stack
RV2:    MOV    E,M              ; e ← (FMP[t])=z=index of message
                                ; buffer
        RXLD   C,D,MBSP         ; c ← (MBSP[z])=u
        RXAD   B,MBC            ; (H&L) ← address of MBC[u]
        INR    M                ; (MBC[u]) ← (MBC[u])+1
        RXAD   D,NMBP           ; (H&L) ← address of NMBP[z]
        CMP    M                ; Note that  a = 0
        JNZ    RV2              ; Jump if more messages in queue

        ; Now to link all the buffers in the message queue back
        ; on the free stack
        LDA    FMBP             ; a ← fmbp
        MOV    M,A              ; (NMBP[z]) ← fmbp
        POP    B
        RXLD   A,B,FMP          ; a ← (FMP[t])
        STA    FMBP             ; fmbp ← z

        ; Now to return all returned-answer buffers
RV3:    RXAD   B,FAP            ; (H&L) ← address of FAP[t]
        JZR    M,RV5            ; Jump if answer queue is empty

        PUSH   B
        MOV    E,M              ; e ← (FAP[t]) = z , i.e., save
                                ; index of first buffer
RV4:    MOV    C,M              ; c ← z
        RXAD   B,NMBP           ; (H&L) ← address of NMBP[z]
        CMP    M                ; Note that  a = 0
        JNZ    RV4              ; Jump if more buffers

        LDA    FMBP
        MOV    M,A              ; (NMBP[z]) <- fmbp
        MOV    A,E              ; a ← z
        STA    FMBP
        POP    B

        ; Now to return all question buffers .
RV5:    LXI    H,FQP
        CALL   RMVQA

        ; Now to return all answer buffers
        LXI    H,ABP
        CALL   RMVQA

        ; All buffers from message system have now been returned
        ; Next, to clear those sender fields in the message buffers
        ; which refer to the process which is being removed
        MOV    A,C              ; a <- t
        LXI    D,NMMB
        INR    D
        LXI    H,MBSP
RV6:    INX    H
        CMP    M
        JNZ    RV7

        MVI    M,0
RV7:    DCR    E
        JNZ    RV6

        DCR    D
        JNZ    RV6

        ; Finally, to remove Qs PCB from any semaphore queue
        RXAD   B,SEMP           ; (H&L) <- address of SEM[t]
        JNZR   M,RV8
        RET

        ; Case where process is waiting on AS semaphore
RV8:    RXLD   C,B,NSWP         ; c <- (NSWP[t])
        LXI    H,AS
        INX    H                ; (H&L) <- address of FWP[AS]
        MOV    E,M              ; e <- x = index of first process
                                ; waiting on AS
RV9:    CMP    E
        JZ     RV10

        RXLD   E,D,NSWP         ; e <- (NSWP[x])
        JMP    RV9

RV10:   MOV    M,C
        RET                     ; Note that  d = 0

RV11:   ORI    10000000B
        MOV    M,A              ; Reset EXC field
        MOV    D,B              ; d <- 0
        JMP    RV1

RV12:   MOV    A,E              ; a <- Y
        ORA    A                ; Test Y
        JZ     HALT             ; Jump if we were asked to
                                ; remove the only process in
                                ; existence

        STA    AP               ; ap <- Y
        JMP    RV1



;                       RMVQA


RMVQA:  DAD    B
        JZR    M,RV15

        PUSH   B
        MOV    C,M
RV13:   RXLD   E,B,NMBP
        MOV    M,B              ; Set memory to 0
        PUSH   D
        RXAD   B,ASTAT          ; (H&L) ← address of ASTAT[z]
        MVI    M,1              ; (ASTAT[z]) ← 1
        RXLD   E,B,MBSP
        MOV    B,C
        MVI    C,MASKA          ; c ← 00000100
        LXI    H,FAP
        CALL   SSUB
        POP    B
        JNZR   C,RV13

RV14:   POP    B
RV15:   RET



;                   RESULT := REMOV (NAME)

;NAME is passed to this routine in the B and C registers and is the
;address of the name of a process which is to be removed.  If  b = 0,
;then the process is trying to remove itself.  RESULT is an integer
;which is returned by the routine in the A register with the fol-
;lowing interpretation:
```

```
;       RESULT                      event
;     ------------------------------------------------------------
;        0          routine completed its task successfully
;        1           name not found among current processes
;                                  and programs


REMOV:  PUSH    B               ; Store NAME on the stack
        LXI     D,AS
        MVI     C,2
        CALL    P               ; P[AS]
        POP     H               ; (H&L) <- NAME
        JZR     H,REM3          ; Jump if this process is trying to
                                ; remove itself

        CALL    SCAN
        SUI     3
        JM      REM1            ; Jump if name found among current
                                ; processes or programs

        MVI     D,1             ; RESULT <- 1
        JMP     EXIT            ; See CREAT

        ; We now have  b = 0 and  c = t , where t is the index of the
        ; PCB with the name of the process Q that we were looking for.
REM1:   RXLD    A,B,EXC         ; a <- (EXC[t])
        PUSH    H               ; Save address of EXC[t]
        RRC
        CNC     STOPO           ; Stop Q if it is not already stopped

        ; We still have  b = 0  and  c = t
REM2:   POP     H               ; (H&L) <- address of EXC[t]
        MOV     A,M             ; a <- (EXC[t])
        CALL    RMV
        JMP     EXIT            ; Note that  d = 0

        ; Case where process is trying to remove itself
REM3:   RDLD    C,CPP           ; c <- cpp
        MVI     B,0
        RXLD    A,B,EXC         ; a <- (EXC[cpp])
        PUSH    H               ; Save address of EXC[cpp]
        CALL    STOPO+1
        POP     H               ; (H&L) <- address of EXC[cpp]
        MOV     A,M               a <- (EXC[cpp])
        PUSH    B
        CALL    RMV
        POP     B
        JMP     NEWP            ; See STOP
```

THE MESSAGE SYSTEM

```
;The next four subroutines (SENDM, SENDQ, SENDA, and WAIT) form the
;basis for the message system between processes.  SENDM allows one
;process to send a message to another.  SENDQ also is used to send
;a message, except that this time an answer is expected back.  This
;answer is returned by the other process via SENDA.  WAIT allows a
;process to wait for either a message, question, answer, or
;interrupt.

;NOTE:  MB will be an abbreviation for "message buffer".

;           RESULT := SENDM (MESSAGE,NAME)

;For definiteness assume that process Q wants to send a message
;to process R.  The 2-byte word MESSAGE is passed in the B&C
;register pair.  NAME is passed in the D&E register pair and is
;the address of a 6-byte block in memory containing Rs name.  The
;routine returns a 1-byte integer RESULT in the A register with
;the following interpretation:

;    RESULT                      event
;  ------------------------------------------------------------
;      0         The routine completed its task successfully
;      1              no more message buffers available
;      2       receivers name not found among current processes


SENDM:  CALL    SMQ
        MOV     E,C             ; e <- t = index of Rs PCB
        LXI     H,FMP
        MVI     C,MASKM
        CALL    SSUB
        JMP     EXIT



;           RESULT := SENDQ (ADDRESS,NAME)

;The only difference between this routine and SENDM is that an
;answer is expected back.  The parameters are also basically the
;same.  The only change is caused by the fact that one can only
;pass two parameters in PL/M.  ADDRESS is the address of a 3-byte
;block in memory with the following two fields:

;     BID:      1 byte
;     QUESTION: 2 bytes which are the question (similar to MESSAGE
;               in SENDM)

;The integer BID is returned by the routine and is the index of
;the MB that was used to send the question.  It has importance to
;this particular process only in connection with the WAIT routine
;when it awaits the answer.
;RESULT is an integer returned in the A register with the same
;interpretation as in SENDM.


SENDQ:  PUSH    B
        INX     B
        MOV     H,B
        MOV     L,C             ; (H&L) <- ADDRESS+1
        MOV     C,M
        INX     H
        MOV     B,M             ; (B&C) <- QUESTION
        CALL    SMQ
        POP     H               ; (H&L) <- ADDRESS
        MOV     M,B             ; Store BID in appropriate location
        MOV     E,C             ; e <- t = index of Rs PCB
        LXI     H,FQP
        MVI     C,MASKQ
        CALL    SSUB
        JMP     EXIT



;           RESULT := SENDA (ANSWER,BID)

;ANSWER is a 2-byte word passed in the B&C register pair and BID
;is a 1-byte integer passed in the E register.  BID is the index
```

```
; of the MB that is to be used to return the answer to the process
; that asked the original question.  The routine returns a 1-byte
; integer RESULT in the A register with the following interpreta-
; tion:

;    RESULT                        event
;  ------------------------------------------------------------
;      0          routine completed its task successfully
;      1        the correct buffer for returning the answer
;                          could not be found
;      2         process to whom answer was to be sent was
;                                removed


SENDA: CALL   SA
       MOV    E,B           ; e <- t = index of PCB which belongs
                            ;          to process to which answer
                            ;          is being sent
       MOV    B,C           ; b <- BID
       LXI    H,FAP
       MVI    C,MASKA
       CALL   SSUB
       JMP    EXIT


;                          SMQ

;This subroutine is used by SENDM and SENDQ.  SENDA also uses
;part of it.


SMQ:   PUSH   B
       PUSH   D
       MVI    C,2
       LXI    D,AS
       CALL   P             ; P(AS)
       POP    H             ; (H&L) <- NAME
       CALL   SCAN
       CPI    1
       JNZ    SXX           ; Jump if Rs name not found among
                            ; processes

       ; b = 0  and  c = t , where  t  is the index of Rs PCB
       RDLD   E,CPP         ; e <- cpp
       MVI    D,0
       RXAD   D,MBC         ; (H&L) <- address of MBC[cpp]
       XRA    A             ; a <- 0
       CMP    M
       JZ     SYY           ; Jump if  (MBC[cpp]) = 0

       DCR    M             ;  (MBC[cpp]) <- (MBC[cpp]) - 1
       RDLD   E,FMBP        ; e <- BID = fmbp
       RXLD   A,D,NMBP
       STA    FMBP          ; fmbp <- (NMBP[fmbp])
       MOV    B,E           ; Store new MB index BID in B
       RXAD   D,MBSP        ; (H&L) <- address of MBSP[BID]
       JMP    SMQ1

       ; Entry point from SA subroutine
SAE:   MOV    E,C           ; e <- BID
       MVI    D,0
       MOV    B,A           ; b <- t
SMQ1:  LDA    CPP
       MOV    M,A           ; (MBSP[BID]) <- cpp
       RXST   D,D,NMBP      ; (NMBP[BID]) <- 0
       RXST   D,D,ASTAT     ; (ASTAT[BID]) <- 0
       RXAD   D,MESS
       DAD    D             ;  H&L) <- address of MESS[2*BID]
       POP    D             ; (D&E) <- MESSAGE or ANSWER
       STDM                 ; Store MESSAGE or ANSWER in MESS
                            ; field of MB
       ; In case of SENDM or SENDQ we now have  c = t  and
       ; b = BID
       RET


                           SA

;This subroutine is used by SENDA.


SA:    PUSH   B
       PUSH   D
       LXI    D,AS
       MVI    C,2
       CALL   P             ; P(AS)
       POP    B             ; c <- BID
       MVI    B,0

       ; Begin searching answer-buffer queue for MB with index BID
       RDLD   E,CPP         ; e <- cpp
       MOV    D,B           ; d <- 0
       RXAD   D,ABP         ; (H&L) <- address of ABP[cpp]
       JMP    SA2

SA1:   LXI    H,NMBP
       MOV    E,A
       DAD    D
SA2:   MOV    A,M
       CMP    D
       JZ     SYY           ; Jump if no buffer found with index
                            ; equal to BID

       CMP    C
       JNZ    SA1           ; Jump if index of this buffer does not
                            ; equal BID

       XCHG
       RXLD   A,B,NMBP
       STAX   D
       ; End of search.  Also, buffer has been unlinked from queue.

       ; Check whether sender of question has been removed in the
       ; meantime
       RXLD   A,B,MBSP      ; (H&L) <- address of MBSP[BID]
                            ; a <- t = (MBSP[BID])
       ORA    A             ; Test if  a = 0
       JNZ    SAE           ; Jump if sender not removed

       ; Case where sender was removed
       LDA    FMBP
       RXST   A,B,NMBP      ; (NMBP[BID]) <- fmbp
       MOV    A,C
       STA    FMBP          ; fmbp <- BID
       JMP    SXX



       ; Exit points
SXX:   MVI    D,1
SYY:   POP    H             ; Pop NAME or BID
       POP    H             ; Pop return address from SENDX
       INR    D             ; Set RESULT
       JMP    EXIT



;                          SSUB

;This subroutine is used by SENDM, SENDQ, SENDA, and RMV.
;For this subroutine we expect MASKX in C, BID in B, index t of
;Rs PCB in E, and address of FXP in H&L.
```

```
SSUB:  MVI    D,0
       DAD    D              ; (H&L) <- address of (FMP or FQP
                             ; or FAP)[t]
       JZR    M,SSUB2

       PUSH   D              ; Save t
SSUB1: MOV    E,M            ; e <- s
       RXAD   D,NMBP         ; (H&L) <- address of NMBP[s]
       CMP    M              ; Note that  a = 0
       JNZ    SSUB1          ; Jump if we still have not reached
                             ; the last MB in the queue

       POP    D              ; e <- t
SSUB2: MOV    M,B

       ; Entry point from send-interrupt routine
SEND1: RXAD   D,RDMK         ; (H&L) <- address of RDMK[t]
       MOV    A,C            ; a <- MASKX
       DI
       ORA    M
       MOV    M,A            ; (RDMK[t]) <- (RDMK[t]) v SMASK
       RXAD   D,WDMK         ; (H&L) <- address of WDMK[t]
       MOV    A,C
       ANA    M
       JZ     SSUB3          ; Jump if (WDMK[receiver] AND SMASK)=0

       MVI    M,0            ; (WDMK[receiver])
       EI
       RXAD   D,DSM
       DAD    D              ; (H&L) <- address of DSM[2*t]
       MOV    C,L
       MOV    B,H
       CALL   V              ; V(DSM[receiver])
       RET                   ; Note that  d = 0

SSUB3: EI
       RET                   ; Note that  d = 0
```

```
;            RESULT := WAIT (WTBLK,MASK)

;This routine allows a process to wait for a message or interrupt.
; WTBLK (passed in the B&C register pair) is the address of a 10
; byte block in memory with the following fields:

;      EVENT:    1 byte.  This binary word specifies what type of
;                data was received and has the same interpretation
;                as MASK.
;      DATA:     2 bytes that are either a message, question, or
;                answer.
;      BUFID:    1 byte.  Since it is possible to send more than one
;                question to another process, in order to determine
;                to which question an answer (obtained via the WAIT
;                routine) corresponds, one merely has to check that
;                the integer BUFID matches the integer BID that was
;                returned by the SENDQ routine.
;      SENDER:   6 bytes which contain the name of the sender.

; MASK is a 1-byte word passed in the E register.  Its bits specify
; what data is being waited for.

;          MASK
;           bit                 Request
;         ---------------------------------
;            0                 Question
;            1                 Message
;            2                 Answer
;           3-7                Interrupts

; Requests for interrupts are handled first.  After that, the word
; is scanned from right to left.
; The routine returns a 1-byte integer RESULT in the A register
; with the following interpretation:

;       RESULT                      event
;      ------------------------------------------------------
;         0          routine completed its task successfully
;         1                  case of dummy answer
;         2               case where sender was removed


WAIT:  PUSH   B              ; Store WTBLK on stack
WT1:   RDLD   C,CPP          ; c <- cpp
       MVI    B,0
       RXAD   B,RDMK         ; (H&L) <- address of RDMK[cpp]
       MOV    A,E            ; a <- MASK
       DI
       ANA    M              ; a <- (MASK AND (RDMK[cpp]))
       CMP    B
       JNZ    WCASE          ; Jump if  a  is not  0

       RXST   E,B,WDMK       ; (WDMK[cpp]) <- MASK
       RXAD   B,DSM
       DAD    B              ; (H&L) <- address of DSM[2*cpp]
       PUSH   D              ; Save MASK
       XCHG
       MOV    C,B            ; c <- 0
       CALL   P+1            ; P(DSM[cpp])
       POP    D
       JMP    WT1

       ; Case where desired data had been received
WCASE: CPI    8
       JP     CASEI          ; Jump if event is interrupt

       PUSH   H              ; Store address of RDMK[cpp] on stack
       PUSH   PSW
       LXI    D,AS
       MVI    C,2
       CALL   P              ; P(AS)
       POP    PSW
       POP    D              ; (D&E) <- address of RDMK[cpp]
       RRC
       JNC    WAM            ; Jump if question is not desired

       ; Case of a question
CASEQ: LXI    H,ABP-RDMK
       DAD    D              ; (H&L) <- address of ABP[cpp]
       PUSH   H
       LXI    H,FQP-RDMK
       MVI    A,MASKQ
       CALL   WMAIN
       JMP    EXIT


WAM:   RRC
       LXI    H,FMBP
       PUSH   H
       JNC    CASEA          ; Jump if message is not desired

       ; Case of a message
CASEM: LXI    H,FMF-RDMK
       MVI    A,MASKM
       CALL   WMAIN
       JC     EXIT           ; Jump if  sdr = 0 , i.e., actual
                             ; sender has been removed in the
                             ; meantime

       RXAD   B,MBC          ; (H&L) <- address of MBC[sdr]
       INR    M              ; (MBC[sdr]) <- (MBC[sdr])+1
       JMP    EXIT
```

```
        ; Case of an answer (and no message or question)
CASEA:  LXI     H,FAP-RDMK
        MVI     A,MASKA
        CALL    WMAIN
        RDLD    C,CPP           ; c <- cpp
        RXAD    B,MBC           ; (H&L) <- address of MBC[cpp]
        INR     M               ; (MBC[cpp]) <- (MBC[cpp])+1
        JMP     EXIT


        ; Case of interrupt
CASEI:  MOV     C,A             ; c <- (MASK AND RDMK[cpp])
        MVI     B,1             ; b <- MASKI
WINT1:  MOV     A,B
        RRC                     ; Rotate MASKI right
        MOV     B,A
        ANA     C               ; a <- (MASKI AND MASK AND (RDMK[cpp]))
        ORA     A               ; Test if  a = 0
        JZ      WINT1           ; Jump if  a = 0
        CMA
        ANA     M               ; (RDMK[cpp]) <- (RDMK[cpp]) AND -MASKI
        MOV     M,A
        EI
        POP     H
        MOV     M,B
        XRA     A               ; RESULT = 0
        RET



;                       WMAIN

;This subroutine does all the necessary work for WAIT.


WMAIN:  DAD     D               ; (H&L) <- address of FXP[cpp] ,
                                ; where  X = M, Q, or A
        PUSH    D               ; Save address of RDMK[cpp]
        MVI     B,0
        MOV     C,M             ; c <- BUFID
        XCHG                    ; (D&E) <- address of FXP[cpp]
        RXAD    B,NMBP          ; (H&L) <- address of NMBP[BUFID]
        MOV     B,A             ; Save MASKX in B
        MOV     A,M             ; a <- (NMBP[BUFID])
        XCHG                    ; (D&E) <- address of NMBP[BUFID]
                                ; (H&L) <- address of FXP[cpp]
        MOV     M,A             ; Set FXP to new value
        POP     H               ; (H&L) <- address of RDMK[cpp]
        ORA     A               ; Test if  (NMBP[BUFID]) = 0
        JNZ     WM1             ; Jump if queue is not empty

        MOV     A,B
        CMA
        ANA     M
        MOV     M,A             ; Appropriate bit in (RDMK[cpp]) has
                                ; been set to 0
WM1:    POP     H               ; (H&L) <- return address from call to
                                ;          WMAIN
        XTHL                    ; Exchange H&L with top of stack
                                ; which holds address of FMBP (in case
                                ; of message or answer) or ABP[cpp]
        MOV     A,M
        STAX    D               ; Store  a  in NMBP[BUFID]
        MOV     M,C             ; fmbp  or  (ABP[cpp]) <- BUFID

        ; Assemble wait block
        POP     H               ; (H&L) <- return address from call
                                ; to WMAIN
        XTHL                    ; Exchange H&L with top of stack which
                                ; holds WTBLK
        MOV     M,B             ; Store record of event, namely MASKX,
                                ; in WTBLK
        INX     H
        PUSH    H
        MVI     B,0
        RXLD    A,B,ASTAT       ; a <- (ASTAT[BUFID])
        ORA     A               ; Test if  a = 0
        JNZ     WM2             ; Jump if MB contains a dummy answer.
                                ; This can happen only if WMAIN was
                                ; called from CASEA.  It may have hap-
                                ; pened that a process was removed and
                                ; it never had a chance to answer the
                                ; question it was asked.  In that case
                                ; (ASTAT[BUFID]) was set to 1.

        RXAD    B,MESS
        DAD     B               ; (H&L) <- address of MESS[2*BUFID]
        MOV     E,M
        INX     H
        MOV     D,M             ; (D&E) <- MESSAGE or ANSWER
        POP     H
        STDM                    ; Store MESSAGE or ANSWER in WTBLK
        INX     H
        MOV     M,C             ; Store BUFID in WTBLK
        INX     H
        XCHG
        RXLD    A,B,MBSP        ; a <- sdr = (MBSP[BUFID])
        ORA     A               ; Test if  a = 0
        JZ      WM3             ; Jump if  sdr = 0 , i.e., actual
                                ; sender has been removed in the
                                ; meantime

;       NOTE: We assume that the carry bit which now is 0 stays
;       that way until we return from this subroutine.
        MOV     C,A             ; Save  sdr  in C
        PUSH    D
        M6A                     ; e <- 6*sdr , d <- 0
        RXAD    D,NAM           ; (H&L) <- address of NAM[6*sdr]
        POP     D               ; (D&E) <- address of SENDER field
                                ; in WTBLK
        TRNAM                   ; Transfer name
        MOV     D,B             ; d <- 0
        RET                     ; Note that the carry bit is assumed
                                ; to be 0


WM2:    MVI     D,1             ; Set RESULT
        RET

WM3:    MVI     D,2             ; Set RESULT (sdr=0)
        STC                     ; Set the carry bit to 1
        RET


        END     HALT
```

SOLUS: SOL USERS SOCIETY

To ALL SOL USERS:

The initial meeting of SOLUS (SOL Users Society) took place on July 31, 1977 at Stanford University.

The agenda included:

I Goals
II Organization
III Newsletter
IV Hardware Review Procedure
V Software Exchange Procedure
VI Other Topics

For further information, contact:

Bill Burns
4190 Maybell Way
Palo Alto, CA. 94306

BASIC-CODED MEMORY DUMPER

John B. Palmer, P.O. Box 27462, Tucson, AZ. 85726

Gentlemen; *Received: 77 Jun 13*

Enclosed is a program that I enjoyed writing and just maybe someone else might like it.

I am a long-time 'ham' just now turned computer hobbyist. I have an Altair 880B with 16K, and audio cassette and a KSR-33 Teletype.

I like to PEEK and POKE into Basic. Without a memory dump it is hard to know what is going on. The PEEK feature of Basic gives values in decimal. I wanted a dump in octal so I wrote the enclosed program.

This program, written in MITS 8K Basic, will give a octal or decimal printed page for any section of memory within the range of Basic's PEEK statement.

The left margin of the print-out gives the address in octal, followed by octal data from eight consecutive bytes of memory. The right margin gives the ASCII characters for the same eight memory locations. Non-ASCII values receive just a ".", rather than a space. Carriage return and line feed are indicated by "CR" and "LF" after the respective octal data.

The letters "P", "I", and "O" indicate a jump, input, or output instruction. However, you have to decide for yourself if it is a machine instruction or something else. Likewise the ASCII characters often are meaningless. Still, it helps.

A decimal dump is formatted so that it can be compared to the octal dump, indicating the equivalent octal page.

The decimal to octal subroutine in lines 600 to 700 may be of interest to others who (like me) have a Basic without the OCT$ feature of Extended Basic.

```
100 PRINT "MEMORY DUMP WITH ASCII"
110 REM JB PALMER MAY77
120 PRINT "WHICH TYPE, ";
130 INPUT "O OR D ";A$
140 IF A$="O" OR A$="D" THEN 170
150 PRINT "O=OCTAL DUMP, D=DECIMAL DUMP, WHICH, ";
160 GOTO 130
170 P$="ESTIMATED DECIMAL ADDRESS TO "
180 PRINT P$;
190 INPUT "START ";ST
200 PRINT P$;
210 INPUT "FINISH ";FI
220 IF A$="D" THEN 790
230 IF ST>PA*256+255 THEN PA=PA+1:GOTO 230
240 MEMRY=PA*256
250 PRINT "START AT OCTAL PAGE ";
260 DD=PA:GOSUB 600
270 PRINT " = DECIMAL ADDRESS"ME;
280 INPUT " -ALRIGHT (Y/N) ";Y$
290 IF LEFT$(Y$,1)="N" THEN 180
300 PRINT:PRINT
310 DD=PA
320 PRINT "PAGE ";:GOSUB 600:PRINT " 0 ="ME"D";
330 PRINT TAB(28)"OCTAL DATA"SPC(16)"ASCII";
340 LINE=0
350 PRINT
360 DD=ME-256*PA
370 GOSUB 600
380 PRINT,
390 FOR K=0 TO 7
400 DD=PEEK(ME)
410 GOSUB 600
420 DD=PEEK(ME)
430 IF DD=10 THEN PRINT "LF";:GOTO 490
440 IF DD=13 THEN PRINT "CR";:GOTO 490
450 IF DD=195 THEN PRINT "P ";:GOTO 490
460 IF DD=211 THEN PRINT "O ";:GOTO 490
470 IF DD=219 THEN PRINT "I ";:GOTO 490
480 PRINT "  ";
490 ME=ME+1
500 NEXT K
510 GOSUB 710
520 LINE=LI+1
530 IF LI<>32 THEN 350
540 IF MEMRY>FINISH THEN 570
550 PAGE=PA+1
560 GOTO 300
570 PRINT
580 PRINT
590 PRINT:PRINT:END
600 REM DECIMAL TO OCTAL
610 03=0:02=0:01=0
620 IF DD>63 THEN DD=DD-64:03=03+1:GOTO 620
630 03$=RIGHT$(STR$(03),1)
640 PRINT 03$;
650 IF DD>7 THEN DD=DD-8:02=02+1:GOTO 650
660 02$=RIGHT$(STR$(02),1)
670 PRINT 02$;
680 01$=RIGHT$(STR$(DD),1)
690 PRINT 01$;
700 RETURN
710 REM PRINT ASCII
720 BYTE=ME-8
730 FOR K=0 TO 7
740 DD=PEEK(BY+K)
750 IF DD>31 AND DD<97 THEN PRINT CHR$(DD);:GOTO 770
760 PRINT ".";
770 NEXT K
780 RETURN
790 PRINT "DECIMAL DUMP"
800 ME=INT(ST/8)*8
810 IF ST>ME+7 THEN ME=ME+8:GOTO 810
820 PRINT "START AT DECIMAL ADDRESS"ME
830 IF ME>PA*256+255 THEN PAGE=PA+1:GOTO 830
840 PRINT:PRINT:DD=PA:PRINT"ADDRESS  [PAGE ";
850 GOSUB 600:PRINT " 0]"TAB(27)"DECIMAL DATA"SPC(15)"ASCII";
860 PRINT
870 PRINT ME,
880 FOR K=0 TO 7
890 DD=PEEK(ME)
900 IF DD>99 THEN DD$=RIGHT$(STR$(DD),3):GOTO 930
910 IF DD>9 THEN DD$="0"+RIGHT$(STR$(DD),2):GOTO 930
920 DD$="00"+RIGHT$(STR$(DD),1)
930 PRINT DD$"  ";
940 ME=ME+1
950 NEXT K
960 GOSUB 710
970 IF ME>FI THEN 999
980 IF ME>PA*256+255 THEN PAGE=PA+1:GOTO 840
990 GOTO 860
999 PRINT:PRINT:END
OK
```

DUE CREDIT

Dear Jim, *Received: 77 Jul 18*

I noticed the version of CHASE which was published in the May issue of *Dr. Dobb's*. According to the article, the game originally appeared in *Kilobaud* No. 2. I'd like to point out that while *Kilobaud* did publish the game, it originally appeared in the January-February 1976 issue of *Creative Computing* which *Kilobaud* did acknowledge. I'm sure this was just an oversight but credit should be given where it is due.

Incidentally, a friend tells me that he saw a realtime version of CHASE at the ComputerMart of NY (that is, the robots keep moving as you enter commands.) Actually, it's not at all difficult to modify the game you published- just take a look at the keyboard and see if a move is entered rather than wait for a key (I think so, anyway).

Regards,
Steve North

ONE OF THOSE DOPEY PUNS

The Editor was recently discussing possible applications of personal computers with a friend. One possible application would be in physician's offices, for use in checking out possible drug interactions prior to prescribing a new pharmaceutical for a patient who was already taking several drugs. (It is becoming increasingly difficult for physicians to keep up with the available pharmaceuticals and their possible interactions. Why not just put all the information in an office computer; tell it what drugs are currently being prescribed for the patient, and the new drug being proposed. . .and, let the computer check its interaction information for possible contra-indications?) Our glorious Editor pointed out that such a system could be updated - - say, each month - - by simply sending the physician a cassette tape with the newest dope on it. (At that point, the conversation degenerated into groans.)

PATCHING MICROSOFT'S 4.0 BASIC ON P.T.'s SOL

Dear Dr. Dobbs: 77 Sept 5

Just a quick note to pass along a couple of things. First, the kudos and brickbats. The good guys; *Dr. Dobbs,* your publication is the most interesting of all the "home computing" publications. Xybek, and their PROM programming board, a good product with excellent documentation, and people with a genuine desire to help their customers, Micro-Chess, a super chess playing program, with good documentation and priced fairly. The bad guys; the multitude of manufacturers who have promised so much, and have delivered so little, (IMSAI, where's the 12K BASIC, Processor Tech, where's the 8K BASIC so long overdue?).

Attached are the patches I made for a friend so he could use his Altair BASIC on his new Sol. I know you would prefer an assembler listing, but I did these patches by hand. The program should first be loaded, the patches made, and a copy of the modified program be dumped before running the first time. The first column is the old data, the second column is the change. I understand that there may be more than one version of the extended 4.0, so take care that your version is the same as this one.

Thanks for the neat publication. I enjoy it from cover to cover.

Jack L. Calaway
165 E Sierra Madre Blvd
Sierra Madre, CA 91024

P.S. This has been typed using Michael Shrayer's "Electric Pencil Word Processor".

```
;PATCHES TO ALTAIR CASSETTE VERSION OF LEVEL 4.0 BASIC
;FOR USE ON A PROCESSOR TECHNOLOGY SOL WITH SOLO
;LOAD THE ORIGINAL PROGRAM, MAKE THE CHANGES AND SAVE
;A COPY BEFORE RUNNING IT.

ADDRESS         ORIGINAL        CHANGE

;8K V-4.0

;DISABLE TERMINAL INITIALIZATION, SET UP CUTTER

193A            DB              3E
193B            FF              00
193C            E6              D3
193D            F0              FA
193E            0F              C9

;SEND NULL CHARACTER ON CSAVE, SO CLOAD WORKS OK

0030            00              F5
0031            00              AF
0032            00              D3
0033            00              FB
0034            00              F1
0035            00              06
0036            00              01
0037            00              C9

;OUTPUT TO THE SOLO, RESET THE FLAGS

0547            DB              F1
0548            00              C5
0549            E6              47
054A            80              CD
054B            C2              19
054C            47              C0
054D            05              78
054E            F1              C1
054F            D3              B7
0550            01              C9
```

```
Address         Original        Change

;PARITY STRIPPER FOR CNTL-C TEST

0551            F5              E6
0552            00              7F
0553            00              C3
0554            F1              70
0555            C9              06

;MAIN CHARACTER INPUT ROUTINE

0556            DB              CD
0557            00              1F
0558            E6              C0
0559            01              CA
055A            C2              56
055B            56              05
055C            05              00
055D            DB              00
055E            01              00

;INPUT TEST

060C            DB              CD
060D            00              1F
060E            E6              C0
060F            01              00
0610            CC              C4      PATCH
0611            6D              51
0612            06              05

;INPUT TEST

0668            DB              CD
0669            00              1F
066A            E6              C0
066B            01              C8
066C            C0              00

;CASSETTE INPUT (NOT CONSECUTIVE)

1141            06              FA
1143            01              50
1144            C2              CA
1148            07              FB

;CASSETTE OUTPUT (NOT CONSECUTIVE)

114F            06              FA
1152            C2              CA
1157            07              FB

;CASSETTE INITIALIZATION

1159            06              F7      RST 6
115A            01              00

; Extended Mods
;DISABLE CONSOLE SWAPPING FUNCTION

3A43            FE              00
3A44            43 (C)          00

;PREVENT TERMINAL SETUP, AND INITIALIZE CUTTER

38EB            DB              3E
38EC            FF              00
38ED            E6              D3
38EE            F0              FA
38EF            0F              C9

;SHORTEN ILLEGAL FUNCTION CALL, AND OUTPUT NULL
 TO CASSETTE

0349            4C              20
034A            45              46
034B            47              55
```

```
Address         Original        Change

034C            41              4E
034D            4C              20
034E            20              43
034F            46              41
0350            55              4C
0351            4E              4C
0352            43              00 (EOM)
;
0353            54              F5 NULL PATCH
0354            49              AF
0355            4F              D3
0356            4E              FB
0357            20              F1
0358            43              06
0359            41              01
035A            4C              FE
035B            4C              F4
035C            00              C9

; OUTPUT

0E01            DB              F1
0E02            00              C5
0E03            E6              47
0E04            80              CD
0E05            C2              19
0E06            01              C0
0E07            0E              78
0E08            F1              C1
0E09            D3              B7
0E0A            01              C9

; PATCH FOR CNTL-C TEST

0E0B            F5              E6 PATCH
0E0C            00              7F
0E0D            00              C3
0E0E            F1              C6
0E0F            C9              0F

; MAIN INPUT ROUTINE

0E10            DB              CD
0E11            00              1F
0E12            E6              C0
0E13            01              CA
0E14            C2              10
0E15            10              0E
0E16            0E              00
0E17            DB              00
0E18            01              00

; INPUT TEST

0EAD            DB              CD
0EAE            00              1F
0EAF            E6              C0
0EB0            01              00
0EB1            CC              C4
0EB2            C3              0B
0EB3            0F              0E

; INPUT TEST

0FBE            DB              CD
0FBF            00              1F
0FC0            E6              C0
0FC1            01              C8
0FC2            C0              00

; CASSETTE INPUT (NOT CONSECUTIVE)

22F6            06              FA
22F8            01              50
22F9            C2              CA
22FD            07              FB
```

NOTES ON T.BASIC & T.PILOT

To: Dennis Allison *June 6, 1977*

As a user of your work via Pittman's 6800 version of Tiny, I read with care when you propose TP.

First reaction – *Horray!*

I'll have to read it closely to see what it won't do that Tiny will. If it will do all, more *Horray*. I manipulate strings much more than I do numbers on all machines that I use.

1) *Please consider that the only non-alpha character under the resting fingers of a terminal is the semicolon. Consider also that the colon has meandered around on different versions of ASCII terminals and typewriters.*

I have the colon 3 different places on 3 terminals in my office.

Therefore; Please consider semicolon instead of colon for the most used punctuation in your language.

(It's not busy for "don't return" since you used @, therefore has no use at all.)

2) *Please put back in an external call* (peek & poke, or USR). A majority of uses of TINY are as higher-level drivers for things being done partly in machine language.

3) I hope it's just editor translations that left out < and > in March ddj.

4) If you can't see using semicolon instead of colon, how about using it instead of , the second most used symbol.

Thanks for listening.

N J Thompson
Univ. of Hawaii at Manoa
Hawaii Inst. of Geophysics

2525 Correa Rd
Honolulu, HI 96822
Cable Address: UNIHAW

Dear Tom:

Regarding your comments: 1. The choice of colon was historical; it's used that way in the PILOT-73 language and other versions. If it's a problem for you, you could put it on a toggle and let the user specify it. That could, of course, get him into trouble. 2. External calls are important, but are something of a bandaid extension to the language. In my first cut, I left them out. Incidentally, there is a revised specification for Tiny BASIC, unpublished and unimplemented, which supports strings and external functions. Also, Bob Albrecht and I are working on a new language. 3. A lot of typo's did creap into the published text. My fault too! I did proof it, but somehow in the rush to get *DDJ* out, the changes did not get made. 4. Take a look at *PCC*'s version of the Tiny PILOT specification. It's a bit more detailed and has a few minor differences. (It was published earlier than the *DDJ* version but was prepared later.) The major difference is that '#' need appear only when a number is wanted inside a string; the usual tiny BASIC convention works fine in assignment statements (variables need not be prefaced by '#').

– Dennis Allison

```
; CASSETTE OUTPUT (NOT CONSECUTIVE)

2304            06              FA
2307            C2              CA
230C            07              FB

; CALL NULL ADDING ROUTINE

230E            06              CD
230F            01              53
2310            FE              03
2311            F4              00
```

WEE IMPROVEMENTS TO MSI's FD8 DISC SYSTEM

Gary D. Gaugler
2276 Beaver Valley Rd
Fairborn, OH 45324

Received: 77-Aug-25

This computer note describes a patch to MSI DOS Version .987 which will produce new output information in response to the "FILES" or "CATALOG" commands. The form of the new output is:

filename trk sctr lngth typ begin.load end.load PC.start

In unmodified form, DOS only lists the filename, track, sector, length, and file type. The patch described will add very useful information in determining a program's position in memory. The address where the file begins in memory and also the address where the file ends are printed, as well as the address where program execution starts (PC contents). Further, if PC.start is not within the file's contiguous memory space, it probably means that the file is a "load" file not a "run" file.

To use this patch, boot DOS. Observe the "DOS READY" message. Load MINIDOS. Press the reset button. Use MIKBUG or the resident monitor to insert the patch. Execute MINIDOS ($7700). Store $10 sectors starting at address $2400 on track 00, sector 00 and on track 01, sector 00. Press the reset button, boot DOS and type "FILES c/r" to observe results. The patch is now installed and saved on the loaded diskette.

Mem Address	Contents of memory locations
2F4F	81 24
2F6E	8D 56
2F70	08
2F71	8D 53
2F73	8D 51
2F75	8D 52
2F77	8D 50
2F79	8D 4E
2F7B	BD 2A96
2F7E	C6 05
2F80	08
2F81	5A
2F82	26 FC
2F84	9C 0A
2F86	27 B1
2F88	7E 2F4B

NOTE: filename is any legal
DOS filename and password.

ABOUT-TO-BE 1802 SOFTWARE LISTINGS BOOKLET

Received: 77 Oct 4

Very little software for the RCA CDP1802 is currently in the public domain. To remedy this situation I am going to publish a 10-page booklet listing available software. If you desire to sell or even give away your software please send me a listing for my review. My booklet will provide a complete description and cost information with a reference number corresponding to a number on an ordering coupon.

I plan to charge $1 for the booklet. This amount will also cover the costs associated with processing the coupons. The use of the coupon will reduce the costs to the person ordering from more than one source.

The publication date is set for early December. Advance orders may be made at $1 per copy. Send all orders, software listings, and other correspondence to:

Ross Wirth
1636 S. 108 E. Ave
Tulsa, OK 74128

E-PROM PROGRAMMER FROM MINIMICROMART

news release *Received: 77 Jul 8*

A simple, low cost E-PROM Programmer kit is available from Mini Micro Mart for the popular 2708 series of devices. The newer 16K (2K x 8) Intel and TI devices can also be accomodated. Originally designed to function with its series of RM 8080 and Z-80 systems, it is equally effective on any system that has available three latched parallel I/O ports such as that provided by an 8255 interface IC. These are found on many systems including the Intel SBC 80/10 modules.

The programmer is software driven from the user's processor. Some software routines are included and are intended to run with a system monitor provided by Mini Micro Mart. The user may need to tailor the software to his own operating system.

A power supply kit is available as a separate item. The needed voltages are regulated +12, +5, and -5, and a source of DC for program pulses in excess of 25V DC.

The price of the kit is $24.95 and includes a zero insertion force socket. Order as 3164-1. With a quick change zero insertion force socket (Textool) order as 3164-2 at $34.95.

Other PROM programmers are also available from Mini Micro Mart. For details write to Mini Micro Mart, 1618 James Street, Syracuse, New York 13203. Phone (315) 422-4467.

Memory Address	New Contents
2B29	50 55
	54 00
	31 7A
317A	86 22
317C	B7 3057
317F	8D 03
3181	7E 2BCD
3184	BD 2C9D
3187	BD 2803
318A	CE 2B69
318D	BD 2EDD
3190	FF 0008
3193	BD 243C
3196	39

HEXADECIMAL CALCULATOR HELPS PROGRAMMERS WORK WITH COMPUTER NUMBER SYSTEMS

Dallas, TX.-- A new handheld calculator from Texas Instruments Incorporated does arithmetic in three different number bases and performs conversions to and from those bases rapidly.

Called the TI Programmer, it is the latest productivity tool for programmers and others who work with computers. The number bases include hexadecimal (base 16) and octal (base 8) as well as decimal for everyday arithmetic.

"The TI Programmer will save many hours of work for people dealing with computers ranging from microcomputers up to the larger data processing installations," a TI spokesman reports. Besides programmers, it is meant for systems analysts, supervisors, students, small system operators, computer hobbyists, logic designers and computer maintenance and service personnel. It sells for $49.95.

Since most computers use a binary number system for internal data storage and addressing, programmers and computer users are often required to convert and manipulate coded numbers in other related number bases. In the past, this required time-consuming manual calculations and the use of addition, multiplication and conversion tables.

Now users of the new TI calculator can get those answers fast by keying in the problems in the same number system used by the computer. Then the calculator performs its operations in that number system and converts results to a number system which users can interpret more rapidly.

For example, a computer may store a 32 bit code containing that many "ones" or "zeros" in binary form to represent either data or internal instructions. In the hexadecimal number base, that same code shrinks to only eight digits which can be operated upon and stored in the calculator memory just as these same things are done with the longer code in the computer system.

Among other typical applications, the TI Programmer will convert memory addresses to decimal form, add relative addresses to a base address to find specific computer memory locations, or determine if there is enough space in the computer's memory to hold a new block of data. Also, the arithmetic and logical functions of the calculator can emulate internal computer operations.

The calculator's parentheses keys help evaluate complex expressions without requiring the user to store or write down intermediate results. The parentheses specify the order of execution of operations in a problem. Up to four operations can be pending at any one time and these can be a mixture of arithmetic of logical operations in any combination of the three number bases.

The TI Programmer represents negative numbers with a two's complement code in both the hexadecimal and octal bases. A one's complement key on the calculator also facilitates working with computers which use that type of arithmetic.

The calculator can perform bit by bit logical operations on numbers in hexadecimal or octal. Included are AND, OR, Exclusive OR and SHIFT operations.

Users will also find the calculator useful as a regular day-to-day, four-function machine in the decimal base. The unique combination of functions make it an excellent tool to handle the bulk of a programmer's personal and professional requirements. Its speed in all number bases is essentially instantaneous.

A constant mode can be used for all kinds of repeated arithmetic and logical operations. Memory features include store, recall and sum.

Other features include an eight digit LED display, rechargeable battery, vinyl carrying case and AC adapter/charger.

The calculator is being test marketed initially on a direct-mail basis from Texas Instruments.

For more information contact: Texas Instruments Incorporated, Inquiry Answering Service, Box 5012, M/S 84 (Attn: TIP), Dallas TX 75222.

TI HEX CALCULATOR: SI OR NO?

Dear Ed: 77-July 1

On page 11 of the March issue of *DDJ* you indicate that TI no longer makes a hex calculator. Well I don't know if they *make* one or not, but they are sure 'nuff *advertising* one. Last week they sent me a brochure describing their "TI Programmer" which speaks octal, hex, and decimal and sells for a mere $49.95 plus tax. For more info, write to Box 53, Lubbock TX 79408.

Sincerely,

Jim Day
17042 Gunther Street
Granada Hills CA 91344

ONGOING PRAISE FOR GODBOUT & MORROW'S STUFF

Dear Dr. Dobb's, 77 Aug 5

In the may issue, you ran a letter which was somewhat critical of the Godbout 4K memory boards. Since my initial version of my KLUGE-8 was entirely populated by products of the Godbout/Morrow stable, and I have been very happy with their products, I feel I must issue a rebuttal. Therefore:

1. Solder mask. Quite unnecessary on the 4K board. George sacrifices the ground plane on his double sided boards to leave the back clear of crowded traces. This requires a few extra bypass caps, but sure makes soldering easy. These boards are easy one-evening projects.

2. Regulators & heat sinks. The three regulators are on a very large foil area at the top of the board, and don't run at all hot. So, why add complications?

3. Open Collector vs. Tri-State output. I'm no engineer, but for the life of me, I can't see much difference in an O/C gate that's high, and a Tri-State buffer that's "off", except the speed. Since the open collector gates are fast enough, and were more readily available when the board was laid out, why change? I am open to education on this point, but they do work OK.

4. Schematic. I don't know what better means, but the Godbout print has told me everything I ever asked of it.

And some notes on the other Godbout/Morrow products:

The Morrow Cassette Interface board. This is a really sweet package, 'tho it took me a while to realize it. I bought one, and decided it was too slow, so I sold it. After a few other tries at the problem, I decided that slow as it was, The Morrow was very reliable, easy to program around, and didn't consume any of my scarce RAM. So, I bought another one, and found the price had gone up while I was messing around with the others. On this board, after loading it with all the extras (3rd cassette channel, parallel port, and TTY/RS232 port) I did decide to add a heat sink on the 5-volt regulator, which ran a little warmer than I liked.

The Godbout 8K memory board. This one has a solder mask and heat sinks. I have a borrowed one now, and just ordered one for myself. This board is considerably more closely-packed than the others, but there still are no squeeze-throughs on the soldering side, and with the mask it isn't at all a bad board to work on.

The Morrow's CPU and front panel. Again, for the amount of stuff on it, this board is surprisingly easy to work on due to the open layout on the soldering side. I have had it running for almost a year now, and have only two minor complaints with it. It could use an extra mounting hole near the keypad, and give a little more rigid assembly, and I still haven't figured out how to trick it out of it's on-board startup routine and into the Boot loader in the cassette board ROM without at least 17 keystrokes.

I have had excellent results from everything I ordered from either Godbout's or Morrow's Micro Stuff, and recommend them highly.

Sincerely,

Tiny Matthews
2028 Merrily Drive
Montgomery, AL 36111

dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 19 **October 1977** **Volume 2, Issue 9**

CONSUMER INFORMATION

A Detailed Report on Product Delivery Complaints – *Tom Williams, Assistant Editor* 6
a report on a multitude of complaints received from DDJ *readers*

The Heath H-8: Pro & Con 10
a negative evaluation & a detailed response

Poly Doc Corrections -- *Richard B. Emerson* 15
additions & corrections to Polymorphic Systems' documentation

SOFTWARE

Editor & a Small O.S. for the 8080 – *Lorin S. Mohler* 13
notes on a low-cost software package

Dijkstra's Minilanguage – *Neils Laughlin* 19

GRAFX – A Graphics Monitor for the 8080/TV Dazzler – *Jonathan W. Mills* 21
complete documentation & source code

Alphanumeric Plotter for Poly Video Display – *Arthur Armstrong* 28

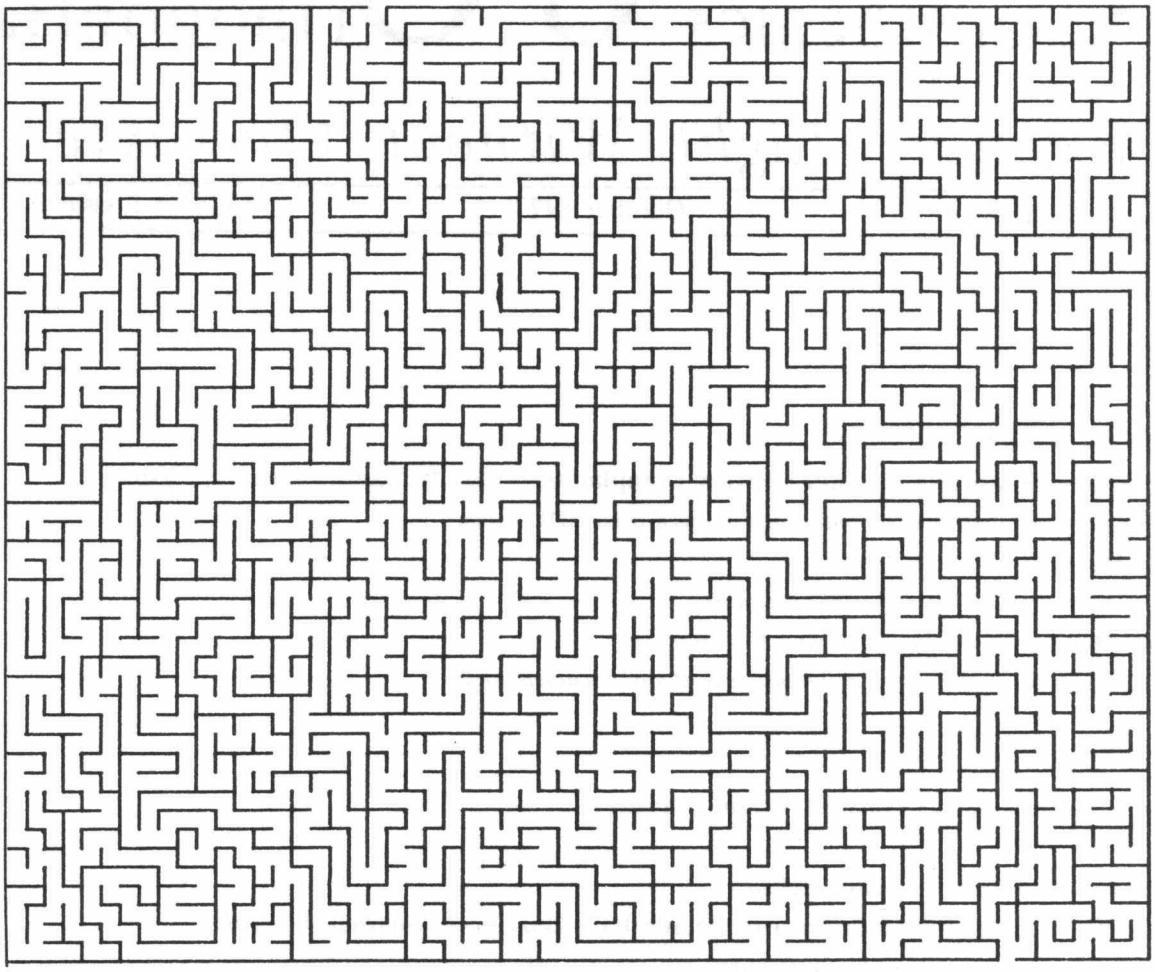
Utilities & Music on the Cosmac Elf – *Edward M. McCormack* 30

PROMable 1K Operating System for RCA's Elf – *Edward M. McCormack* 34

Multiple Column Accounting – *Charlie Pack* 36
a journal balancing program

OPLEGL Correction & a 6502 Scanning-Debugger – *H. T. Gordon* 42

IN



© David Dameron 1979

16-YEAR OLD PLACES HEX ON OCTAL

Dear Dr. Dobb, *Dated: 77 Sep 13*

I would like to express my opinions about the two conversion formats of computer address and data information currently implemented on nearly all micro- and mini-computers. I believe hex to be dominant over octal in many aspects:

1. Since all computers' address and data word sizes are in multiples of four (4) bits, octal representation often wastes a digit; that digit representing one or two bits instead of three.
2. Hex never wastes a digit: each nibble (four bits) of a number is represented by one hex digit.
3. In the instance of the PDP-8, a twelve (12) bit minicomputer, one address word can be represented by four full octal digits without waste. But – only three hex digits are needed to stand for twelve bits, also without waste.
4. When one memorizes an instruction set, types in an object program, or prints out an assembler listing, thousands upon thousands (1000/1000) of wasted characters are spewed out, typed in, or memorized unnecessarily when the octal format is implemented instead of hex (expecially the amazingly unorthodox 'split' octal is used).
5. Octal also wastes time in many other ways. . . I wouldn't have had to waste the time to compile this letter if octal did not exist.
6. There is even an instruction on the 8080 and Z80 microprocessors which aids in the hex display of a byte by adding six to each nibble of the accumulator when greater than nine (if carry is set).
7. As I said before, four bits are effectively represented by one, not one and a half, digits in the hex format. Two hex digits represent one and a half bytes, four hex digits represent a chomp, and eight hex digits represent four bytes, two chomps, or whatever ails you. . .
8. 'Octal' is a two syllable, five letter word, whereas 'hex' is a one syllable, three letter word. (Although it is short for 'hexidecimal' which is a five syllable, eleven letter word).
9. I find no good reason to use octal as a printing format instead of hex.

In conclusion, I would like to request that more output listings of assemblers and octal memory dumps, etc . . be hex dumps and hex assembly listings. It will save space and time for all!!!

By the way, DDJ and PCC are the best micro and mini computer publications available!!! Keep it coming!!!

Hexidecimally yours,
Mark J. Nitzberg
4D 41 52 4B 20 4A 2E 20 4E 49 54 5A 42 52 47
115 101 122 113 040 112 056 040 116 111 124 132 102 105 122 107
15 South Drive
East Brunswick, NJ 08816

P.S. Believe it or not, I am *sixteen* (0fh, 026q) years old by some coincidence. Also, note that *sixteen* more characters were required to represent by name in ASCII octal than hex!!

DR. DOBB'S ADOPTS PETERBOROUGH PRINCIPLE

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

Ever since I first saw *73* magazine (a ham radio mag out of Peterborough, NH) – and noticed that they used the "radical" scheme of simply numbering their issues sequentially, rather than using a Volume number and Issue number – I have been thinking of switching *Dr. Dobb's* to this more convenient issue-numbering practice. It's useful 'cause from time to time, we get queries about the "missing" 11th and 12th issues for each year (we only publish 10/year). It's particularly appropriate right now because I have managed to get so far behind in our "regular monthly" publication schedule (but, we're catching up fast, now that we've taken on Tom Williams as our Assistant Editor). And, it's more regular and orderly – as is appropriate for a computer publication – than the Volume/Issue practice. Ya see, step 1 was to stop using Roman numerals for Volume numbers (some publications still do). Now, step 2 is to rid ourselves of this sector/byte addressing in favor of simple, linear byte addressing. Note: For those to whom it matters – like the Post Office 2nd Class clerks – we will continue to give the official month of issue in the Staff Box on page 2.

COMPUTER NETWORKING SYMPOSIUM ADVANCE PROGRAM AVAILABLE

News Release *Received: 77 Oct 25*

The advance program for the COMPUTER NETWORKING SYMPOSIUM is now available. The COMPUTER NETWORKING SYMPOSIUM will be held in Gaithersburg, Maryland, December 15, 1977. The symposium will explore the planning, implementation, evaluation and use of both large and small scale computer networks. Important technical advances will be reported and the progress of existing networks will be reviewed. For a copy of the advance program write to COMPUTER NETWORKING, Post Office Box 639, Silver Spring, Maryland 20901, or call 301-439-7007. The COMPUTER NETWORKING SYMPOSIUM is being cosponsored by the IEEE Computer Society and the National Bureau of Standards.

MORE ON MODEMS FOR CASSETTE RECORDING

(No, There are Better Cheaper Ways)

Alan Parker
9660D Barrel House Road
Laurel, MD 20810

Received: 77 Jul 7

What's all the fuss about using an acoustic modem as a tape interface. With a good tape recorder (not a $20 wonder) it will work, more or less. Modems cannot tolerate any frequency distortion, which is what you get from tape recorders. The telephone lines don't tend to change frequencies below 7Khz or so. I found that a model worked quite well with a Wollensak reel to reel type deck, but did not work at all with my cassette unit (which is cheap).

Now if I had a modem (which I do) I would be inclined to use it as a modem (which I do). For people needing a cassette interface, who like me don't believe in spending $100 for a simple circuit see the April 1, 1976 issue of *Electronics*, page 83. There is a cassette interface published there that is simple (6 chips), cheap (less than $15), and very low on power (CMOS). And it works. It can tolerate speed variations of up to 5% (my $20 special rings in below that). This circuit also requires no adjustments.

GODBOUT RAMs SOL & GREEN PRAISE

Dear Jim: *Received: 77-Sep-9*

Just discovered that you can't use the Godbout 8K RAM board with the SOL SYSTEM unless you tie the PROTECT line low on the board. It flips back and forth if left unterminated and writes erratically. Otherwise a good board and a bargain at $163.84. This problem will affect any computer that doesn't use the PROTECT line.

I am a Ham as well as a Computer Hobbyist and I have had many dealings with both John Craig and Wayne Green. In fact, I have sold them at least a half dozen manuscripts. I feel that I have always been treated fairly and honestly. Wayne likes to keep things stirred up but that is needed sometimes. As to your comments on the Computermania I'll have to let them stand since I can't dispute what you say.

73,
Rod Hallen
Road Runner Ranch
P O Box 73
Tombstone, AZ 85638

A Detailed Report on Product Delivery Complaints

By Tom Williams

Tom Williams, Assistant Editor, *DDJ*

In the June/July issue of *Dr. Dobb's Journal*, due to a rising number of complaints we were receiving, we initiated a project which was somewhat facetiously called a "pseudo-contest" for the manufacturer with the longest delivery delay. We asked readers who felt they had been wrongly dealt with to submit documented details for publication. They were to include copies of any cancelled checks, invoices, original order form and any correspondence on the matter in question. In addition, we asked everyone submitting such documentation to sign a statement saying that at the time they placed the order they had no indication there would be more than a one-month delay in delivery and that they had not received an offer of immediate refund of their advance payment when an undue delay became apparent.

We are delighted to report that response to this project has been only nominal. Perhaps it was a bit too much to ask of people when the most they could expect was not immediate satisfaction but merely the airing of their grievance in public with the documented response of the manufacturer. There were, on the other hand, enough people who were sufficiently motivated – or frustrated – to send in some very well substantiated case histories. Thus, we have concluded that their stories are representative of something. Conversations with manufacturers – mostly by phone and mostly with the presidents of the companies involved – revealed certain common denominators which will be summarized in the conclusion of this article. We also found, not suprisingly, that there are generally two sides to every story and that while this may not constitute fairness to someone who's had his $400 tied up for six months, neither is it a case of out-and-out fraud. Of course, there are exceptions.

Once again it should be made clear that this investigation concerned itself with *delayed delivery* of merchandise primarily in cases where some form of advance payment had been made. It was *not* our intention to get into questions of the quality or suitability of the products themselves. It should also be noted that a goodly number of customers who had waited what they felt was an unreasonable amount of time were nonetheless unwilling to accept a refund that was offered. They wanted those goodies that badly.

The case histories that follow are ones we received that were particularly well documented and which we feel, based on more general information available to us, are indicative of what is going on in the market as a whole. It should further be pointed out that the companies mentioned were not singled out as particularly major offenders; they are the cases which people sent us, and due to the small actual number of cases we received *they can only serve as examples.* Nonetheless, the cases we recount here are real enough in themselves that useful information can be obtained from them.

Ohio Scientific

On September 27, 1977, we received a letter dated September 22 from Mr. Troy N. Belote of Fayetteville, Arkansas, complaining about the delay in delivery of a Sanyo video monitor he had ordered from Ohio Scientific Instruments of Hiram, Ohio. The order had been acknowledged over four months earlier on May 19, 1977. A telephone call to Mr. Belote on October 10 determined that he had finally received his monitor on October 5, but he was still dissatisfied with the service he had received from OSI. He said that he had contacted the Arkansas Attorney General's office and filed a consumer complaint. A copy of the acknowledgement from OSI gave the estimated shipping date as 45 to 60 days. The form also stated that the customer would be notified if there were any delays beyond the date given. There is also no evidence in the documents sent in by Mr. Belote that OSI attempted to voluntarily notify him of a delay in delivery. There are, however, copies of two phone bills showing calls Mr. Belote made to OSI.

A telephone conversation with Charity Cheiky of OSI on October 10 revealed that Mr. Belote had won a 460-Z board offered by OSI as one of the prizes in the 1977 *Kilobaud* sweepstakes and that OSI had offered $99 credit toward some other item in their catalog if the winner did not find the 460-Z suitable. Mr. Belote chose the Sanyo 9" monitor and paid the $54 difference. When I asked Ms. Cheiky why it took so long to ship such a standard item, she said it was not something they sell a lot of and on their last order from their supplier, they had only received a partial shipment. She did not say why OSI had failed to notify the customer. The total time for delivery in this case was from May 19 to October 5, 1977, or 139 days.

ECD Corporation

Since as early as January, 1977, ECD Corporation of Cambridge, Massachusetts, has been running full-page color ads for their Micromind computer system which sells for $987.54 without options. Included in the ad is a coupon where the customer can check a box labelled: "Fantastic! Check enclosed: $987.54. Shipping paid by ECD." This same ad appears with a few exceptions in the inside back cover of *Byte* up through the September, 1977 issue. There is a new ECD ad on the inside back cover of the October issue, the theme of which is: "It's th' backlog that usually gets ya'. . ."

On May 20, 1977, George Zuras of Redwood City, California, wrote a check to ECD for $415.96 as down payment for a Micromind system. He was told at the time that delivery would be three months with the balance due two weeks prior to delivery. When he called back three months later, in August, he was told delivery would be delayed until September and has since been told that his computer won't be delivered until December, seven months following his order.

When I talked to Mr. Zuras by phone on October 10, 1977, he did say that ECD had offered to refund his money if he was unwilling to wait longer than 30 days. Mr. Zuras said he thought ECD was being above-board but was still miffed that they had had his money for so long. He was, on the other hand, unwilling to take the refund offered because he wanted that computer.

A telephone conversation the next day with Ed Costello of ECD largely confirmed Mr. Zuras' story. Mr. Costello said that their prototype system was working and that they now have a production backlog. In addition, he said ECD had difficulty with some of its subcontractors concerning production of a redesigned PC board. Mr. Costello also reconfirmed December as the tentative delivery date.

The two sides of the story appear to match and Mr. Zuras – though less than delighted – says he is willing to wait.

Polaris Computer Systems

Polaris Computer Systems, reportedly of 3311 Richmond Ave., Houston, Texas, 77098, seems to be in the check-cashing business – nice work if you can get it. $25 is not the moon, but all Laris Pickett of Douglas, Kansas, received for his down payment was his cancelled check and a registered letter returned "undeliverable as addressed". We checked with Directory Assistance and found that the original number for Polaris Computer Systems had been changed. The new number, however, had been disconnected. Other attempts to locate or contact Polaris via telephone information were fruitless. Caveat emptor!

The Digital Group

TJB Microsystems is a dealer in Edmonton, Alberta, and has had difficulty ordering from a couple of suppliers in the U.S. TJB's problems have been aggravated by the fact that smaller companies have no one trained to deal with customs formalities. That old nemisis, the U.S. Postal "Service", also played a certain part in TJB's troubles.

On February 9, 1977, John Atwood of TJB placed an order with Digital Group for a Z-80 system, a power supply, and a printer with power supply and cabinet. TJB enclosed a bank draft in US funds for $2166. The Z-80 system was shipped on April 7, the power supply, memory boards and CPU cabinet were shipped April 26 and the printer was not shipped until August 8. TJB supplied copies of the export forms to document these dates.

I talked with John Atwood on October 11 and his main complaint was that he said he had never been contacted voluntarily by Digital Group about the shipping delay nor had he received a written explanation. He added that one cabinet contained in that order had yet to arrive.

The next day I called Dick Bemis, President of the Digital Group, and asked about TJB's problems. He seemed quite forthcoming and eager to be of help. The main problem, he said, had been in production delays involving suppliers and, at the time of our conversation (October 12), Digital Group was discouraging printer orders because of the problem in supplying cabinets.

Mr. Bemis said they had experienced delivery problems through July and August but that the situation had improved since September except for the cabinets. The steps Digital Group had taken to improve delivery were to hire 40 new people and to expand their plant facilities.

When I asked about informing customers about delays, he observed that would be a major dedication of resources – resources that might better be used for production. He went on to say that if a partial shipment had been made, the paperwork involved with keeping the customer notified becomes cumbersome. At this point we cannot repress the editorial observation that this sounds like the kind of task you could train a computer to do. . .

At any rate, according to Dick Bemis, Digital Group has solved most of its production problems and customers should not have to wait more than four weeks for delivery.

Miniterm Associates

TJB's woes continue with an order placed January 12, 1977, with Miniterm Associates of Bedford, Massachusetts. They ordered six of Miniterm's Merlin displays and four of the MBI firmware and enclosed a bank draft for $1336, U.S. The order was acknowledged January 21. We also have on hand a copy of a letter from Dave King, President of Miniterm, dated January 21, saying that partial shipment would be made "next week" with the remainder to be shipped by February 14. The first Merlin interface was shipped February 22. The next shipment of three Merlins was March 23 and the last shipment was dated July 1.

John Atwood told me that TJB had finally received everything, but only after the Postal Service had lost one box, reimbursed Miniterm, and Miniterm had finally reshipped. The impression I got from Dave King when I called October 12 was rather vague. He cited unspecific "production delays", the fact that they had moved, "paperwork problems", and a lost invoice. John Atwood is conviced that Miniterm is totally confused.

When I asked Dave King about any attempts on Miniterm's part to notify the customer about delays, he said the customer had called several times, and he returned a phone call. He did say, however, that the delivery problems had been overcome and if I were to order a Merlin that day, he could ship in 2–7 days.

Associated Electronics

It is a sad fact that some customers in the microcomputer marketplace wind up getting just plain "stuck". One unfortunate who didn't take it lying down was Herbert Shore of La Jolla, California. He purchased a 16K memory kit from Associated Electronics of Garden Grove and Fullerton, California, on April 17, 1977, while attending the First West Coast Computer Faire. He paid the full price of $314.95 for the unit using his BankAmericard, and has supplied us with copies of the sales slip and the statement showing that Associated Electronics had posted the sale on April 20. On June 23, 1977, Mr. Shore sent a letter by certified mail, cancelling the order and demanding a full refund. On August 16 Mr. Shore filed a suit in Orange County Small Claims Court to recover his money, court date set for September 22.

On October 14 I called Brice Ward, one of the owners of Associated Electronics to ask about Mr. Shore's complaint. When he learned the nature of my call, he became quite hostile and said he would not discuss it. He said we could handle it any way we liked and hung up on me. Well, the way we like to handle it is to report what happened, so there it is.

I called Herbert Shore the same day to check on the outcome of his court claim. He told me that Associated Electronics had failed to appear and he had gotten a default judgement. This does not seem to be an isolated incident.

A reader sent us a clipping from the October 5 edition of the *Valley News* of Van Nuys, California, where a reader assistance column titled "Valley Open Line" received a letter from a reader who had had a similar experience with Associated. The reporter who called Mr. Ward to inquire about his reader's complaint got a lecture on minding his own business and was also hung up on. In view of all this, we cannot recommend that you associate with Associated.

Southwest Technical Products Corporation

In this particular case, it is not so much the delay in delivery – which did exist, but was less acute than other cases – that is of interest. Rather, the interesting facet of this

case is the effectiveness of Federal agencies when presented with a consumer complaint. Mr. Joseph O'Loughlin of Huron, Ohio, ordered a SWTPC 6800 computer on June 7, 1977, and enclosed payment of $395. The order was confirmed on June 10, stating that the shipping delay was four weeks. When he had not received the order by July 22, Mr. O'Loughlin called and was told that there would be an additional four weeks delay. He then requested the order be cancelled and demanded a refund. The refund arrived within nine days (two more than the letter of the law) and Mr. O'Loughlin told us that he really couldn't call that unreasonable.

Mr. O'Loughlin also filed a complaint with the Federal Trade Commission and received a letter in reply to the effect that the FTC had limited resources and manpower and could only take action in cases affecting substantial numbers of customers, and that his complaint had been entered into a computerized system.

I called Dan Meyer, President of SWTPC, about Mr. O'Loughlin's complaint. He said he couldn't understand why the customer had not been notified, as delay notices are generated automatically by their computer. (At this point we can imagine Mr. O'Loughlin despairing of computers and taking up needlepoint.) When asked about the reason for the delay, Mr. Meyer said that they had something of a problem since last spring, due to an unanticipated demand creating an order backlog. He said current delivery delays were four to six weeks (at the time of our conversation, October 21). Part of the problem, he indicated, was due to a supplier of memory chips (the only one in the world who makes them) having had a yield bust. The unavailability of parts had thrown off the whole production schedule. When asked about communication with customers, he said they get so many orders, so fast, that they must decide whether to devote their resources to production or to customer service. This again reveals a problem to which I shall return at the conclusion of this article.

Processor Technology

Dan Wingren of Dallas, Texas, appears to be a very long-suffering gentleman. On November 26, 1976, he ordered Processor Technology's 8K BASIC and says he was told at the time that the tapes would be shipped in ten days or two weeks. He paid for the order in the amount of $66 with BankAmericard. After various phone calls and letters and after cancelling his original order, he finally received a notice dated March 17, 1977, saying that 8K BASIC would be available May 16 and that customers would be notified of any schedule change. He subsequently reinstated his order for 8K BASIC, TREK, and Gamepack, as well as for a subscription to the P.T. newsletter, *Access*. He then received a letter, dated March 30, 1977, from Ralph Palsson, Processor Technology's Customer Service Manager, informing him that 8K BASIC would be delayed – probably until mid-May.

On July 11 Mr. Wingren sent an angry letter demanding either immediate delivery of his order or a prompt refund. On July 22 his BankAmericard account received a credit in the amount of $70, the full amount of his reinstated order.

I talked to Ralph Palsson of P.T. about this long delay and about the story on Processor Technology's 8K BASIC, in general. He said that their BASIC was being totally written in-house. They were incorporating a lot of extra features (data files, text editing, etc.), and had run into a lot of software bugs along the way. A subsequent conversation with Bob Marsh, President of Processor Technology, turned up the fact that although development of 8K BASIC had been begun in late 1975, it was halted in mid-stream for a considerable period of time. He stated: "Substantial delays in delivery were caused by uncertainties and hassles caused by litigation with some members of the software team".

Ralph Palsson was able to give me some firm dates about delivery and assured me that Processor Technology had learned its lesson about advertising a product before it is actually available. Let others heed.

I was told 8K BASIC would be completed and ready for production by November 15, and they would be starting deliveries by the second week in December. With luck, Processor Technology will be more accurate in the future than it has in announcements of the imminent delivery of 8K BASIC.

In the meantime, it has come to our attention that a Byte Shop in Boulder, Colorado, is offering PolyMorphic's 11K BASIC with software patches that will allow it to implement on a Sol-20. I talked with Terry Flander of the Boulder Byte Shop and he confirmed that such a package was available and that it included all the features of Poly's 11K BASIC except the PLOT and RTC. It sells for $60. Those interested may write to The Byte Shop, 3101 Walnut, Boulder, Colorado 80301, (303) 444-6550.

Dr. Dobb's Journal

To no one's surprise, our own publication received a number of nominations for the Grand Prize in our pseudo-contest. For devious deeds by *Dr. Dobb's* we have reserved special treatment – a separate section at the end of this article wherein we summarize some of the complaints and offer our own explanations and lame excuses.

Others

Among some of the other complaints we received was one that could best be called a "can of worms", with both the consumer and vendor investing so much ego into the argument that the facts of the matter become blurred. In such cases, though there may or may not be a legitimate grievance, it seems best to declare both sides winners (or losers); send them to their corners, and refrain from further comment.

Summary

Let us first repeat that the responses we received do not constitute a significant statistical base. They should be viewed as documented examples of the type of undocumented complaints we have been receiving for some time. We do believe, on the other hand, that certain conclusions can be drawn from the material presented here.

The first and most glaring fact is that there is a definite lack of communication between manufacturer and customer. People send in an order with a substantial payment, receive an acknowledgement, and then wait literally months without the slightest word on the status of their order. Aside from being downright illegal, this is discourteous and detrimental to customer relations. *Many of the complainants I talked to*

said they would have been content to wait longer for their merchandise, if the manufacturer had just taken the trouble to let them know that their order would be delayed and what the problem was.

This "communications gap" has its origins in two different views of the market and what people expect of it. A mail-order customer may assume that ordering a computer system should entail no more problems than ordering a Wisconsin cheese. And indeed most of the ads in the magazines support this assumption. These ads, conceived for the most part by advertising agencies without sufficient supervision of the manufacturers, awaken expectations that are very often disappointed, and as the market moves from the technical hobbyist to the more general consumer these disappointments can only increase. *It must be incumbent on the manufacturers and on the publications* to educate their customers and readers concerning the nature of this industry and the possible delivery problems.

The fact is that this industry didn't even exist three years ago. It is also an industry that uses the very latest available technology, and depends on components that are available from only a few sources. If one such component should become unavailable for some reason (like a bee flying into an oven causing a plant shutdown) it can suddenly stop delivery of a product that a company had every reason to advertise as deliverable off-the-shelf.

Dan Meyer of Southwest Technical Products observed that it is not always the best idea to order directly from the manufacturer. Often, a store may be able to better anticipate demand and order stock in advance. If a dealer doesn't have it, it may not be available any faster from the factory.

The practice of advertising products before they are actually available has been common in the electronics industry for years, but it has never been the norm in the general consumer market and it will have to be stopped. Many companies are beginning to see the error of their ways and are halting the practice. Many of the manufacturers are simply inexperienced in dealing with such a large demand and have become, quite innocently, over-extended. "We just didn't anticipate the demand" was the answer to a great number of my questions.

The fact that things are so intensely competitive will, of course, ultimately work to the consumer's advantage. Companies which, innocently or not, do not satisfy their customers will not survive. And in a marketplace where it is difficult for honest manufacturers to establish themselves, dishonest companies may rip off a few unfortunates, but will not stand a chance in the long run.

Dr. Dobb's Does It Too

On October 4, 1977, we received a registered letter from a reader who requests that his name not be used. He had ordered a subscription to *Dr. Dobb's* and a copy of *Volume 1* in January. Having heard nothing by *April* 25, he sent a letter asking what had happened and received a letter from People's Computer Company dated May 17 which promised delivery. Our patient reader inquired once again in June, then again in August but to no avail. This latest missive, entering us in our own contest, has finally jarred a response.

I asked Bill Bruneau, Circulation Manager for People's Computer Company, about this complaint and how typical it was of the situation in general. He said that since June, there had been very little trouble getting people their copies of *Volume I*. Prior to that time we had also committed the sin of advertising *Volume I* before it was actually available. Additionally, there had been problems with record keeping.

This reader's problem with his subscription had been fixed in June, but his order for *Volume I* had somehow slipped past us. Well, there we are with egg on our face. Bill Bruneau tells me, though, that this is atypical. If someone genuinely hasn't gotten his copy of *Volume I* he sends it out by First Class Mail, but he has only had to do this in about 1% of the cases.

As for subscriptions, yes, there are still problems, but we are catching most of them eventually. The main problems arise if someone's subscription is entered incorrectly – as in a coding or keypunch error (human error, not the computer's fault) – or if we somehow miss an update, in which case it can take a whole issue to find the mistake. The Postal Service has also played a part in the problem. Delivery time *locally* can take as long as two to three weeks and transit time to the east coast can take as long as six weeks.

Additionally, there has been a severe problem with the Editor's failing to produce the issues on a regular schedule. Jim Warren, the Editor of *DDJ*, has managed to badly overcommit himself, and – even as *DDJ* has grown and has needed more of his time, he has often had less time. Jim's comments: "I have appologized for delayed issues before. I wish to repeat that very sincere appology. I have finally given up all hope of running *Dr. Dobb's* as a one-person anarchy, and have found an Assistant Editor. Due to this, I believe that our production problems – which are entirely my doing – are over, and our publication should be back on schedule by the November/December issue. . . That is, issue Number 20."

We are evolving a system which minimizes complaints, i.e., problems, which is far from perfect yet, but things *are* improving. Specifically, virtually everyone eventually gets a response, but correcting the "problem" has priority over a personal letter. One problem is that writing a letter is within 2 magnitudes (1%) of the subsciption price, and we can only afford two employees including Bill Bruneau to handle 14,000 subscriptions. The industry average is one person per 3,000 subscriptions.

The research I did for this article makes me optimistic that as our fledgling marketplace matures, things are improving in the microcomputer industry as a whole. Thus, I can close with the observation that we *are* all getting our act together.

MOSTEK/ZILOG FLAG FLAP

Dear Dr. Dobbs, *Received: 77 Oct 31*

I just bought a copy of MOSTEK's Z80 Programming Manual, thinking it would be a useful supplement to the ZILOG Technical Manual which came with my Digital Group Z80 CPU kit.

Well, it's not. But more to the point, I found a significant difference between the two chips. The ZILOG version sets the Carry flag when subtraction generates a borrow. The Mostek chip resets. This could lead to some very puzzling software behavior. Mind how you go.

Sure do enjoy *DDJ*, keep it up.

T. F. Caldwell
P.O. Box 116
Burgess, VA 22432

The Heath H-8: Pro & Con

Editor's Preface: We have received the following evaluation–criticism–of the H-8 from someone whom we know personally, and whose judgement we generally respect. At the author's request, we are witholding his name, however we will forward any correspondence that may be directed to him immediately upon receipt.

As this is a significant criticism of the H-8, we are following our policy of providing Heath the opportunity to offer a substantive and in-depth response to the criticisms. Heath's director of computer products, Lou Frenzel, has been given an early copy of the negative article (with the knowledge of the author), and we are publishing his response in its entireity.

We would very much appreciate additional comments and evaluations of the Heath products–compliments and complaints–from purchasers and users of the H-8 and H-11. We are convinced that, if Heath products have imperfections, Heath will do their best to rectify those problems. Heath has a long history of being the definitive electronics kit manufacturer, and we have no doubt that they will continue that tradition with their new computer kit product lines. Careful and responsible evaluations of their products–positive and negative–can only be of assistance to them.

A more personal note: I had mixed feelings about allowing the publication of this "blind" article. I would certainly not have allowed it had it not been that (1) I personally know the author and respect his technical judgement, and (2) he raises points and offers criticisms of the H-8 that–in several cases–I have heard from others, or have thought myself. Founded or unfounded, they are being discussed in the hobbyist community, and are thus worthy of being aired and addressed by Heath.

I have heard mixed feedback on the H-8 software. Some users have praised it; some have maligned it. I have generally heard praise for the usefulness of the front panel system on the H-8. Many of us questioned Heath's failure to adopt the "S-100 bus"–whatever that is (it's never been adequately defined, electronically or functionally), and Lou has provided the best possible response to that question–a response that is equally applicable to the other excellent non-S-100 systems, e.g. Digital Group, Apple, SWTPC, etc. The anti-octal comment raises the unending octal/hex debate (I prefer octal, 'cause I only have eight fingers–and two thumbs . . . and, mostly because it was my mother tongue on an old PDP-8). The lack of an extender board will soon be remedied by Bob Mullen of Mullen Computer Co. I tend to agree with the complaint about the lack of standard parts designations and slanted board mounting. Please note that the criticizing author maligned the H-8 for lacking an S-100 bus, while praising Apple . . . which doesn't have an S-100 bus.

Regarding Lou's rebuttal: A number of computer stores are beginning to refuse to back "mixed-vendor" systems that turn up problems due to unsuccessfully mixing S-100 "standard" parts. On the other hand, Heath would hardly refuse to warranty a stereo tuner simply becuase it was being used with a non-Heath stereo amp. The consumer market need not necessarily follow the tradition of the computer industry market regarding non-support for systems incorporating foreign subsystems. Incidentally, Godbout Electronics is about to market an H-8 memory board . . . at noticeably lower per-byte cost than Heath's memories. (This may be a faster-appearing 2nd source supplier than Heath was expecting. It will be interesting to note their response.) Lou commments that the tilted boards make it easy to get to components. I don't see how this can be the case when one needs to reach components on more than one card. I do believe that Heath uses first-line parts. Additionally, it is my impression that they test all *parts prior to inclusion in kit packages (I saw the testers when I visited the Heath plant in June).*

My conclusions are that: (1) The H-8 has some problems . . . as is inevitable of any new high-technology product. (2) Some of the criticisms are probably just–and, in some cases, are equallly applicable to other vendors' products. (3) The critical author's sarcasm should be taken with a grain of salt, and perhaps reserved for the entertainment of those of vitriolic bent. (4) In spite of the imperfections of the H-8, Heath is well worth considering–comparing against the imperfections of other manufacturers. And, Heath is to be praised without end for the quality of its documentation–an order-of-magnitude improvement over the majority of personal computing documentation I have seen, to date. – JCW, Jr.

A CRITICAL LOOK AT THE HEATH H-8

This is probably the worst 8080 based personal computer system to come along in a while. While Heath (a division of Slumberger) is a reputable company that has put out (and continues to put out) many fine products backed up by some of the best assembly, check-out and operational manuals in the business, this one is a dog. It is badly designed from the printed circuit layout to the cabinetry to the operating software.

The first thing that any potential customer should know is that the H-8 is not in any way compatible with the S100 bus. Heath cannot ever afford to put out the hundreds of different boards that dozens of aggressive companies are producing. Heath claimed that they invented their own bus so that they could produce boards that would be cheaper than the competition. Wrong. Their product is in the upper price ranges. While they were looking at MITS prices, independents such as Godbout (and others) were showing the way to really popular low cost items.

The boards are tilted back at an angle as is the front panel. Then they are held by an aluminum bail that is screwed into

RESPONSE TO CRITICAL ARTICLE ON THE H8 COMPUTER

Louis E. Frenzel
Director, Computer Products
Heath Company
Benton Harbor, MI 49022

77-Oct-25

The author's first objection has to do with Heath not adopting the S100 bus. When the development of the H8 began in 1975, the term S100 did not exist. Nor was that bus widely used as it is now. The MITS Altair had only recently been introduced. It was the only microcomputer available. Naturally we looked at that machine and studied the bus structure.

The S100 bus is an old bus design that assumes you are using an 8080 without the system controller (8228/8238). There are many unnecessary signals on the bus. Less than half of the total 100 lines on the bus are useful or needed. In addition, the bus has a variety of noise and timing problems. It is also an expensive bus in that it uses high cost printed circuit board edge connectors. We knew that we could produce a lower cost bus with greater electrical integrity. That is what we set out to do as others have done (Digital Group, etc.). We

CRITICISM, *cont.*

each board. The boards are also attached by another screw at the bottom. Changing boards is a pain. The connectors and the mother board run along the right side of the case. Female connectors are soldered to each PC board and male connectors stick out from the backplane (sideplane???). This makes it easy to short power and other lines. This also means that they can't use the simple and effective printed fingers the S100 boards exhibit. It means more parts, more difficult assembly, and higher cost. Furthermore it is possible to push the boards onto the connectors in the wrong position. This is regrettably easy to do. The S100 boards are not symmetrical and cannot be put in the wrong way (so long as there are card guides as on almost all S100 systems). So Heath is wrong again. The sideplane also makes an extender board a sort of right-angle affair. Heath doesn't offer one, and it will have to be rather mickey-mouse even if they do. There is no reasonable way to extend a card out sideways as the sides are firmly attached to the computer. It is not like having a lift-off top.

The layout on the cards, in an attempt perhaps to make them easy to build with your feet, is wide open. There are a few chips here and there separated by acres of nothing. Thus when everybody is coming out with 16K to 64K cards there is Heath with an 8K card. It comes with 4K; the extra chips are an option. At least Heath provides sockets for all the chips.

The front panel is not as primitive as the Altair and Imsai. It has a calculator style keyboard (which bends the PC board when you press a button firmly) and some numeric LEDs. Everything is in OCTAL, whereas the 8080 world is 85% Hex by now. Why didn't the people at Heath notice what the really smart designers – like those that produced the SOL, the APPLE, the Poly-88 – were doing? Were their heads in the sand? The few clever ideas (like a beep to indicate a command accepted) are far outweighed by the deficiencies of the H-8.

Heath, to be consistant, should now do to all their other product lines what they have done to the computer. For example: they shouldn't use standard audio connectors on their hi-fi equipment. That way you'd have to buy Heath's connectors. They could have the impedence of their audio power amplifier outputs set to 217 ohms, so you'd have to buy Heath 217 ohm speakers. Given a little more creative marketing, Heath would re-do all its manuals to use non-standard units of voltage and current so that you'd have to use Heath-designed meters to check it out. They've already gone a bit in this direction: instead of giving a familiar designation like 7404 on the schematic diagrams, all they show is the Heath house number. Thus, when debugging the circuitry you have to constantly reference their parts lists, then look up the pin-outs in the charts. While the practice of using house part numbers might be OK for some other things, many practitioners know the various ICs by the more conventional numbers, and can read a schematic easily if they are given.

Heath has gotten a bit too big for its britches with the H-8. Let's hope they do better with the H-11, which does use the standard Q-bus like any other LSI-11 computer. The H-8 is not recommended and Heath should have its corporate head examined for springing this kit on its trusting customers.

RESPONSE, *cont.*

felt that it was a poor business decision to perpetuate this design in a new product. Progress is made only when someone is willing to make changes and improvements.

Since we elected to use our own bus, we took the position that we must support it. Users that select the H8 will want the wide variety of memory, interface and other accessory cards available for S100 bus computers. Your author makes the statement that "Heath cannot ever afford to put out the hundreds of different boards that dozens of aggressive companies are producing". He is wrong since we *can* afford to produce a wide variety of accessories. Those accessories will not run into hundreds, of course. If you take a critical look at the two hundred or so different cards available for S100 computers, you will find that over 95% of them are memory cards of some configuration. The remaining cards are mostly interfaces for serial and parallel I/O. A few special cards such as video interfaces, floppy disk controllers, the TV dazzler, DA/AD, speech and music synthesis are also available. You can be sure that many of these cards will be available for the H8.

Your author also states that "Heath claims that they invented their own bus so that they could produce boards that would be cheaper than the competition". Not true. We made no such claim. The H8 cards are slightly larger than S100 cards but they contain about the same amount of circuitry. They are priced approximately the same as equivalent S100 cards. Naturally the pricing varies with the type of board, the components used, the quantity sold, the development effort involved and many other factors. Heath has always offered reasonably priced equipment, and we intend to maintain that policy with our computer product line. Keep in mind that Heath does not use surplus or graded parts to effect a one-time low price. Only high-quality first-line parts are used.

A major reason for not choosing the S100 bus is that it is too difficult to support. We go to great lengths to help our customers get their computer kits together and working properly. This can be done only if we can be sure that the customer is using *all* Heathkit equipment, equipment designed to work together. Like other computer manufacturers such as DEC, Data General and IBM, we simply find it impossible to support "foreign" attachments. It is particularly difficult with the S100 bus because of the wide variety of memory and interface cards available and the many variations in the bus signals. As we have discovered in evaluating S100 bus systems and cards, some of the various accessories work in some computers but not in others. It is impossible for us to guarantee the proper functioning of our computer equipment with attachments of someone else's design. I believe that you will find this attitude exists amongst S100 computer makers as well. MITS, IMSAI, Polymorphic, Processor Tech, Vector and others certainly do not look forward to supporting their computers with "foreign" accessory cards. There are too many variations and some combinations simply do not work.

Some computer stores have already recognized this problem and are beginning to take a hard line on selling and servicing S100 equipment with different mainframes, memories and interfaces. The concept having many hundreds of accessories available for your S100 bus computer is excellent. But in practice it simply does not work out. The choice is more limited than one suspects and care must be taken to ensure that the various accessory cards selected do indeed work properly with the mainframe.

Many of the problems that computer hobbyists are having today are associated with mixing S100 accessories and mainframes purchased from different sources. Some of these problems simply cannot be resolved. Neither the accessory

RESPONSE, *cont.*

manufacturer nor the mainframe manufacturer want to take the responsibility for getting the complete system to work. As a result, many hobbyists are simply being frustrated and discouraged. This is just the kind of problem we are trying to avoid. I think other manufacturers would do well to adopt a similar policy if the hobby is to grow and prosper. If a hobbyist wants an S100 bus computer, he should at least buy all his cards from the same source if he wants to ensure getting a workable system.

In my observation, interest in the S100 bus has already peaked. My own research reveals that the trend is definitely away from the S100 bus and toward newer, improved designs. The S100 was first and it works. But it isn't really a standard and it has too many liabilities. I don't think that most hobbyists will stay with S100 when something better comes along. Hobbyists traditionally follow developments in technology (e.g. "hams" with amateur radio equipment) and want progress as much as industry. S100 will probably never die, but it will fade quickly.

Given another choice, I doubt that we would select the S100 bus for our H8 computer. Perhaps it appears as a bold and incorrect decision to many, but in our opinion it is the best choice and we intend to support it. An H8 customer will not be left out of the action.

The author's comments on the physical design of the H8 tend to fall mostly in the nit picking category. If he had actually built and used a production H8 he would certainly realize that the items mentioned are indeed not problems.

The card mounting scheme was used to achieve a low-profile cabinet. The side-mounted bus and tilted cards gave us the desired physical dimensions. The mounting bracket on each card is an excellent heat sink for the voltage regulators and at the same time provides extremely stable physical mounting characteristics. To change a card you simply remove two screws. There is little or no need to change boards frequently. Only during the early stages of construction and checkout may possible board changing be necessary.

The connectors used on the PC cards and on the bus mother board in the H8 are a much lower cost combination than the edge connectors and the costly tooled tongue S100 boards. Even though they are lower in cost they are certainly not less reliable. In fact, manufacturer tests indicate that they are far more reliable than edge connectors. Greater reliability at lower cost is excellent value. True, the male pins do extend outward from the bus and if you are not careful tools, wire, solder and other metallic items could short the pins. But the pins are neatly out of the way on the side of the computer, and we have never experienced a problem. If one is reasonably careful, I doubt that any trouble will be encountered.

Several other personal computer manufacturers use a similar connector scheme. The connector system used in the SWTP 6800 computer is practically identical, and to my knowledge they have never encountered problems. It is also possible to have similar problems with edge connectors. It is just as easy to drop a piece of wire or solder down into an edge connector and also create a short. Regardless of the type of connector used you have to be careful when working around them.

Incidentally, it is not possible to push the boards onto the connectors in the wrong position. If your author had constructed an H8 board he would have found that each is keyed to prevent this problem.

As for an extender card, we have yet to find a reason to offer one. Because the cards are tilted, the components are easy to get to. Troubleshooting a card is usually done by placing it in one of the rear slots and removing any unnecessary cards. This gives full access to the card with no extender needed. The straight up and down design of most other systems makes it extremely difficult to get to the circuitry on the card and an extender card is needed. Extender cards are "bad news" because they typically upset the transmission line characteristics of the bus and introduce or mask problems. We avoided that.

The author also criticized the wide open construction of our cards. Incredible! This is done deliberately to make the card easier for beginners to build. Solder pads are larger and the copper lands on the card are more widely spaced. This makes foil shorts less likely. The easier a card is to build, the more likely the customer's success will be. Heat build-up is also less of a problem. This certainly leads to greater customer satisfaction. We will continue with our open card design as much as we can. Yet where higher density is needed, we will certainly use it.

Incidentally, the front panel PC board does *not* bend when you press the keyboard. If the panel is mounted properly with all hardware in place, it is solid as a rock.

The author mentioned 16K and 65K memory cards in contrast to Heath's 8K card. We felt that the 8K design was optimum from a size and pricing standpoint. The 65K cards are ridiculously expensive. The average hobbyist simply cannot afford them. The 16K memory cards are also expensive for beginners who want to start with less. As a hobbyist learns more about computers and becomes proficient in programming, he will naturally want to expand his memory. This is when the 16K card becomes realistic. At least it is realistic from a size standpoint. Even then the hobbyist must convince himself (or his wife!) to spend many hundreds of dollars. It's a lot easier to justify an additional 4K or 8K than it is a full 16K. We do see the need for a higher density memory card, and a 16K memory card is soon to be announced for the H8.

The octal versus hexadecimal issue continues to be a strong point of argument amongst many hobbyists and manufacturers. For those of us who have been in the computer industry for many years, it is a standing joke to argue the pros and cons of octal vs. hex. There is no correct solution to this issue. Perhaps if it was important someone would have solved it by now. Either octal or hex works well. It seems to be all in what you get used to.

We chose octal because it fits best in machine language programming with the 8080 instruction set. Octal is also more decimal-like and therefore is more intuitive for the beginner. Octal seems to be easier to grasp and use when first learning such notation.

Octal is also used on the LSI-11 which is the heart of our H11 computer. We felt it would be desireable to be consistent despite the differences in the two machines. Octal can't be all bad since the two largest minicomputer companies, DEC and DG, still use it exclusively over hex.

Frankly, I think that any computer professional or hobbyist should be able to work with either octal or hex. It is silly to get emotional when one is just as easy to use and understand as the other. Besides, once you learn machine language programming you are anxious to get away from octal or hex coding. Most hobbyists use a higher level language such as BASIC or assembly language to write programs in which case the octal/hex problem essentially goes away.

Finally there is the issue regarding the labeling of the integrated circuits with their familiar generic designations. Typically we have always used the Heath part number on

EDITOR & SMALL O.S.

Dear Sir:

Enclosed is a description of EDIT, a text editor, and SOS, a Small Operating System.

These programs are designed for 8080 based microprocessor systems. I have noted in letters to the editor of many magazines that the computer hobbyist is looking for such software. Please pass this information along. There are many satisfied users of both software packages in my local area.

EDIT and SOS are available from:

LSM ENGINEERING
P.O. BOX 3243
ORANGE, CA. 92665

EDIT is available on Tarbell block format cassette or hex checksum paper tape for $22.50, and SOS is available on Tarbell block format cassette for $15.00. California residents add 6% state sales tax. Dealer inquiries are invited. Complete documentation is included with each package.

Regards,
Lorin S. Mohler
219 N. Sweetwater St.
Anaheim, CA. 92806

EDIT PRIMER

EDIT is a text editor program designed to be run on an 8080 microprocessor based system.

EDIT can be used to generate or modify pure source files for BASIC, FORTRAN, assembly language or other languages, or plain text that uses the standard 96 character ASCII set. Source files can be manipulated by characters, string, line, and page to perform additions, insertions, replacements, and deletions of text.

Commands may be concantenated to form command strings for execution, or a command string may be defined in a macro that can then be executed repeatedly.

EDIT has provisions for tabbing, selection of console width, and the number of nulls output after a carriage-return line feed.

Any type of terminal may be used for the operator's console, including teletype, CRT terminal, and, with appropriate handlers, video modules.

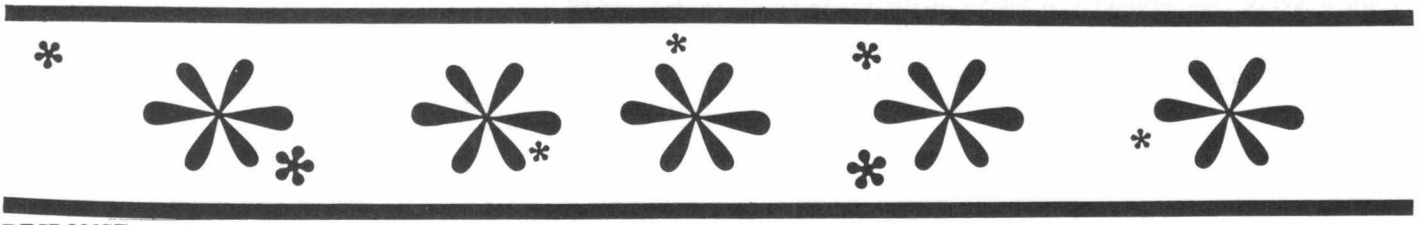
Source files may be entered through the console keyboard, console paper tape reader, or, with properly interfaced input handlers, from any input peripheral device.

Source files may be output to the console, console paper tape punch, or, with properly interfaced output handlers, any output peripheral device.

Tarbell cassette interface handlers and source listing are included. Complete input/output specifications are given for writing the required peripheral handlers for other peripheral devices.

SOS PRIMER

Do you have to hand load a bootstrap routine every time you want to get your system on the air? Or are you like one of those who have disconnected the power switch and let the system run for weeks hoping the power company won't hiccup? Or do you have a minimum boot stored in ROM that just isn't all that could be desired? If so, SOS may be your



RESPONSE, *cont.*

schematics. But the part is easily identified by simply referring to the IC chart in the manual. Your author may be interested to note that most of the H8 diagrams do already incorporate the standard designations. If your author had reviewed actual H8 documentation he would not have made this point.

I am rather surprised at the comments by this author and his specific topics of discussion. Most of the commentaries seem to be opinion or guesses rather than based on actual facts and experience. I doubt that the author would have even written the letter if he had actually constructed an H8 himself and used it extensively. This is the only way that a fair and objective evaluation can be made.

In addition, most of the points are minor ones. While the prospective purchaser of a computer may be concerned with some of the details criticized by the author, I suspect that most individuals are interested in more important things such as: (1) which microprocessor is used, (2) physical appearance, and (3) software. (Your author took a slap at our software in the first paragraph of this article, but never mentioned it further. Could it be that he has never seen or used it?) Documentation and manufacturer support are other important factors that are more important to the user than the minor items criticized by the author. The strength of the H8 lies in the fact that it uses the popular 8080. The computer is attractively styled and easy to build. We believe that the documentation is the best available in the industry. The software package is complete and easy to use. It is also included with the computer at no additional charge.

As for support, Heath also offers the best there is. A customer can purchase by mail order or through one of 50 retail stores across the country. Both factory and retail stores offer service and parts replacement. And don't forget the Heath User's Group. Heath is a large and reputable manufacturer who has made a major commitment to the personal computer business. We plan to support and upgrade the H8 computer for years to come. Many new and exciting peripherals, accessories, and software are on their way. My advice to a prospective computer user is to base his decision on major points such as these rather than the nit picking details blown out of proportion by your author.

The success of the H8 will be determined by the customer in the market place. If our first two months' sales are any indication, the H8 has been well received. Initial results have been simply spectacular. Our H8 customers know that your author is wrong and so will prospective customers who review our documentation or try out an H8 in the store before they buy. We know that everyone will not like or buy the H8. But the H8 certainly does not deserve the unfair criticism of your author.

answer. SOS is a Small Operating System designed to be stored in ROM. It required less than 2K bytes (692 hex) of ROM (allowing room for additional routines in the remaining ROM area on a 2K ROM board) and 256 bytes of RAM for scratch. SOS has commands for saving (CSBK), loading (CLBK), and verifying (CVBK) data blocks (with checksum) via a Tarbell cassette interface, reading binary (RBIN), and Intel hex checksum (RHEX) paper tapes via an Oliver (or similar) optical paper tape reader and system utilities, including a memory FILL, memory block transfer (XFER), memory block verify (VMEM), and memory DUMP and ENTR routines.

SOS is supplied with a commented source lisiting and information on patching SOS without reassembly to add up to four new keyboard commands, modify for terminal displays with less then 64 characters per line, and add special functions.

The SOS package contains an operating instruction manual, suggested patches and information on how to externally call routines within SOS or execute (XEQT) subroutines outside SOS. The SOS object code assembled at B000 hex with scratch at B800 hex is supplied on cassette using a block format with checksum.

Also supplied on the cassette is a line numbered source ready to block load into memory and assemble, wherever you want, with a PTC ALS-8. Information is supplied for the simple modifications required to allow use of the PTC PKG 1 or similar (ESP1, IMSAI) assembler if you don't have an ALS-8.

The following are examples of commands below that might be used in the generation and editing of a text file.

After 'EDIT' has been loaded and started it will display the bootstrap message and prompt (*). The '$' symbol is the echo for the escape key. The '^' symbol indicates where a space is to be entered. The symbol ↵ indicates where the carriage-return is keyed. The prompt (*), shown in the examples to maintain character positioning, is output by EDIT.

Enter the command including two escapes ($$), the remaining portion of the COMMAND/RESPONSE display is EDIT's response.

COMMAND/RESPONSE	EXPLANATION
*:1FFF$$ 3583 *	Limit (:) the text buffer not to expand past memory location 1FFF hex. The decimal number displayed '3583' is the character space available in the text buffer. (Assumes 8K ram available)
*<C0@$>C8@$$ *	Open the cassette peripheral input (<) handler located at C0@ hex and the output (>) handler located at C8@ hex.
*ITHE^LAZY^BROWN^FOX^JUMPS^HIGH^OVER↵ ^THE QUICK DOG ↵ $$ *	INSERT (I) lines of text into the text buffer. The cursor remains after the last character entered (the carriage-return).
*T$$ THE LAZY BROWN FOX JUMPS HIGH OVER THE QUICK DOG *	TYPE (T) the contents of the text buffer. The cursor is not moved.
*BCQUICK$LAZY$$ *	Move the cursor to the BEGINNING (B) of the text buffer, then CHANGE (C) the first occurence of the text string 'QUICK' to the new string 'LAZY'
*-1LCLAZY$QUICK$$ *	Move the cursor to the beginning of the previous LINE (-1L) then CHANGE (C) the first occurence of the text string 'LAZY' to the new text string 'QUICK'. The cursor remains after the last character changed.
*2T$$ BROWN FOX JUMPS HIGH OVER THE LAZY DOG *	TYPE two (2T) lines starting from the current cursor position.
*L1T$$ THE QUICK BROWN FOX JUMPS HIGH OVER *	Move the cursor to the beginning of the LINE (L) and TYPE one (1T) line.
*SHIGH$-5DL1T$$ THE QUICK BROWN FOX JUMPS OVER *	SEARCH (S) for the first occurence of the text string 'HIGH', DELETE the five preceeding (-5D) characters before the cursor, move the cursor to the beginning (L) of the LINE, then TYPE one (1T) line. Note that after a successful search the cursor is positioned after the last character of the string searched for.
*C ↵ $$ *	CHANGE (C) the first occurence of the string 'CARRIAGE-RETURN' to the null string, making two lines into one line.
*L1T$$ THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG *	Move the cursor to the beginning of the LINE (L) and TYPE one (1T) line.
*ZITHE^END ↵ $$ *	Move the cursor to the end (Z) of the buffer and INSERT (I) a new text string.
*XMC ↵ $. ↵ $$ *	Define a Macro (XM) to CHANGE (C) the string 'CARRIAGE RETURN' to the string 'PERIOD CARRIAGE-RETURN'
*B5X$$ STRING NOT FOUND C S. $$ *	Move the cursor to the BEGINNING (B) of the text buffer then attempt to execute the Macro command five (5X) times. Since the string 'CARRIAGE-RETURN' only exists twice in the text buffer, the "STRING NOT FOUND" message and the remaining unexecuted portion of the command string will appear. The cursor will move to the beginning of the text buffer due to the error.
*T$$ THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG. THE END. *	TYPE (T) the contents of the text buffer
	Start the cassette drive in the record mode.
*1P$$ *	PUT one (1P) line of the text buffer from the current location of the cursor (now at the beginning of the buffer) to the output peripheral previously opened (cassette).
*T$$ THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG. THE END. *	TYPE (T) the contents of the text buffer. Note that the text buffer is not disturbed by the PUT command.
*B1KT$$ THE END. *	Move the cursor to the BEGINNING (B) of the buffer, KILL one (1K) line starting from the location of the cursor, then TYPE (T) the contents of the buffer.
*PE$$ *	PUT (PE), with an End of file 'EOF', the contents of the buffer to the peripheral previously opened.
*B1@KT$$ *	Move the cursor to the BEGINNING (B) of the text buffer, attempt to KILL ten (1@K) lines, then TYPE (T) the contents of the text buffer. Note that the buffer is empty.
	Rewind the cassette tape then start the drive in the playback mode.
*YA$$ *	YANK (Y) one page, then APPEND (A) another page from the peripheral device previously opened.
EOT *	YANK first clears the text buffer then reads in a page of text. The APPEND command adds text to the end of the current contents of the text buffer. The 'EOF' message indicates that the end of file was encountered, requiring a peripheral reset before another YANK or APPEND may be performed. Stop the cassette drive.
*BT=$$ THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG. THE END. 2/54 *	Move the cursor to the BEGINNING (B) of the text buffer, TYPE (T) the contents of the buffer then display the number of lines and characters (=) in the text buffer.

TINY BASIC & DIGITAL GROUP DISPLAY ON A HONEYWELL 112

Dear Jim, *Dated: 77 Sep 11*

In the May issue of *Dr. Dobb's Journal* you published my letter saying I was interested in putting Tiny BASIC onto my Honeywell 112 computer. So I thought you might like to know how the project turned out. I finished building the Digital Group (TVC-F) display and it worked first time, and so that allowed me to begin writing Tiny BASIC in my assembler language.

The complete interpreter is now up and running; it has a few extra features making it an extended version of Tiny BASIC. My implementation allows subscripted variables, single dimensional matrix operations and ASCII string handling plus special screen control and positioning commands.

The interpreter takes approximately 3K of memory and I have left 1K for future expansion, the other 4K of my machine holds the basic program and as my machine has 12 bit words I hold 2 six bit characters per word making it equivalent to 8K bytes in a normal micro.

In the memory the BASIC program is held in the following format:
1 loc for the line number (max 4095)
1 loc for the BASIC command (form LT for let, PR for Print, etc.)
extra locations for the rest of the statement as required ending with a carriage return character.

Space characters are not stored unless it is a character string which is being entered to memory (which saves a lot of space), space characters are automatically reinserted into the listing by the interpreter. Multiple statements per line are ok. As interest in the Honeywell 112 is probably rather restricted I don't think you would be interested in the Interpreter listing; however, if there is anyone in your wide readership who has a 112 and would like a high level language for it, I would be happy to supply help if possible.

Thanks as always, Jim, for an excellent publication and good luck for the future.

Yours faithfully,
Ian D. Spencer
Am-Museum 1A
5353 Mechernich
West Germany

POLY DOC Corrections

By Richard B. Emerson

Dear Mr. Warren, *Received: 77 Sep 7*

It was a pleasure to have met you at Atlantic City. I'm sure you remember me, I was only the thousandth guy you met with a Byte bag, long hair, and tired feet. Seriously, we had talked about Jef Raskin's piece about PolyMorphic and their documentation (or lack thereof). To recap, the assembly section of the system manual is not too well organized and has too many inaccuracies. I also mentioned that I had called PolyMorphic and had less than total satisfaction with a Mr. Tom Jensen. I had called to complain about missing parts (diodes for the power supply), verify the latest documentation for BASIC, and inquire about using IMSAI's 4k RAM.

As I indicated to you, I have put together a set of notes about the errors that I encountered. I'm sure that I have overlooked some problems and that advanced hardware types will find some of my remarks a bit tedious, but I hope that hardware neophytes will find things of value. All remarks and procedures have been *successfully* checked out, but as I said, I may have overlooked some minor "gotchas". I have not attempted to really verify the software section, but aside from odd organization, it seems to be generally in better shape.

You suggested that I speak to the senior representative at the PolyMorphic booth, but I didn't have any luck there. Things were rather (!) busy, so I guess I can't complain too much. Needless to say, I am sending a copy of this set of notes along with the letter attached to the notes.

I'll be happy to help other POLY 88 people on a time permitting basis. To that end in addition to my address I've included my phone number. Naturally you have my permission to publish any or all of the enclosed.

Thanks for your attention and keep up the great work.

Sincerely Yours,

Richard B. Emerson
158 E. Butler Avenue
Ambler, PA. 19002

COPY OF A LETTER DIRECTED TO POLYMORPHIC SYSTEMS

Dear Sirs: *Received: 77 Sep 2*

I recently purchased a POLY 88 system from Personal Computer Corp. in Frazer, Pa. The system consists of a System 2 with IMS's 8k RAM and IMSAI's 4k RAM. The peripherals are a converted JVC receiver, a Superscope C-103, and a Claire-pendar keyboard. At this time the system is running satisfactorily, but it has been a difficult fight to get to this point. Problems can always be expected when attempting to bring any computer up for the first time. As a systems analyst for a large university, it is a fact that is often called to my attention. The problems that I encountered, however, were avoidable for the most part had more consideration been given to the manual shipped with the system. In addition to the eros and confusion in the documentation, there were missing diodes and sockets.

The carton I received from Personal Computer was open, but all of the sub-packages were present. They had to get tapes for the assembler and BASIC withBPRINT from "the back room". In fact, they did not have any copies of the assembler on hand and I had to make a copy of the assembler. I received summaries of the assembler and BASIC commands, but a second trip was required to get a copy of "BASIC: A MANUAL". On opening the individual packages, I found that the two 60S05 diodes had not been packed, nor had two 14 pin sockets for the cassette card. The sockets were easy to get around but the 60S05's were not so easily replaced. There were none in stock at Personal Computer and, in order to test the power supply, instead of the 6 amp units, I installed 4 amp units. I have since been given diodes "that should work". A call to your plant and a Tom Jensen resulted in the suggestion that I get the diodes from my retailer. If I could have gotten the diodes from Personal Computer , I would not have called.

The difficulties with the documentation were, and continue to be, a far greater source of aggravation. The problem easiest to describe is the lack of organization. The cassette assembly instructions contain introductory remarks that are vital to the assembly of any component of the system. My unfamiliarity with tantalum capacitors may have resulted in damaged regulators; had I seen the note on polarity markings that might not have happened. The notes on handling CMOS chips are of considerable importance as well. Checkout procedures require the use of the Front Panel Mode, but there is no quick summary of how to input programs or data. That summary is hidden in Volume 2 and no reference is made to that list. Frankly I could go for pages (an entire manual's worth!) about how the manual could be rewritten, but instead I suggest you ask a "typical" customer to build a system and note where problems occur. Heathkit has used this strategy with a great deal of success and I am sure that after-sales costs have been reduced as well.

The specific technical errors, procedures, and component descriptions are easier to summarize. To that end I have prepared a page by page list of suggested corrections. Since I have not built the serial or printer output card, I have not written any comments. I did build the cassette interface and corrections include a complete rewrite of the alignment procedure. I have found that, in spite of the errors in the manual, the system is in fact better than advertised. The BASIC manual, once I received a copy, is an improvement over the systems manual, but a more complete description of how the PLOT instruction works would be welcome. Tips on how to protect various areas of the screen as occurs in the Front Panel Mode would also be welcome. With that in mind, I think a simple newsletter would build a more cohesive group of users. There are significant differences in the POLY88 implementation of the S-100 bus and publishing notes on how to get around some of the difficulties that this produces would certainly get both thanks and increased sales of main frames.

As I said at the outset, the system is up and running to my satisfaction. A complete review of the manual and more quality control would have save me and, I'm sure, others a lot of headaches. I look forward to hearing your suggestions and comments on the enclosed notes.

Sincerely Yours,

Richard B. Emerson
158 E. Butler Avenue
Ambler, PA. 19002

cc: Personal Computer Corp.

Notes on building a POLY88 System 2 – August 1977

CPU Assembly

P. 2 - The 2 high current diodes 60S05 are not easily replaced. Mine were missing and I used 4 amp diodes to check out the power supply ONLY!
You will need heat sink grease for the voltage regulators that use sinks. The 79L12 does not need a heat sink.

P. 8 - Double-check the positioning of the 1N4148 diode. Its position is not marked on the back plane. Consult the cassette interface notes for details on diode markings.
Step 5. should refer to D3 not De.

Fig A-1 R8 is only a jumper, not a resistor.

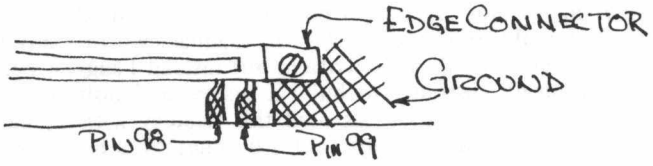
Fig A-2 The 6-32 x .38 screws are too long. They will snag capacitors and other parts on the edge of cards.

P.10 - Do *not* cut the black lead to 5". There will be no slack in this lead when you mount the transformer. 6½" should be long enough.
The three leads that are twisted together will carry 110 VAC and are being twisted to keep hum down and minimize their rolling around. Don't get over-enthusiastic about the twisting.
Step 14 suggests soldering the line cord at the rivets. It is an excellent idea as you will soon see.

P.11 - It is difficult if not downright impossible to mount the leads without bolting the transformer in place.

P.12 - The cabinet is *not* finished! There are video jacks, sockets and various other bits of hardware that will need to be mounted.

P.13 - Ground Pad Location Sketch:



P.14 - Record the supply voltages (and the line voltage, if possible).This will help when it comes time to service the machine. If the unloaded supply voltages are off, it is possible that the power line voltage may have changed ("brownouts", new line loads such as air conditioners, etc).
- Be sure to use liberal amounts of heat sink grease when installing both the regulators and heat sinks.

P.15 - Check the polarity of the tantalum capacitors before installing them. There will be a plus or a stripe on the positive lead side of the capacitor.
- IC43 (78M05) should read IC*34* (78M05)
- Most sockets have some sort of mark to indicate pin 1, be sure to align the sockets accordingly as some outlines will be covered over.
- Record all regulated *and* unregulated voltages. Don't forget the line voltage!

P.17 - C1's position on the CPU card is incorrectly marked. C1 should run from the *THICK* trace containing the upper half of C1's outline to the next *THICK* trace directly below the outline.
DO NOT USE THE HOLE RESERVED FOR R12!
- C25, C30, and C43 will be in the serial port components bag, bag #4.

P.18 - D3 (1N4148) will be in bag #4 (serial port components).
- Record all voltages as before and see how much change there is, or isn't, from previous readings.

P.19 - Verify pin 1's location before installing chips, an error here will be very costly!
- The DIP switch should be oriented so that toggle 1 is topmost and toggle 7 is closest to the fingers of the edge connector.

P.20 - R14 is the current limiter for the zener diode (D2) used to provide -12 volts to your keyboard.

* * *

If you are using a 1N4742 zener diode rated for 1 watt, you may compute the value for R14 as follows: Look up the last voltage you wrote down for the unregulated -18 to -22 volt supply. Subtract 12 volts from that value; ignoring the sign, you should get about 6 to 10 volts. You now have the desired voltage drop, but you must adjust for the voltage drop across D2 so add .7. Having figured the total voltage drop, you now must figure the current draw. The specifications for your keyboard should say how much current is need at -12 volts. (A Silent 700 keyboard made by Claire-Pendar draws 10 ma. or .01 amps, for example) To insure good regulation, a 1N4742 diode needs at least 1 ma, but to include a safety margin, we'll assume that there should be at 20 ma flowing (or .02 amps).
The desired value for R14 = Vdrop/Itotal. In this example, assume that we have a -20 volt supply we want to drop to -12 volts at 10 ma. R14 = (20-12)+.7 / (.02+.01) = 290 ohms. Select the resistor nearest this value and make sure that it is rated at ½ watt. Do not operate the diode without its load for extended periods as the power through R14 will be about .6 watts!
- R17 should be 2200 ohms (RED-RED-RED).
- R25 should be 3300 ohms(ORANGE-ORANGE-RED).

P.22 - 8097's and 74367's are interchangeable.
- IC16 should be a 74393.

P.23 - IC34 may also be a 74LS123.

P.25 - To select F800h, toggles 1 and 2 should be off, the rest (3 -7) should be on.

P.26 - 4.4.1 see the errata notes for converting to a DB-25 S connector.

P.28 - The plus or minus on JMP 7 indicates the state of the strobe line when the strobe signal is absent. + means that the VOM showed a normally high state.

P.29 - Perform 4.0 monitor mods as outlined in volume 2, pp 74-75.
- Polarity for tantalum capacitors is indicated by a + or stripe near the positive lead.
- IC's may be checked by prying up one end of the chip and looking underneath for bent pins. Make sure that no pins are bent when you replace chips.
- Check to see that all of the appropriate errata procedures have been completed.
- Phase 2 of the clock is brought from pin 6 of IC13 to pin 49 of the bus. There is no pin 49 on IC13!

P.32 - Unsolder the 100 ohm resistor R19 and be sure that your D.C. blocking capacitor(s) don't eliminate any biasing you wanted to send to your video monitor.

P.43 - 33uf is, in fact correct. Ignore the gap between the 33 and the u.

Cassette Interface Assembly

P. 7 - The file name for the small dumper is SMD. The cassette will be marked with this name.

P.13 - A DB-25 P must be made up to mate with the DB-25 S connector on the interface. After completing the voltage checks and writing down the results, wire up three audio cables. Refer to the sketches in appendix A for the proper pinouts. The only difference between the sketches is in how the motor control is wired. It is not essential that this wiring be completed in order to test the cassette interface. Do not connect the DB-25 P until all alignment procedures have beem completed.

P.15 - After setting the address of the cassette interface, do NOT install the board in the cabinet. Because at least

one trace will have to be cut, it is best that the card be rested on the top of the transformer. Naturally it is crucial that there is some sort of insulation to keep the card from shorting out!

* * *

New alignment procedures:
While the procedures outlined originally are basically correct, a few significant details have been omitted. For alignment of the Byte/Polyphase cassette card you will need a logic probe (or 'scope), a voltmeter, and a tantalum capacitor rated somewhere between 1 and 10 uf at 16VDC. Once the alignment is complete the capacitor will not be need so that, with a bit of care, capacitors from a memory kit might supply a part for the alignment.
Make sure that JMP 1 is shorted by a trace on the bottom of the card. Connect the capacitor's plus or striped lead to pin 1 of IC6 (4027). The other end of the "cap" should go to pin 5 of IC1 (8T20). Be sure to take extra care with the CMOS chip IC6. After checking to see that there are no shorts from installing the capacitor and that the card is properly insulated from the transformer, install the CPU and VTI cards. Connect the free end of the cassette interface cable to port 2 of the CPU card (port 2 is the rightmost socket in the upper right corner) making sure pin 1 is in the lower left corner of the socket. Power up the POLY88, push the reset button on the front of the cabinet, and type a B. Measuring between TP1 and TPG (ground), you should see 2 to 5 volts and, of course, your logic probe should show a continuous high state. At TP1 we want to get a pulse train with 75% duty cycle. This means that if you are using a VOM you must multiply your first measurement by 0.75. Shut down the POLY88 and tie pin 9 of IC5 (74LS86) to TPG (ground). The pad marked TPG near pin 14 of IC5 is recommended, but either one will do.
Double check your work and then power up the POLY88. Adjust R27 (the variable resistor or "pot" on the far side of Q1 (2N5447) from the 25 pin connector) until you get a value equal to 75% of your first measurement between TP1 and TPG. A 'scope should show a 75% duty cycle. A logic probe should show a low state with positive pulses at TP3. If you see nothing after adding the jumper to ground, check to see that the jumper is installed correctly and make sure that you typed B to activate the card.
With the power off, remove the jumper from pin 9 of IC5 to ground. Cut the trace on the underside of JMP1, removing a ground to R21 (2200 ohms), R4 (2200 ohms), and pin 1 of IC4 (74LS257). Disconnect the capacitor you have attached to pin 5 of IC1 (8T20) and remove IC1. With IC1 removed and the loose lead from the cap clear of other components, turn on the power. Measure the voltage between TP3 and TPG. It should be 2 to 5 volts (logical high). Multiply the voltage by 0.25 and record both values. Turn off the power and replace IC1. Reconnect the temporary capacitor to pin 5 of IC1.
Power up the POLY88 and enter the program on p.16. If you are uncertain about how to proceed, read volume 2, pp. 59-66.
Execute the program beginning at 0D00h. Set R26 (the pot closest to the 26 pin connector) fully counter clockwise and measure the voltage at TP3. Slowly rotate the pot clockwise until you see the voltage drop to 25% of the first measurement at TP3. There will be two positions that will give this voltage, but it is the first position (the one closest to the full counterclockwise stop) that counts. You are looking for a 25% duty cycle at TP3 or a low with positive pulses. TP2 should show alternating 1's and 0's. If you have gotten the results indicated above, the Byte and Polyphase procedures are complete and you should remove the capacitor after turning off the power. It is possible that some additional "fine turning" may be needed to match the interface to your recorder. It is helpful to mark the initial settings on the pots before attempting fine turning.
Before mounting the cassette interface in the cabinet, you might consider replacing JMP2 and JMP3 with jumper that are accessable from the top of the card. Installing Molex connectors and wire jumpers will mean easier changes without removing the card. Be sure to cut the jumper traces on the under side of the card if you install the wire jumpers. JMP1 may be replaced with a single pole single throw switch. There is a hole near the video connector that may be used to hold the switch.

General Remarks

It cannot be repeated often enough, WHEN IN DOUBT, ASK!! Your POLY88 is an excellent piece of equipment and represents a fair-sized investment. A hasty action during construction can reduce chips to silicon ashes. Take the time to double check things BEFORE things get out of hand.
A full cabinet MUST have good ventilation and convective cooling is not sufficient. PolyMorphic makes a fan kit that may be retrofitted when you fill up the slots.
When installing non-PolyMorphic cards, check to see if there are any signals required that are not on the bus. The appendix of the CPY/VTI assembly instructions contains a summary of differences. In most cases only a jumper or two will be needed, but if there is any doubt, ASK!!

* * *

Prepared by Richard B. Emerson. The contents of these notes are the results of personal experience. No other claims are made or implied. Copyright (C) AUGUST 1977 – All Rights Reserved.

People who play with bytes should expect to get bitten (with a tip of the hat to Bill Bierley) – Dennis Allison

SIERRA COMPUTING ANYONE?

Received: 77-Oct-18

Can you program a working Tic Tac Toe game in an hour without any brainstrain?

Are you looking for a way to make a living programming games, systems, and exploring the strange wonders of software?

Are you burned out on the big city?

We are a small engineering group inventing games and toys. We live in Grass Valley, California in the Sierra foothills, an hour from skiing and two hours from Chinese food in Oakland.

If this sounds like your ticket, send us your Tic Tac Toe program and a resume, to:

Cyan Engineering
P O Box 1875
Grass Valley, CA 95945
(916) 273-6194

PILOT/CPM BUGITO

Gentlemen: *Received:77-Sep-26*

I have, I think, uncovered a bug in Pilot.

I interfaced Pilot to IMSAI's CPM disk system to read programs off disk. The CPM ED puts a line feed (0AH) at the end of each line. The program checks for this and sets up a character skip to take care of it. However, no account is made for the 1st line of the program or for statements following Labels.

I fixed this by having SCAN check for TEMP "ON". I set TEMP "ON" in IEP for the first line. LOOKL sets in on for the labeled lines.

There is probably a better way of doing this. I would appreciate any comments.

Maurice Katzer
5740 Gordon Lane
Ft Smith, AZ 72903

P.A.T.B. PATCH
MEMORY POSITION 0011 HEX
WAS 3A 00 08 CHANGE TO CD BB 07

Dear Dr. Dobbs *Received: 77 Jun 8*

Enclosed is a simple video driver for Palo Alto Tiny Basic. I am using it with a Polymorphic VTI board, however it should work with most other video boards. Depending on the particular board, line 0210 may have to be changed to ANI 7F (E6 7F) and line 1410 changed to MVI A,20H (3E 20).

The software switch (Control O) will turn the teletype off but leave the video on.

The driver fits in after Russ Hayden's strings, PEEK-POKE and CALL to subroutine extensions (*Dobb's* Vol. 2, #2) and doesn't go over 2K.

Robert Connolly P.O. Box 27087
Tempe, Arizona 85282

```
1900                    0010 SCRN EQU 0CC00H  SCREEN ADDRESS
1900                    0020 SSIZ EQU 1024  SCREEN SIZE
1900                    0030 LNLN EQU 64  LINE LENGTH
1900                    0040 END EQU 0D0H
1900                    0050 CURS EQU 64FH STORAGE FOR CURSER
1900                    0060  ORG 7BBH
07BB E5                 0070 VOUT PUSH H
07BC CD C4 07           0080  CALL SCOUT
07BF E1                 0090  POP H
07C0 3A 00 08           0095  LDA 800H REPLACE PATCHED AREA
07C3 C9                 0100  RET BACK TO PATB OUTPUT
07C4 FE 0D              0110 SCOUT CPI 13 CARRIAGE RETURN?
07C6 CA DD 07           0120  JZ SCROL
07C9 FE 20              0130  CPI 20H
07CB D8                 0190  RC
07CC 2A 4F 06           0200  LHLD CURS
07CF F6 80              0210  ORI 80H
07D1 77                 0220  MOV M,A
07D2 23                 0230  INX H
07D3 7C                 0240  MOV A,H
07D4 FE D0              0250  CPI END
07D6 CA DD 07           0260  JZ SCROL  SCROLL IF FULL LINE
07D9 22 4F 06           0270  SHLD CURS
07DC C9                 0290  RET
07DD                    0300 *MOVES EVERYTHING UP ONE LINE
07DD C5                 0310 SCROL PUSH B
07DE 01 00 CC           0320  LXI B,SCRN
07E1 21 40 CC           0330  LXI H,SCRN+LNLN
07E4 7E                 0340 XX MOV A,M
07E5 02                 0350  STAX B
07E6 03                 0360  INX B
07E7 23                 0370  INX H
07E8 7C                 0380  MOV A,H
07E9 FE D0              0390  CPI END
07EB C2 E4 07           0400  JNZ XX
07EE                    0409 *FILLS BOTTOM LINE WITH BLANKS
07EE 3E 3F              0410 YY MVI A,3FH
07F0 02                 0420  STAX B
07F1 03                 0430  INX B
07F2 78                 0440  MOV A,B
07F3 FE D0              0450  CPI END
07F5 C2 EE 07           0460  JNZ YY
07F8 C1                 0470  POP B
07F9                    0479 *HOMES CURSER
07F9 21 C0 CF           0480  LXI H,SCRN+SSIZ-LNLN
07FC 22 4F 06           0490  SHLD CURS
07FF C9                 0510  RET
```

BACKSPACER FOR DR SUDING'S 16 x 32 TVT

Dear Mr Warren: Sept 26, 1977

I have enjoyed your magazine for some months now, but it is only of late that I have gotten my system far enough along to be able to implement some of your published programs. I don't know if the enclosed short subroutine may be of interest to your readers; if not, just file in the round file. After wirewrapping Dr. Suding's TV readout device as it appeared in the August, 1976 issue of Byte magazine, I had to develop software to run it. The enclosed backspace subroutine is somewhat unconventional; it forces the video card to run backwards and replaces the character with a blank.

Bytes 021 and 034 can be any number as long as the MSB equals zero. Bytes 011, 012, and 013 are necessary to my system in order to clear the parallel keyboard latch.

Don Woods
12012 Pebblebrook Lane
Carmel, Indiana 46032

```
XXX  000   315 015 XXX   CALL BACK SPACE SUBROUTINE
     003   315 046 XXX   CALL BLANK OUT
     006   315 015 XXX   CALL BACK SPACE SUBROUTINE
     011   *** *** ***   (clears keyboard latch)
     014   311           RETURN
     015   305           PUSH B - START OF SUBROUTINE
     016   006 000       MVI B COUNT BYTE
     020   076 001       MVI A
     022   315 050 XXX   CALL TVT OUT
     025   005           DCR B
     026   302 020 XXX   JNZ  JUMP BACK IF NOT ZERO
     031   006 377       MVI B  COUNT BYTE
     033   076 001       MVI A
     035   315 050 XXX   CALL TVT OUT
     040   005           DCR B
     041   302 033 XXX   JNZ  JUMP BACK IF NOT ZERO
     044   301           POP B
     045   311           RETURN
     046   076 240       MVI A   LOAD BLANK
     050   323 XXX       OUT TO TVT
     052   257           XRA A  TURN OF TVT STROBE
     053   323 XXX       OUT TO TVT
     055   311           RETURN
```

Dijkstra's minilanguage

By Neils Laughlin

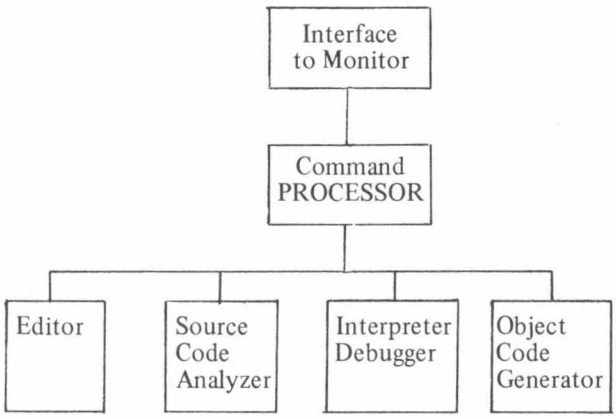
Dear Dr. Dobbs: *Received: 77 Jun 6*

I am in the process of writing a compiler/interpreter for a version of Dijkstra's so-called minilanguage he introduced in his book A Discipline of Programming *in 1976.*

I would like to hear from any hobbyist who is working along similar lines. Eventually, I hope to have my IMSAI 8080 programmed exclusively with this language, eliminating all assembler code, even from the monitor and I/O drivers. The compiler will be written in the language itself and should be ready by January 1978. I should point out that I do not yet have my hardware up and running - - I wish I bought it assembled.

Included is a description of Dijkstra's minilanguage as I will use it, along with a recoding of "HI-LO", a game program which appeared in April 1977 issue of Kilobaud.

Neils Laughlin
136 S. Virgil Ave. #126
Los Angeles, CA. 90004
(213) 389-2714



I. Minimum features.

Since the compiler/interpreter is to be written in the language itself, it must have the features that will support the effort. This means indirect recursion, arrays, subroutine calls, and integer and character data types. Procedures will not be implemented.

II. General Structure of the Compiler/Interpreter/Editor.

As you can see from Figure I, the system wil consist of an editor, source code analyzer, interpreter/debugger, and object code generator. One of the advantages of BASIC is the integrated editor and interpreter. That is not, however, an excuse to intertwingle the code for each function. Most interpreters (at least the faster ones) create and intermediate form of the source code, usually for a hypothetical stack machine. This is the job of the source code analyzer. The object code generator will produce position independent code. Eventually, I will write my own editor in the language. For now, I intend to patch an existing editor.

III. I/O Statements.

I have not finalized the design on I/O statements. Dijkstra's presentation had no I/O. I am considering two possibilities:

1. Use BASIC type I/O statements. The sample program uses this, with an extra parenthesis.

2. Use PASCAL type I/O statements. This is attractive because of the greater data structuring capabilities.

IV. Data Structuring.

I intend to use PASCAL as a model. Of course, the first version will have no user types, only integers, characters and arrays. Even there, I intend to depart somewhat. Consider,

```
VAR     NAME      :CHAR X(15);
        ADDRESS   :CHAR X(25);
        AGE       :INTEGER;
        SALARY    : BCD 9(7)9(2);
```

Here, I added a little COBOL, providing "pictures" for CHAR (character) and BCD (decimal) data types. The initial version, however, will not have BCD.

Once I have the system up and running, I will add more PASCAL type data structures, starting with SET and RECORD.

V. Statements (refer to Dijkstra's excellent book, "A Discipline of Programming").

1. *The guarded statement and the IF statement.* Consider,

```
IF X < 0 -->  PRINT("X IS TOO HIGH") #
   X = 0 -->  PRINT("X IS JUST RIGHT") #
   X < 0 -->  PRINT("X IS TOO LOW")
FI
```

The above IF statement is made up of three guarded statements, separated by the symbol (#). One of them is,

```
X > 0 -->        PRINT("X IS TOO HIGH")
```

The three guards (which can be any logical expression) are X < 0, X = 0, and X > 0. The right arrow (-->) separates the guard from its statement list which can be other IF or DO statements.

During runtime, one guarded statement whose guard is true is executed, and then control passes to the statement following FI. If no guards are true, the job is aborted. The statement IF FI is equivalent to ABORT.

2. *DO statements.*

The DO statement differs from the IF statement in the way the guarded statements are controlled. Both are lists of guarded statements, bracketed by IF FI in one case and DO OD in the other. Consider,

```
IF A > 0 AND B > 0 -->   X VIR:=A;
                         Y VIR:=B;
                         DO X > Y --> X:=X-Y #
                            Y > X --> Y:=Y-X
                         OD
                         PRINT("THE GCD OF ";A;"
                         AND ";B;" IS ";x) FI
```

The job is aborted if both A and B are not greater than zero. The guards for DO are X > Y and Y > X. The DO statement cycles through its guards, selecting a guarded statement whose guard is true for execution. When all guards are finally false (X=Y), and this holds for the initial condition as well, control passes to the statement following OD. The statement DO OD is equivalent to SKIP.

NOTE 1: The above is Euclid's Algorithm.

NOTE 2: For both IF and DO, the reader may note the similarity of a list of guarded statements to decision tables.

NOTE 3: IF and DO provide *all* of the control structures needed - - simplicity is the key. I have tentatively named this language PL/MINUS ONE.

NOTE 4: X VIR:=A. The symbol (VIR:=) is the same as (:=) except that it highlights initialization.

VI. The following is a recoding of the game program "HI-LO" which was published in BASIC in the April 1977 issue of *Kilobaud.* This will give everyone a chance to evaluate the language with a larger program example. The function RND (0,100) returns an integer random number between 0 and 100.

CHASING 'CHASE' BUG

Sirs: *Dated: 77-Aug-20*

I have discovered a bug in the assembler language version of "Chase", written by Joseph J. Sanger and published in the May 1977 issue of *DDJ*.

The sympton was that after a few games, the program would crash and display garbage. After first suspecting intermittent hardware, I found the problem.

Subroutine 'DEATH', invoked whenever you are touched by a robot, post or fence, jumped back to 'START' for another round after informing you of your demise. This left the stack pointer unrestored to its previous place. Eventually, the stack "grew down" over the end of the program, causing it to crash.

Here is a simple fix for this bug:

```
LOC   OBJ CODE     SOURCE STATEMENT

029A               1      ORG  029AH
029A  C3  F4  05   2      JMP  FIX
05F4               3      ORG  05F4H    ;end of prog
05F4  31  50  06   4  FIX: LXI  SP,STACK ;fix stack
05F7  C3  2C  00   5      JMP  START    ;back to nxt game
05FA               6      END
```

If you have an assembler, a "cleaner" fix would be to insert a LXI SP, STACK instruction at 'START' in the source code.

Phil R Karn
WA3RVL/WB2AJX

continued from page 19

```
PROGRAM HILO;
VAR NOGUESS, GUESS, RNDNO, GAMESTATUS: IN-
   TEGER;
BEGIN
GAMESTATUS VIR:=1;
DO GAMESTATUS = 1 -->
   RNDNO VIR:= RND(0,100);
   NOGUESS VIR:=0;
   GUESS VIR:= -1;
   DO GUESS <> RNDNO -->
      PRINT("GUESS");
      INPUT(GUESS);
      NOGUESS := NOGUESS+1;
      IF GUESS > RNDNO --> PRINT("WRONG, TOO
                           LOW") #
         GUESS = RNDNO --> PRINT(GUESS;" IS COR-
                           RECT AFTER ONLY ";);
                           PRINT(NOGUESS;" GUESS-
                           ES") #
         GUESS > RNDNO --> PRINT("WRONG, TOO
                           HIGH")
      FI
   OD
   PRINT("DO YOU WANT TO PLAY AGAIN (1=YES,
      2=NO)");
   INPUT(GAMESTATUS);
   DO GAMESTATUS <> 1 AND GAMESTATUS <> 2 -- >
      PRINT("FOLLOW INSTRUCTIONS, RETRY");
      INPUT(GAMESTATUS);
   OD
OD
END.
```

THE SECRET SOFTWARE SYNDROME

Jim Day Received: 77 Sep 27

Professional programmers who sell software to computer hobbyists seem to complain alot about being ripped off by people who use their programs without paying for them. While I have no way of knowing how valid their complaints are (will all those using stolen software please raise their hands?) there can be no doubt that software is easy to copy and hard to trace.

I believe that most hobbyists are honest and wouldn't think of using stolen software. Nevertheless, I consider the hobby market a rather risky one for software vendors. Ripoffs aren't the main hazard however; the real zinger is the freebie. There are alot of very bright hackers out there in hobbyland. Some of them are software pros and some are talented amateurs. They're busily writing all kinds of software for their own machines: monitors, assemblers, compilers, interpreters, etc. Much of their work is first rate. Best of all, it's free. Hobbyists are more than willing to share their programs. How can commercial vendors compete with that?

When faced with the choice of paying $100 or more for a piece of commercial software *without* source code or a freebie *with* source code, most people will at least try the freebie first. If it doesn't work out, they can always go for the commercial product.

Personally, I wouldn't want to take a piece of sourceless software as a free gift, let alone pay money for it. Sooner or later, I always want to modify any software I use. Without the source code this can be a real hassle. (In fact, it can be a hassle even with the source code and full documentation.)

I wouldn't say that software vendors can't make money in the hobby market. Far from it. But I do think they're going to have to realize that the ground rules are different from the usual commercial practice. I believe their best bet would be to price their product so low that no one would be tempted to rip it off. Another approach would be to sell their software to the hardware manufacturers or computer stores who, in turn, would give it away or resell it for five or ten dollars.

By refusing to sell source code, vendors are simply cutting down their own sales potential. This practice can't prevent anyone from making a tape copy for his buddy down the street, nor can it keep anyone from using a disassembler to see what makes a program tick. Yet some vendors are so secretive they won't even sell you a Basic manual unless you buy the whole package and sign a non-disclosure agreement. Who needs that kind of trip?

Editor's Note: It is my impression that a great many hobbyists rip off for-sale software, often with vengeful glee . . . when it is high priced (whatever that means–say, over $25). It is also my impression that very few rip off low-priced software.

I suspect a similar situation would hold with stereo music tape cassettes. If the recording manufacturers priced a 22-minute Beatles (is my age showing?) tape at $100 or more, no doubt few would be sold but many would be copied. As it is, many are sold–enough to provide a fair if not handsome profit for artist ("programmer") and recorder–and some are copied.

The problem of providing source code has some interesting aberrations. Let us say that the software artists sell their creations to hardware manufacturers, as suggested–thereby making a fair and equitable income for their talent and labor. Further, let us say that the hardware maker then gives this software away as an inducement to buy his system. Presumably the software has been configured to match his hardware. If so, if the sources become available, it then becomes a simple matter to reconfigure the software to run on competitors' machines using the same microprocessor, thereby ripping off the market advantage obtained at significant expense by the original hardware maker. What then? –JW

GRAFX: A Graphics Monitor for the 8080/TV Dazzler

BY JONATHAN W. MILLS

For any readers of HP-65 Notes out there, hi! And to all the rest of you users of what we call big machines, greetings. I am Jonathan Mills, the junior half of K&J, editors of "The Recreational Programmer" (unabashed plug for the Notes), and now an owner of an IMSAI 8080 which bears the same relation to my HP-65 as the 65 does to my ancient SR-10.

Since I am visually oriented, I had great hopes of using the IMSAI for simulations that would do more than print out "THE TEMPERATURE AT WALLA WALLA IS 10 DEGREES CELCIUS. RAIN IS EXPECTED by 4 PM." No indeed! I wanted the whole weather map! Well, this entailed an investment in a bit-mapping graphics device of some sort. For reasons of dependability and ease of construction, I acquired a Cromemco TV Dazzler. This kit is easily built, although someone had removed the operating manual while it was waiting to be sold. No problem. A call to Cromemco, and I had it within three days! Super people, and efficient.

A quick look soon showed that unless all I wanted to display was a "crazy quilt"of bits in their original sin (preferred states, I understand they're called) I was going to have to write software. Or buy a joystick, analog-to-digital interface, and software. Say, $200, not to mention a paper tape reader. Ouch.

GRAFX is my answer to that software. Simply put, it is an interactive program that redefines the numeric (and certain other) keys of your terminal/keyboard so that they control a cursor on the Dazzler display. The version of GRAFX that I present here is designed solely to let you enter pictures, diagrams, plots, etc., into memory MANUALLY. I also have a version running that uses subroutines for the cursor control functions, but while trying to keep the interactive features of the program by using programmed changes to executable code, I rewrote a floppy disk, and about 15 (sob!) other programs. I'll keep this tiger for the time being, rather than unleashing it amongst the dragons.

So, Tom, here is the article . . . and now let's look at GRAFX!

Jonathan W. Mills
P.O. Box 2571
Kalamazoo, MI 49003

This program provides an easy way to write graphics into a 2K block of RAM addressed by the Cromemco TV Dazzler. This program accomodates the complex Dazzler interpretation of memory so that the user sees it as 128 horizontal lines, each 128 contiguous bits wide, instead of four quadrants with each byte's bits arranged in a "ZZ" pattern. A detailed explanation of this transform operation is provided in the theory of operation.

GRAFX puts the Dazzler into the 128H by 128V ("resolution X 4") mode, and black-and-white display. Other modes can be chosen, but consideration must be given to resulting "oddball"behavior, such as losing the cursor offscreen when using only 512 bytes of memory for the display, or failure of the cursor of "move"in the normal (64H by 64V) mode. In this case, the current location will go through several color changes as the cursor steps through the memory nybble and will move to the "next" location only after entering the next "nybble".

The monitor is designed to use a standard numeric keypad (0-9 and . and ,) for cursor control. The cursor can be MOVEd, SET, (current bit turned on, and cursor moved to next location), or ERASEd (current bit erased, and cursor moved to next location) in any of four directions: up, down, left, and right.

Three additional commands (- and : and /) allow for clearing the screen with a cursor "home"; "homing" the cursor without affecting the display; and jumping back to a resident monitor, respectively. An ASCII to HEX table is provided in the operating instructions in case these symbols are not clear, and as an aid to modify the monitor for other systems.

GRAFX' initialization routine does not destroy the display memory. This allows the user to develop complex graphics during successive sessions at the terminal, and to save and load displays for program usage.

GRAFX may be located anywhere in memory within 437 contiguous bytes, *but* IT MUST NOT BE RESIDENT WITHIN THE DISPLAY MEMORY! If it is, you will either crash or delete the monitor. Although GRAFX was designed for 2K of memory for the display, it may be used with only 512 bytes if the warning about "oddball" behavior is taken into account, and the cursor is kept on the screen.

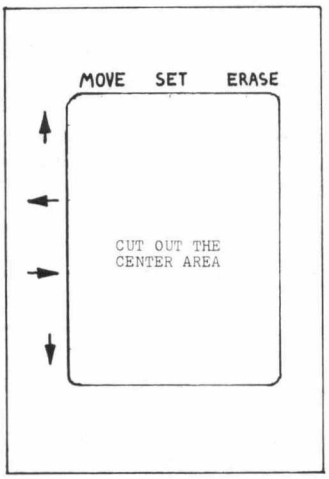
OPERATING INSTRUCTIONS

1. Load a display from a previous session into memory, if desired.
2. Load GRAFX. Be sure that it is not loaded into display memory.
3. Set hi byte of display starting address into sense switches A15 thru A8. DISPLAY MUST BE ADDRESSED IN 512 BYTE INCREMENTS! Thus an address such as 0100H or 5700H is illegal, although 0000H or 5800H is alright.
4. Begin execution of GRAFX at the INIT routine. Your Dazzler should output either a pattern (memory bit states) or a display if you loaded one in step 1. If you loaded a display, and it wasn't output, check the sense switches. If they are correct, start looking through memory by repeating steps 3 and 4 for different addresses: the display may have been loaded into a location other than what you had intended. **IF YOU DISPLAY THE MONITOR, AND KEY A LEGAL COMMAND OTHER THAN "/" (BREAK), YOU HAVE A 99.9% CHANCE OF CRASHING IT!**
5. When the display is active, the monitor is now executive. Only the following keys will be active (all others will be gobbled by GRAFX; none will be echoed to your teminal):

ASCII	HEX	OPERATION
–	2D	Clear screen, home the cursor to upper left of screen
;	3A	Home the cursor
/	2F	Break; return to the calling system
0	30	Move the cursor down one location (bit)
,	2C	Set cursor at current location, then move cursor down one bit
.	2E	Erase current bit, then move cursor down one bit
1	31	Move cursor right one bit

2	32	Set current bit, then move cursor right one bit
3	33	Erase current bit, then move cursor right one bit
4	34	Move cursor left one bit
5	35	Set current bit, then move cursor left one bit
6	36	Erase current bit, then move cursor left one bit
7	37	Move cursor up one bit
8	38	Set current bit, then move cursor up one bit
9	39	Erase current bit, then move cursor up one bit

Since it is rather too much to keep referencing a table, I have made an overlay that fits nicely over the numeric keypad on my Hazeltine 1500. With modifications, you should be able to use it on *any* numeric keypad. (Besides changing the cutout, this could include modifying routines INCHR and MON1 to accept other possible ASCII characters.) Here is a scaled-down copy of my overlay. The original was cut out of acetate sheet, and the lettering done in light blue Testor's PLA.



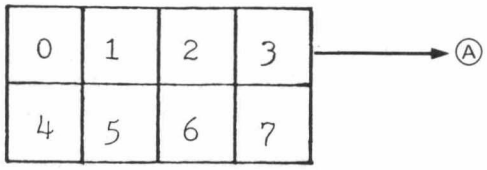
This overlay will make it a lot easier to use GRAFX. As you see, each key's function is a combination of its X-coordinate (MOVE, SET, or ERASE) and its Y-coordinate (UP, LEFT, RIGHT, and DOWN).

THEORY OF OPERATION

GRAFX uses a transformation table to allow it to interpret the Dazzler representation of memory. Big words! What that means is shown in the diagram:

GRAFX' INTERPRETATION OF A BYTE IN MEMORY

Each "bit" is actually a location in the table.

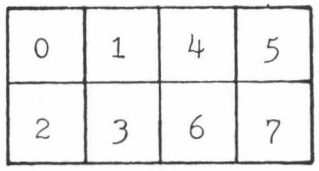


(Bits are numbered 0 to 7, LSB to MSB)

Is transformed by using this table of cursor corresponding values to match. . .

MEMORY LOCATION	VALUE
ANY	01H
ANY+1	02H
ANY+2	10H
Ⓐ → ANY+3	20H
ANY+4	04H
ANY+5	08H
ANY+6	40H
ANY+7	80H

THE TV DAZZLER'S REPRESENTATION !

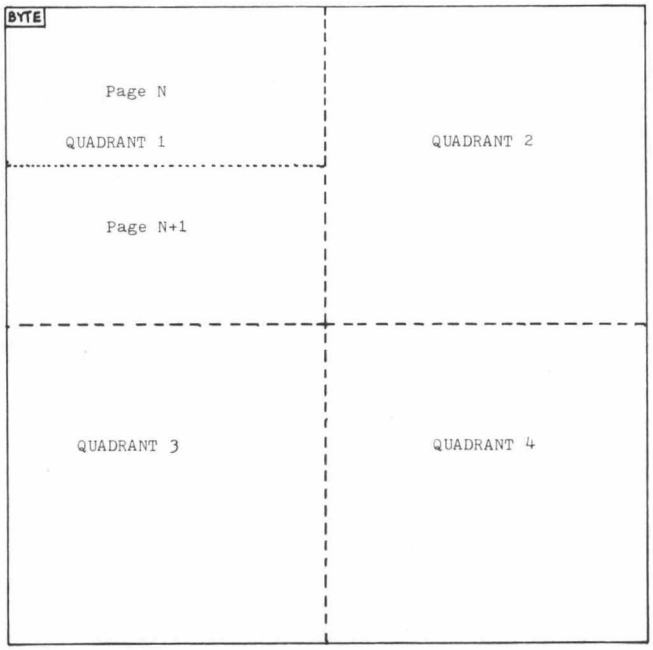


(Bits are numbered 0 to 7, LSB to MSB)

Note the "doubleZ" pattern of the Dazzler representation. If the Dazzler had been implemented so that it actually used GRAFX' byte interpretation, simple RRC instructions would replace this entirely too complicated table look-up kluge!

Using this table we can now move the cursor horizontally (by incrementing to the next word in the table) or vertically (by adding or subtracting an offset – 04H – to the current word in the table). Of course, special checks are needed to insure that the cursor hasn't crossed either a word boundary (i.e., left the table of values) or a page boundary (i.e., left one of the four quadrants of the total display). And this brings us to the "big picture": how GRAFX transforms four unfortunate quadrants into one big one that is 128H by 126V.

The diagram below shows how the Dazzler divides up the 2K of display memory:

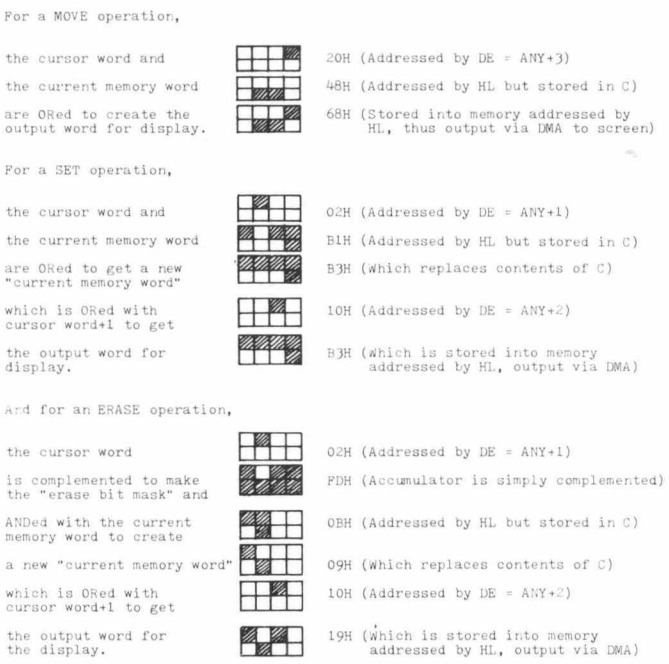


Each quadrant contains 512 bytes in a 16H by 32V byte square. Since the Dazzler formats each byte into a rectangle that is 4H by 2V bits, each quadrant is seen to be 64H by 64V bits square.

To get our 128H by 128V interpretation, GRAFX uses the registers H and L as a memory word pointer. Using suitable offsets for the quadrants' internal boundaries (i.e., wrap boundaries, denoted by solid lines), and cursor bit can be made to move along what *appears* to be a path that is continuous. Actually, that cursor bit is madly dancing through your computer's memory!

With this as a guide, you should be able to figure out the boundary checks and various offsets used in the routines of GRAFX.

The last thing to cover is the operation of the cursor output. This is done pictorially to give a "single-step" view of a MOVE, a SET, and an ERASE. Note that ▨ indicates that the bit is set (turned one; equal to 1) and that □ means the bit is reset (turned off; equal to 0).



And, bringing up the rear is the inevitable dump, this one in hex. After examining the tiny BASICs printed so far, I changed my mind about assembling GRAFX into location 0100H. Instead, I have located it at 0C00H (the start of the fourth "K") so that you can have Tiny BASIC, or HI, or whatever, *and* GRAFX, *and* 2K worth of display in only 5.5K worth of memory.

One last note. My hard-copy device uses a 10-bit interface. Although I've checked and rechecked the object code listing, there *may* be a dropped bit in there. I doubt it, but if strange things occur, compare the object listing and the source listing. I *know* the source code is correct.

★ ★

Postscript. XX in the object listing indicates the two bytes reserved for the stack. In the source code listing, S is equivalent to SP, and P is the same as PSW in Intel assmeblers. Aside from that, you should be able to get up and running with your Dazzler and this software for less than 20¢ worth of electricity. . . and an hour of your time!

★ ★

TDL's VIDEO DISPLAY BOARD

News Release *Received: 77 Aug 29*

Technical Design Labs has introduced a video interface for S-100 Bus microcomputers.

The VDB is low in cost and yet provides the capabilities of video terminals costing thousands of dollars more. It consists of two boards, one piggybacked to the other. The unit occupies one edge connector on the bus, but takes up the space of two boards.

The VDB contains its own display buffer memory and provides two pages of display, each with 25 rows of 80-characters. The display buffer memory does not use any memory address, thus leaving the entire computer memory address intact for user programs.

This unit displays, in addition to the 96 upper and lower case ASCII characters with descenders, 64 unique display symbols, thus permitting a graphic resolution with 160 horizontal elements by 75 vertical elements. The display can accept data at a 400,000 character per second rate.

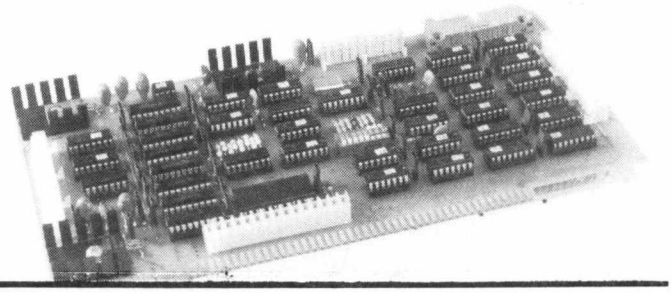
The VDB provides a true hardware blinking cursor to facilitate the programming of special edit functions. It is addressable and indicates on the display screen the physical location where the next symbol will be written into or read from.

A mode register allows any combination of characters to blink, insert or do both independently. Cursor and display may also be inhibited under mode register control.

The VDB works with either modified TV set or monitor and has an on-board 8 bit parallel keyboard port with status strobes.

The VDB is priced at $349 in kit form and $449 when assembled and tested. Software character and graphics output drivers for Z-80 and 8080 systems are supplied. These drivers are ROMable.

For additional information, contact Donna Galletti, Director of Sales, Research Park, Building H, 1101 State Road, Princeton, N J 08540. Telephone: (609) 921-0321.



MEMORY LOCATION	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0C00	21	87	0D	DB	FF	77	0F	F6	80	D3	0E	3E	68	D3	0F	C3
0C10	24	0C	2A	86	0D	7C	C6	08	36	00	23	BC	C2	18	0C	C3
0C20	24	0C	E1	71	11	88	0D	2A	86	0D	31	B6	0D	4E	1A	B1
0C30	77	DB	00	E6	40	CA	31	0C	DB	01	E6	7F	47	E5	31	96
0C40	0D	E1	3E	2C	B8	CA	52	0C	E1	3C	FE	3B	CA	31	0C	C3
0C50	44	0C	31	B4	0D	E9	C6	02	BC	C8	C6	02	BC	C9	3A	87
0C60	0D	C6	02	BC	C8	3C	BC	C8	C6	03	BC	C8	3C	BC	C9	1A
0C70	2F	A1	C3	77	0C	1A	B1	4F	E1	3A	90	0D	BB	DA	87	0C
0C80	13	13	13	13	C3	2E	0C	71	3E	EF	BD	DA	9D	0C	1B	1B
0C90	1B	1B	7D	C6	10	6F	D2	2D	0C	24	C3	2D	0C	3A	87	0D
0CA0	3D	CD	56	0C	CA	B2	0C	CD	56	0C	C2	8E	0C	C6	FA	C3
0CB0	B4	0C	C6	02	67	C3	8E	0C	1A	2F	A1	C3	C0	0C	1A	B1
0CC0	4F	E1	3A	90	0D	BB	DA	D1	0C	13	BB	DA	D9	0C	C3	2E
0CD0	0C	13	3A	92	0D	BB	D2	2E	0C	71	3E	0F	A5	FE	0F	CA
0CE0	E6	0C	23	C3	F3	0C	CD	5E	0C	01	F1	FD	CA	F2	0C	01
0CF0	F1	01	09	1B	1B	1B	1B	C3	2D	0C	1A	2F	A1	C3	02	0D
0D00	1A	B1	4F	E1	3A	90	0D	BB	DA	34	0D	1B	3A	94	0D	BB
0D10	C2	2E	0C	71	3E	0F	A5	FE	00	CA	24	0D	2B	13	13	13
0D20	13	C3	2D	0C	CD	5E	0C	01	0F	FE	CA	30	0D	01	0F	02
0D30	09	C3	1D	0D	1B	BB	DA	2E	0C	C3	13	0D	1A	2F	A1	C3
0D40	44	0D	1A	B1	4F	E1	3A	90	0D	BB	DA	7F	0D	71	3E	0F
0D50	BD	DA	65	0D	3A	87	0D	D6	02	CD	56	0C	CA	74	0D	CD
0D60	56	0C	CA	79	0D	13	13	13	13	7D	D6	10	6F	D2	2D	0C
0D70	25	C3	2D	0C	C6	06	C3	7B	0D	D6	02	67	C3	65	0D	1B
0D80	1B	1B	1B	C3	2E	0C	00	00	01	02	10	20	04	08	40	80
0D90	8B	0D	8F	0D	87	0D	75	0C	12	0C	6F	0C	00	A6	78	0C
0DA0	C1	0C	BE	0C	B8	0C	03	0D	00	0D	FA	0C	45	0D	42	0D
0DB0	3C	0D	22	0C	XX	XX										

ROUTINES FLAGS

	C	Z	S	P	AC
1					
2					
3					

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

```
ADDR    LABEL   OPERATION       COMMENTS
        * This is the non-destructive initialization routine.  It will
        * read the starting address for the 2K of display from the sense
        * switches, then format this address for the Dazzler.  The display
        * is then set for the 2K byte, 128H by 128V, black-and-white mode,
        * and the routine ends by outputting the cursor in the "home" loc.
        INIT    LXI   H,BEGIN+1 Store hi byte of display start address here
                IN    0FFH      Get start address from the sense switches,
                MOV   M,A       Store it,
                RRC             Adjust byte to indicate the start address
                ORI   80H       For the TV Dazzler,
                OUT   0EH       And output it to the Dazzler.
                MVI   A,68H     This byte indicates that display is B&W
                OUT   0FH       And 128H by 128V.  Output it to Dazzler.
                JMP   HOM1      Go initialize monitor and output cursor.
        *
        * This routine will clear the screen by setting the 2K of memory
        * to zero. The routine ends by initializing the monitor and
        * outputting the cursor in the "home" location at upper left screen
        *
        CLRSC   LHLD  BEGIN     Put display start address in HL
                MOV   A,H       Put hi byte of display start address in A
                ADI   8         Create display end address
        CLR1    MVI   M,0       Clear memory word
                INX   H         Point to next memory word
                CMP   H         Is it the end of display address?
                JNZ   CLR1      No-continue clearing the screen
                JMP   HOM1      Yes-go initialize monitor, output cursor
        *
        * This routine will HOME the cursor to the upper left of the screen
        * It also contains the routine HOM1 that initializes the monitor's
        * stack, cursor word pointer, and memory pointer.
        HOME    POP   H         Get memory pointer from the stack
                MOV   M,C       Restore current memory word to RAM
        HOM1    LXI   D,CRSWD   Initialize the cursor word pointer
                LHLD  BEGIN     Initialize the memory pointer
                LXI   S,TOP     Set pointer to top of monitor stack
        *
        * The following routine is the monitor memory-mapped output
        * routine.  It is entered via HOM1 by routines HOME, INIT, & CLRSC
        * The cursor control routines jump to CRSOR if the memory pointer
        * was changed, but if it remains the same jump to CRS1.
        *
        *
        *
```

ROUTINES FLAGS

	C	Z	S	P	AC
1					
2					
3					

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

```
ADDR    LABEL   OPERATION       COMMENTS
        CRSOR   MOV   C,M       Get new memory word and make it current
        CRS1    LDAX  D         Put cursor word in A
                ORA   C         Cursor word+memory word=output word!
                MOV   M,A       Output A to screen
        *
        * INCHR is the graphic monitor's link to your keyboard.  It is
        * written for a Processor Tech 3P+S using ports 0 for status and
        * 1 for the data word.  If you modify this to accomodate your own
        * system, be sure not to use the stack, or registers C,D,E,H,L.
        * Register B is used to hold the transmitted byte while it is
        * checked against legal commands by MON1.
        *
        INCHR   IN    0         Get status word
                ANI   40H       AND it with status mask
                JZ    INCHR     No input, so jump back for another try
                IN    1         It's input! Put it in A
                ANI   7FH       Mask off parity bit
                MOV   B,A       Save ASCII character in B
                PUSH  H         Save memory pointer on stack
                LXI   S,ADDRS   Set stack pointer to start of jump table
                POP   H         Put first routine address in HL
                MVI   A,2CH     Put first legal character in A
        MON1    CMP   B         Is input character a legal command?
                JZ    MON2      Yes-go prepare for jump to routine
                POP   H         No-get next routine address off stack
                INR   A         Increment to next compare character
                CPI   3BH       Is it legal?
                JZ    INCHR     No-go back and get another character
                JMP   MON1      Yes-go and compare it with input character
        MON2    LXI   S,STACK   Set stack pointer to bottom of stack
                PCHL            Jump to routine whose address is in HL
        *
        * This routine is used by MOVDN and MOVUP to check for page
        * boundaries within the 2K byte display.  Z=1 indicates that H
        * holds a page boundary for which we are checking.
        *
        PGCK2   ADI   2         Increment hi address byte in A by two
                CMP   H         Is it equal to hi address byte in H?
                RZ              Yes-return from subroutine with zero set
                ADI   2         No-increment A by two
                CMP   H         Is it equal to hi address byte in H?
                RET             Return with zero set if it is.
        *
        *
```

ROUTINES	FLAGS					
		C	Z	S	P	AC
1						
2						
3						

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

ADDR	LABEL	OPERATION	COMMENTS
	*		This routine is the page check boundary routine for MOVRT and
	*		MOVLF. It returns with zero set if H contains one of the page
	*		boundaries in question.
	*		
	PGCK1	LDA BEGIN+1	Put hi byte of display start address in A
		ADI 2	Increment by 2 to get first page boundary
		CMP H	Is H equal to page boundary?
		RZ	Yes-return with zero set
		INR A	No-increment by 1 to get next page boundary
		CMP H	Is H equal to page boundary?
		RZ	Yes-return with zero set
		ADI 3	No-increment by 3 to get next page boundary
		CMP H	Is H equal to page boundary?
		RZ	Yes-return with zero set
		INR A	No-increment by 1 for last check
		CMP H	Is H equal to page boundary?
		RET	Return. Zero will be set if condition met
	*		
	*		These two routines (ERSDN and SETDN) are used to set the current
	*		bit of the current memory word held in register C to either 0
	*		for ERSDN or 1 for SETDN. Both routines then enter MOVDN to move
	*		the cursor to the next location.
	*		
	ERSDN	LDAX D	Put cursor word in A
		CMA	Complement it to create erase bit mask
		ANA C	Erase bit mask+memory word=output word
		JMP SETD	Go make output word current in register C
	SETDN	LDAX D	Put cursor word in A
		ORA C	Cursor word+memory word=output word
	SETD	MOV C,A	Make output word current in register C
	*		
	*		This routine moves the cursor down one location on the screen.
	*		MOVDN and MOVUP operate in a similar manner. Since each eight
	*		bit word is "folded" into two four bit nybbles, the routine will
	*		first try to move the cursor into either the hi nybble (MOVDN) or
	*		the lo nybble (MOVUP). If it can't, then either a word or a page
	*		boundary has been reached. The routines each check for valid
	*		page boundaries, and if none are found, assume that a word
	*		boundary has been reached. In this case, HL is set to the next
	*		memory word, and the cursor word reset. If at a page boundary,
	*		the memory word is offset according to the page.
	*		Note that ERSDN and SETDN enter MOVDN via SETD. No jump is
	*		involved.
	*		

ROUTINES	FLAGS					
		C	Z	S	P	AC
1						
2						
3						

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

ADDR	LABEL	OPERATION	COMMENTS
	MOVDN	POP H	Get memory pointer from stack
		LDA LOBYT	Put termination byte of lo 4 cursor words
		CMP E	in A and compare it with cursor word pointer
		JC DOWN1	Cursor word pointer is in hi 4 words
		INX D	Cursor word pointer is in lo 4 words, so
		INX D	Add 4 to pointer to put it at the hi word
		INX D	Corresponding to the current lo word
		INX D	Done!
		JMP CRS1	Go output cursor
	DOWN1	MOV M,C	Restore current word to memory
		MVI A,0EFH	L will be greater than this if at page
		CMP L	Boundary; is it?
		JC DOWN2	Yes-go find out which boundary it is
	DOWN5	DCX D	Cursor word pointer is in hi 4 words, so
		DCX D	Subtract 4 from the pointer to put it at
		DCX D	The lo word corresponding to the current
		DCX D	hi cursor word.
		MOV A,L	Put lo byte of memory pointer in A
		ADI 10H	Offset it to next lower memory word
		MOV L,A	Put it back in L
		JNC CRSOR	If no carry out of L, go output cursor
		INR H	Otherwise increment H to adjust pointer
		JMP CRSOR	And go output cursor
	DOWN2	LDA BEGIN+1	Put hi byte of display start address in A
		DCR A	Adjust it for page check routine
		CALL PGCK2	Check page boundaries; are we there?
		JZ DOWN3	Yes-go add offset for page jump to HL
		CALL PGCK2	No-check last two boundaries; are we there?
		JNZ DOWN5	No-go set pointer to next word
	DOWN4	ADI 0FAH	Yes-add wrap offset to A
		JMP DWN3A	Go put new hi byte of memory pointer in H
	DOWN3	ADI 2	Add page offset to A
	DWN3A	MOV H,A	Put it in H
		JMP DOWN5	Go set memory pointer to next byte and bit
	*		
	*		ERSRT and SETRT are the two routines called to write into
	*		memory (and thus the screen) or erase memory (and the screen)
	*		Both routines enter MOVRT via SETR
	*		
	ERSRT	LDAX D	Put cursor word in A
		CMA	Complement it to create erase bit mask
		ANA C	Erase bit mask+memory word=output word
		JMP SETR	Go make output word current in C
	SETRT	LDAX D	Put cursor word in A

ROUTINES	FLAGS					
		C	Z	S	P	AC
1						
2						
3						

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

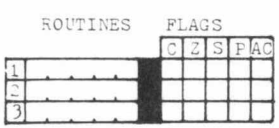
ADDR	LABEL	OPERATION	COMMENTS
		ORA C	Cursor word+memory word=output word
	SETR	MOV C,A	Make output word current in register C
	*		
	* MOVRT and MOVLF operate in a manner similar to MOVDN, but the		
	* cursor word pointer is kept in either the lo 4 words or the hi		
	* 4 words, depending upon the line. A word boundary is thus		
	* reached when the cursor word pointer moves out of the current		
	* (i.e., lo or hi) word set. Page boundaries for MOVRT and MOVLF		
	* are checked by the routine PGCK1.		
	*		
	MOVRT	POP H	Get memory pointer from stack
		LDA LOBYT	Put termination byte of lo 4 words in A
		CMP E	Is cursor word pointer in hi 4 words?
		JC RT1	Yes-go to hi 4 word routine
		INX D	No-set cursor word pointer to next word
		CMP E	Is it still in the lo 4 word set?
		JC RT2	No-go check page boundaries
		JMP CRS1	Yes-go output cursor
	RT1	INX D	Set cursor word pointer to next word
		LDA HIBYT	Put termination byte of hi 4 words in A
		CMP E	Is cursor word pointer still in hi 4 words?
		JNC CRS1	Yes-go output cursor
	RT2	MOV M,C	Restore current memory word to RAM
		MVI A,0FH	Put mask in A
		ANA L	Save least significant 4 bits of L
		CPI 0FH	Are they equal to 0FH?
		JZ RT3	Yes-indicates a page boundary; go check it
		INX H	No-just a word boundary. Increment HL.
		JMP RT4	Go reset cursor word pointer
	RT3	CALL PGCK1	Check for valid page boundaries
		LXI B,0FDF1H	Load wrap offset into B just in case
		JZ RT5	At wrap page boundary; go add offset
		LXI B,01F1H	Put page offset into B
	RT5	DAD B	Add offset in B to H
	RT4	DCX D	Reset cursor word pointer by subtracting
		DCX D	4 from it.
		DCX D	
		DCX D	
		JMP CRSOR	Go output cursor
	*		
	* ERSLF and SETLF write		and erase the current bit, then move the
	* cursor to the left one		location. Both enter MOVLF via SETL.
	*		
	*		

ROUTINES	FLAGS					
		C	Z	S	P	AC
1						
2						
3						

	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

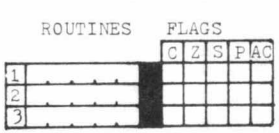
ADDR	LABEL	OPERATION	COMMENTS
	ERSLF	LDAX D	Put cursor word in A
		CMA	Complement it to create erase bit mask
		ANA C	Erase bit mask+memory word=output word
		JMP SETL	Go make output word current
	SETLF	LDAX D	Put cursor word in A
		ORA C	Cursor word+memory word=output word
	SETL	MOV C,A	Make output word current in C
	*		
	* This routine moves the		cursor one location to the left on the
	* screen. Operates in a		manner similar to MOVRT.
	*		
	MOVLF	POP H	Get memory pointer from stack
		LDA LOBYT	Put termination byte of lo 4 words in A
		CMP E	Is cursor word pointer in hi 4 words?
		JC LEFT1	Yes-go to hi 4 word routine
		DCX D	No-Set cursor word pointer to next word
		LDA BYTE1	Indicator that pointer has dropped out of
		CMP E	lo 4 words from bottom; has it?
		JNZ CRS1	No-go output cursor
	LEFT5	MOV M,C	Restore current memory word to RAM
		MVI A,0FH	Put mask in A
		ANA L	Save least significant four bits of L
		CPI 00H	Are they equal to zero?
		JZ LEFT2	Yes-indicates page boundary;go check it
		DCX H	No-just a word boundary. Decrement HL.
	LEFT4	INX D	Reset cursor word pointer by adding 4 to it
		INX D	
		INX D	
		INX D	
		JMP CRSOR	Go output cursor
	LEFT2	CALL PGCK1	Check for page or wrap boundaries
		LXI B,0FE0FH	Put page offset in B just in case
		JZ LEFT3	At page boundary; go add offset to HL
		LXI B,020FH	Put wrap offset in B
	LEFT3	DAD B	Add offset in B to H
		JMP LEFT4	Go reset cursor word pointer & output crsor
	LEFT1	DCX D	Set cursor word pointer to next word
		CMP E	Dropped out of hi 4 words into lo 4 words?
		JC CRS1	No-go output cursor
		JMP LEFT5	Yes-go check boundaries
	*		
	* ERSUP and SETUP write		and erase the current bit, then move the
	* cursor one location up.		Both enter MOVUP via SETU.
	*		



	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

ADDR	LABEL	OPERATION	COMMENTS
	ERSUP	LDAX D	Put cursor word in A
		CMA	Complement it to create erase bit mask
		ANA C	Erase bit mask+memory word=output word
		JMP SETU	Go make output word current
	SETUP	LDAX D	Put cursor word in A
		ORA C	Cursor word+memory word=output word
	SETU	MOV C,A	Make output word current in C
	*		
	* This routine moves the cursor up one location on the screen.		
	* It operates in a manner similar to MOVDN.		
	*		
	MOVUP	POP H	Get memory word pointer off stack
		LDA LOBYT	Put termination byte of lo 4 words in A
		CMP E	Is cursor word pointer in hi 4 words?
		JC UP1	Yes-go to hi 4 word routine
		MOV M,C	No-Restore C to RAM & check boundaries
		MVI A,0FH	Put boundary indicator word in A
		CMP L	If L ≤ 0FH we're at boundary; are we?
		JC UP2	No-just a word boundary. Go output cursor.
		LDA BEGIN+1	Yes-check for page or wrap boundary
		SUI 2	Adjust hi byte of memory pointer for PGCK2
		CALL PGCK2	Check for wrap boundaries
		JZ UP3	At wrap boundary; go add offset to HL
		CALL PGCK2	Check for page boundaries
		JZ UP4	At page boundary; go add offset to HL
	UP2	INX D	Reset cursor word pointer to appropriate
		INX D	Hi or lo memory word
		INX D	
		INX D	
		MOV A,L	Move memory pointer up to next word
		SUI 10H	Offset to move memory pointer
		MOV L,A	Restore L
		JNC CRSOR	No need to adjust H; go output cursor
		DCR H	Ooops! Carry set. Adjust H due to borrow.
		JMP CRSOR	Go output cursor
	UP3	ADI 6	Add offset to H since it is in A
		JMP UP4A	Go restore H
	UP4	SUI 2	Subtract page offset from H since it is in A
	UP4A	MOV H,A	Restore H
		JMP UP2	Go reset cursor word pointer,ouput cursor
	UP1	DCX D	Set cursor word pointer to lo word that
		DCX D	Corresponds to current hi word.
		DCX D	
		DCX D	One more step on the next page...



	ROUTINE 1	ROUTINE 2	ROUTINE 3
A			
B			
C			
D			
E			
H			
L			
SP			

PAGE: OF:
FLAG STATUS AS OF END ROUTINE

ADDR	LABEL	OPERATION	COMMENTS
		JMP CRS1	Go output cursor
	*		
	*		
	*		
	*		
	* System RAM includes mask words to take care of the complex		
	* assignment of screen locations to each bit in a memory word,		
	* and a jump table for use by MON1. The graphic monitor's two		
	* byte stack resides at the end of the jump table, allowing us to		
	* save a step, since input of an illegal character pops H		
	* automatically. It also allows only one PUSH or CALL at a time!		
	*		
	BEGIN	DW 00H	Storage for display start address
	CRSWD	DB 01H	Lo cursor words follow
		DB 02H	
		DB 10H	
	LOWRD	DB 20H	LOWRD defines termination byte of lo 4 wrds
		DB 04H	Hi 4 cursor words follow
		DB 08H	
		DB 40H	
	HIWRD	DB 80H	HIWRD defines termination byte of hi 4 wrds
	LOBYT	DW LOWRD	Clumsy way to get termination bytes is due
	HIBYT	DW HIWRD	To the limited set of pseudo-ops in the
	BYTE1	DW CRSWD-1	IMSAI SCS-2 Assembler. But, it's free!
	ADDRS	DW SETDN	
		DW CLRSC	
		DW ERSDN	My relocated SCS-2 monitor starts at A600H
		DW 0A600H	You will have to make a change to get the
		DW MOVDN	Graphics monitor to jump you back to your
		DW MOVRT	Monitor.
		DW SETRT	
		DW ERSRT	
		DW MOVLF	
		DW SETLF	
		DW ERSLF	
		DW MOVUP	
		DW SETUP	
		DW ERSUP	
		DW HOME	
	STACK	DS 2	Here's the monitor's stack
	TOP	EQU STACK+2	And this is the pointer to its top.
		END	That's all folks! 437 bytes worth.
	*		

ALPHANUMERIC PLOTTER FOR POLY VIDEO DISPLAY

By Arthur Armstrong

Dear Jim,

77 Sep 8

This is a manuscript and program for you to consider for publication. If you are not paying but would like to print my article please do so. I will settle for eternal fame.

Thanks for the job you are doing with the *Journal*. You have put me on to EMPL and PILOT and I expect that these two alone will be worth the ticket.

Sincerely,
Arthur Armstrong
3345 Moore St
Los Angeles, CA 90066

Editorial Note: Profuse thanks for the material and since we do not pay for contributions we bestow herewith eternal fame. – TRW

After using my Polymorphic video display board for about six months for drawing graphs I wanted a way that would be easier to use and quicker in execution. The BASIC programs calculating and poking into the VDM memory, while challenging to write, were complex and slow. This article presents the results of my attempt to improve the process. I won't mention all the things I tried that didn't work. Perhaps someone more familiar with MITS BASIC than I can offer something better. This patch is for MITS 3.2 8K BASIC. I suspect most basics would be comparable.

The video display is considered to have coordinates from 0 to 127 from left to right and from 0 to 47 from bottom to top. The plot function (PLT) uses the first two variables defined in the program to locate the horizontal and vertical coordinate, respectively. Thus, to draw a diagonal from bottom left to top right:

```
10 X=0:Y=0
20 FOR X=0T0127
30 Y=X/2.7
40 PLT
50 NEXT
```

The plot function, PLT, replaces LET in the BASIC instruction list. Since LET is optional in MITS BASIC this will present no limitations although it may cause some trouble if you have programs from other sources using LET. It is not difficult to change from LET to PLT and conversely as will be described below.

In MITS 3.2 BASIC there is an end-of-program pointer which contains the address of the first available memory location after the BASIC program text. The pointer is located (low, high) at hex 0245 and 0246. BASIC uses this pointer to locate storage locations for program variables. Variables are stored sequentially in the order the program defines them. Thus, if your program starts with X=0:Y=0 then X will be the first variable in memory and Y will be the second. This structure is utilized by PLT to find the desired values to be plotted.

Variables are stored in floating point form. To avoid going into details on floating point, suffice it to say that: Variables are stored in six bytes. Bytes one and two are used to store the variable label or name. Bit seven (most significant bit) of byte five is used for the sign of the variable. The bit is zero for a positive number and one for a negative. Bit seven of byte six is always one if the variable is greater than or equal to 0.5 in magnitude and zero otherwise. Bits six through zero of byte five represent the most significant part of the binary representation of the number (with an implied one in bit position seven). Bits six through zero of byte six represent an exponent or an amount to shift the value of byte five to obtain a binary integer for the variable.

This is all the information needed by the plot function. PLT will determine the coordinates of the point to be plotted, determine if they are within the correct range for the display, calculate an address in the display memory for the point, calculate the correct value needed to generate the graphics character, and store this in the calculated address.

To convert your 3.2 8K MITS BASIC to use this function:
- change location 00A2 from 4C to 50 to change L of LET to P.
- change location 00A3 from 45 to 4C to change E of LET to L. (location 00A4 contains T which is not changed)
- change location 017E from 14 to the low address byte of PLT in your memory.
- change location 017F from 07 to the high address byte of PLT. (changing these four locations back will restore LET)
- insert the machine code corresponding to the listing below.
- be sure that you use your board address in line 1430.

To use PLT:
- be sure that the first variable defined in your program corresponds to the vertical coordinate.
- write a program that will generate X and Y in the proper range: X from 0 to 127, Y from 0 to 47. There is no error if out-of-range values are used or if fractional values are used.

Each time that BASIC encounters the instruction PLT it will jump to your patch and the patch will plot the point.

Example:
```
10 X=0:Y=0
20 FOR X=0 TO 127
30 Y=10*SIN(X/10)+23
40 PLT
50 NEXT
```

```
0000                    0010 * POLY PLOTTER
0000                    0011 * ART ARMSTRONG
0000                    0015 * 7/27/77
0000                    0018 * ASSEMBLED AT 0000 FOR TRANSCRIPTION
0000                    0019 * CONVENIENCE ONLY.  WON'T RUN.
0000                    0020 * CHANGES IN 3.2 BASIC:
0000                    0021 * AT 00A2 CHANGE 4C TO 50
0000                    0022 * AT 00A3 CHANGE 45 TO 4C
0000                    0023 * SET THIS ADDRESS:
0000                    0024 * LOW AT 017E, HIGH AT 017F
0000                    0030 PSW EQU 6
0000 C5                 0040  PUSH B
0001 D5                 0042  PUSH D
0002 E5                 0044  PUSH H
0003 21 7A 00           0050  LXI H,EXIT
0006 E5                 0051  PUSH H * SET UP FOR OFF SCREEN EXIT
0007 16 02              0100  MVI D,2
0009 2A 45 02           0110  LHLD 0245H * END OF PROGRAM POINTER.
000C 47                 0120 CONV MOV B,A * X INTO B ON PASS TWO.
000D 23                 0130  INX H
000E 23                 0131  INX H
000F 23                 0132  INX H
0010 23                 0133  INX H
0011 23                 0134  INX H * H,L POINTS TO EXPONENT
0012 7E                 0135  MOV A,M * GET  EXPONENT
0013 B7                 0140  ORA A * IS IT ZERO?
0014 CA 37 00           0145  JZ ZERO * YES, NUMBER IS ZERO TOO
0017 E6 7F              0150  ANI 7FH * IF BIT 7 IS 1 THEN
0019                    0151 * NUMBER IS >= 0.5, USE IT.
0019 BE                 0155  CMP M
001A C2 21 00           0160  JNZ EXPOK
001D AF                 0162  XRA A * IF NO.<.5 FORCE TO ZERO
001E C3 37 00           0163  JMP ZERO
0021 FE 08              0165 EXPOK CPI 08H * IF EXP > 8
0023 D0                 0170  RNC * THEN NUMBER > 127, EXIT
0024 5F                 0175  MOV E,A * EXPONENT TO E
0025 2B                 0180  DCX H
0026 7E                 0185  MOV A,M * GET FRACTION
0027 F6 80              0190  ORI 80H
0029 BE                 0195  CMP M * IS IT NEGATIVE?
002A 23                 0196  INX H
002B C8                 0200  RZ * YES, EXIT
002C E5                 0212  PUSH H
002D 6F                 0214  MOV L,A
002E 26 00              0216  MVI H,0
0030 29                 0220 ADJST DAD H * SHIFT FRACTION
0031 1D                 0225  DCR E * ACCORDING TO EXPONENT
0032 C2 30 00           0230  JNZ ADJST
0035 7C                 0240  MOV A,H
0036 E1                 0245  POP H
0037 15                 0260 ZERO DCR D
```

COSMAC HLL?

Dear Dr. Dobb, *Received: 77 Jul 19*

Does anyone know if any high-level language has been written for the RCA COSMAC (1802)? (A similar request appeared in the March 1977 issue of *DDJ.*) I will be getting a COSMAC system soon (the "COSMAC ELF" which was written up in *Popular Electronics*) and will probably write a tiny BASIC (or may be something else) for it, unless one already exists.

After I get my systems running, I expect to be able to send some software to *DDJ.* **[You say the nicest things.]**

Jim Howell 555 Candlestick Way
San Jose, CA 95127

HEATH PSEUDO-FACT

Heath is supposed to be planning on including a video-input jack on their models of TV kits . . . jus' for us.

```
0038 23                 0262  INX H
0039 C2 0C 00           0265  JNZ CONV * BACK TO CONVERT Y VALUE.
003C                    0270 *
003C                    0272 * NOW HAVE X IN B REG AND Y IN ACC.
003C 62                 0275  MOV H,D * ZERO TO H
003D 4F                 0280  MOV C,A * Y INTO C
003E 3E 2F              0290  MVI A,2FH
0040 91                 0300  SUB C * IS Y>47?
0041 D8                 0310  RC * YES, EXIT
0042                    1230 * D CONTAINS ZERO FOR QUOTIENT.
0042 D6 03              1240 DIV3 SUI 3 * DIVISION BY THREE
0044 DA 4B 00           1250  JC DVEND *GO IF DONE
0047 14                 1260  INR D * ELSE BUMP QUOTIENT
0048 C3 42 00           1270  JMP DIV3  * DO IT AGAIN
004B C6 03              1280 DVEND ADI 3 * RESTORE REMAINDER IN A
004D                    1282 * AND CLEAR CARRY.
004D                    1285 *  AT THIS POINT THE D REGISTER
004D                    1286 * CONTAINS INT((47-Y)/3)
004D                    1288 * WITH THE REMAINDER IN THE A REG
004D 4F                 1290  MOV C,A * SAVE REMAINDER
004E                    1295 * NOW BUILD THE ADDRESS.
004E 6A                 1300  MOV L,D * QUOTIENT. (H IS ZERO)
004F 29                 1320  DAD H * MULT BY 64
0050 29                 1321  DAD H
0051 29                 1322  DAD H
0052 29                 1323  DAD H
0053 29                 1324  DAD H
0054 29                 1325  DAD H
0055 78                 1350  MOV A,B * GET X
0056                    1360 * CARRY IS STILL CLEARED
0056 1F                 1370  RAR * DIVIDE BY TWO
0057                    1375 * A HAS INT(X/2)
0057                    1376 * REMAINDER IN CARRY BIT
0057 F5                 1380  PUSH PSW * SAVE CARRY
0058 85                 1390  ADD L
0059 6F                 1400  MOV L,A
005A 7C                 1410  MOV A,H
005B                    1420 *
005B                    1421 * IMPORTANT!
005B                    1422 *
005B                    1423 * LINE 1430 MUST ACI THE HIGH BYTE
005B                    1424 * OF YOUR BOARD ADDRESS.
005B                    1425 *
005B CE 7C              1430  ACI 7CH * M.S.BYTE OF SCREEN ADDRESS.
005D                    1434 *
005D 67                 1440  MOV H,A
005E                    1450 * H,L HAS ADDRESS FOR PLOT CHARACTER.
005E                    1452 * = BOARD+INT((47-Y)/3)+INT(X/2)
005E                    1560 * NOW CREATE THE GRAPHICS SYMBOL
005E F1                 1600  POP PSW * GET CARRY
005F 3E 04              1610  MVI A,4
0061 DA 66 00           1620  JC REM1 * GO IF X REMAINDER IS 1
0064 3E 20              1630  MVI A,20H
0066 0C                 1650 REM1 INR C * CHECK INITIAL ZERO
0067 0D                 1660 SYM DCR C * C COUNTS Y REMAINDER.
0068 CA 6F 00           1670  JZ POKIT
006B 0F                 1680  RRC * DIVIDE BY TWO
006C C3 67 00           1690  JMP SYM
006F                    1691 * READY TO STORE SYMBOL ON SCREEN.
006F                    1692 * USE LEFT LIST FOR WHITE ON BLACK.
006F                    1693 *   (BLACK IS 3F)
006F                    1694 * USE RIGHT LIST FOR BLACK ON WHITE.
006F                    1695 *   (WHITE IS 00)
006F                    1696 *
006F                    1697 *
006F 4F                 1700 POKIT MOV C,A *      NOP
0070 3E 3F              1710  MVI A,3FH *        NOP
0072 96                 1715  SUB M *            NOP
0073 B1                 1720  ORA C *            NOP
0074 4F                 1725  MOV C,A *          NOP
0075 3E 3F              1730  MVI A,3FH *        NOP
0077 91                 1735  SUB C *            ORA M
0078 77                 1740  MOV M,A *          MOV M,A
0079 E1                 1745  POP H * CLEAR STACK
007A E1                 1750 EXIT POP H
007B D1                 1760  POP D
007C C1                 1770  POP B
007D C9                 1780  RET
?
```

MODIFIED MICROTEC ASSEMBLER SYSTEM

Dear Sir,

I have just finished rewriting and testing my version of the system originally written by Microtec of Sunnyvale, CA. Here is a brief description of the changes that I have made.

Lines entered without a number but preceded by a space are automatically given a number 5 greater than the last line of the file, and added to the file.

The lines of a file may be renumbered, with the user specifying the starting value and the increment.

The PAGE commands has been replaced by COPY which will copy a section of memory from one place to another. The start, end, and destination addresses are specified by the user.

The addition of the string editing functions, FIND and ALTR, allowing the correction of individual characters in a line, without replacing the entire line.

A MOVE command has been added which will move blocks of lines from one place in a file to another.

An ASC pseudo op has been added to the assembler, simplifying the assembly of ASCII character strings.

Provisions have been made, through the use of the CONT command, to perform multiple file assemblys, with or without the use of an external storage device.

Additionally, numerous other corrections and additions have been made throughout the program.

This version of the system was written to run on an IMSAI 8080 and requires slightly over 4K for the system, which expands to 5K with the use of the software memory-protect feature of the IMSAI memory boards. This unused memory is free for the user's mass storage I/O drivers. 2K is required for the system's working storage and the assembler's symbol table.

I rewrote the system for my own use but would like to make it available to others either by having it published or directly. I do not know if the original is in the public domain or not. The program listing is 60 pages, and the source deck is 3K cards. I am willing to make copies of the listing and/or the source deck. I do not have the facilities to make cassette tapes.

Scott C. Crumpton 11130 NW 9 Pl
Gainesville, FL 32601

P.S. – I am sending a similar letter to *Kilobaud, Byte* and *Interface Age.*

CHANGE THE ANS FLOWCHART STANDARD?

The ACM's SIGDOC group has started a review of the flowchart standard (ANS X3.5-1970). The committee seeks written comments and suggestions for the improvement, modification, or change of the current standard. Or, if no change is appropriate at this time, please say so.

All types and levels of suggestions and comments are welcome. All concerned persons in the computer field are elgible. You have knowledge and experience - - let them count by sending your written comments and suggestions to Ned Chapin, Flowchart Committee Chairman, InfoSci Inc., Box 7117, Menlo Park, CA 94025.

UTILITIES & MUSIC on the COSMAC ELF

by Edward M. Mc Cormick

Received 77 Sept 22

It is possible to do a number of interesting things with the basic Elf at an additional hardware cost of less than $5. This article describes programs to read and write programs and data to and from ordinary cassette recorders, to play music, to interface with a Teletype, to measure frequencies and to measure time intervals. The trick is to trade software for hardware by taking advantage of the precise timing control possible in the Elf. These applications require a 2 megahertz crystal.

It will be noted that these programs deviate from normal programming practice because of the need to have the execution time the same no matter which logical path is followed and to assure that both halves of a tone being generated are exactly equal.

As a practical matter, it is often advantageous to prenumber all core before entering any programs or data into the Elf. When the program in Table 1 is executed, the content of each storage location is the address of that location. Thus, especially when entering programs manually, it is easy to see if the next unused location is what or where it should be. One can then check if too many or too few bytes have been entered. This can be important in debugging.

Cassette Read-Write

The so-called 'Kansas City' standard makes it feasible to use ordinary cassette recorders for program or data storage. This standard uses 8 cycles of a 2400 hertz tone for a mark (logical 1) and 4 cycles of a 1200 hertz tone for a space (logical 0). Eleven bits are required for each byte; first there is a space to indicate the beginning of the asynchronous transmission, then eight marks and spaces for the byte itself, and finally two marks to finish. The byte rate is thus about 27 per second.

The program in Table 2 will write the 152 bytes in hex locations 68 through FF onto tape in about 6 seconds. The program for doing this is stored in 00 through 62. The hardware for this interface is given in Figure 1. A register is used

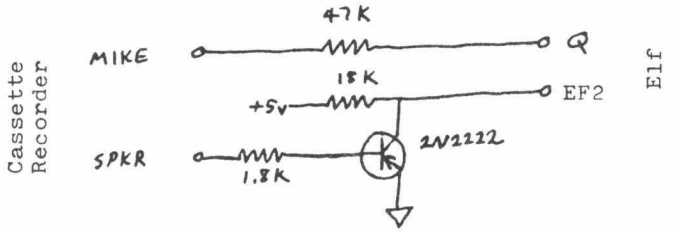


Figure 1. Circuit for interfacing cassette to Elf.

to reduce the voltage across the Q LED to the level needed at the mike input of the cassette recorder. (It may be necessary to experiment with this input level especially if the recorder does not have autolevel.) On play back the cassette speaker (or earphone) output is applied to a 2N2222 transistor to provide clean logical ones and zeroes. (The output should be set at the maximum to assure this.) The transistor output is connected to EF2 in the micro.

When the recording program runs it will first generate a steady 2400 hertz tone. After recording about a 10 second leader, the data recording is initiated by depressing the IN switch. The resulting data burst can be seen on the display and heard on the speaker. (This presumes the speaker amplifier given in the preceding article.) A trailer tone should also be recorded.

The program for reading back from cassette is given in Table 3. It will read the bytes on tape into hex locations 68 through FF. The Elf RUN switch should be turned on only when the cassette is playing back the 2400 hertz header. When the data itself is encountered it will be displayed and can be heard in the speaker. If core has been prenumbered it will generally stop with FF in the display. The recorder should be stopped while in the trailer.

Pressing the IN switch then causes the program starting in hex 68 to be executed. When the RUN switch is turned off the micro is ready to read another program from the cassette unit. To re-execute the program at 68 the 02 in location 01 must be changed temporarily to 68.

Music Playing

It is possible to program the Elf to reproduce the notes of the musical scale. A program for doing this is given in Table 4.

The music program requires two bytes of data for each note and the second the pitch. Table 5 gives the hex values for whole note duration and pitch for various notes. For example, a whole note middle C is 41 33, a half note F is 2B 24, etc. These pairs of bytes must immediately follow the music program, that is, start in hex AC. The data for the first 8 bars of Simon's 'Feeling Groovy' is given in Table 6.

The tempo of the music depends on the value in hex 77. The greater the value, the slower the tempo. The usual range is 10 to 20.

Incidentally, this version of the music playing program starts at hex 68 and thus can be read by the cassette read program given above. However, this leaves space for only 43 notes in a basic Elf. If the program starts at 00, then 91 notes can be stored.

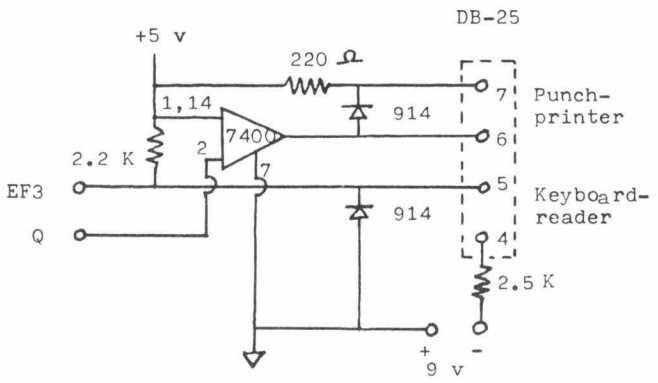


Figure 2. Hardware interface for 20 mil Teletype.

Teletype Interface

The circuit in Figure 2 will provide the hardware interface between the micro and a 20 mil Model 33 Teletype. The signal to drive the Teletype comes from Q, the input from the Teletype is on EF3.

The program needed for both reading and writing is given in Table 7. When first executed, the first part of the program (locations 00 through 3E) causes characters read from the keyboard or tape reader to be written into successive core positions starting at 7A. Whenever core is filled or the IN switch is operated, the second part of the program (locations 3F through 6F) reads from core back to the Teletype causing printout and tape punching if desired. Again, after reading all core or when the IN switch is operated it will revert back to reading from the Teletype. This program illustrates the basic input-output techniques and can be adapted for specific needs.

Note that the system will read all 8 bits from a byte of core and will punch all 8 bits. Similarly, all 8 bits on tape will read back into core. However, striking a key on the Teletype enters only 7 information bits since it is an ASCII device. Likewise, the printer will ignore the 8th (most significant) bit. Thus, Elf programs as such must be initially entered from switches or a hex keyboard; however, this system will subsequently keep all 8 bits intact.

Frequency Counter

The input used for reading back from cassettes can also be used when the micro is programmed as a frequency counter. The program for this is shown in Table 8.

When the IN switch is operated, the program counts the number of cycles occurring in a one second interval and displays the most significant byte of that count. Operating the IN switch again causes the least significant byte to be displayed. Finally, operating the IN switch results in a second frequency count, etc.

The input signal should overdrive the 2N2222 to assure clean logical 0 to 1 crossings. The maximum input frequency is about 5800 hertz.

Interval Timer

By using a similar technique, it is possible to measure time intervals with the ELF. The program is given in Table 9.

When the IN switch is operated the micro waits for the first 0 to 1 crossing. The count in the hex display is in 16 microsecond units; thus the maximum time with this program is about one second. The program could be modified for decimal display and for larger time intervals.

Table 1. Program to prenumber core.

Loc	Instr
00	E2
01	F8 09
03	A2
04	82
05	52
06	64
07	30 04

Loc	Instr	Remark
00	E1	X=1
01	7A	Turn Q off
02	F8 68	Set first core
04	A1	* address
05	F8 00	Clear
07	A6	* reg 6
08	A7	* reg 7, eob sw
09	F8 10	Generate a mark,
0B	A2	* 8 cycles at
0C	F8 01	* 2400 hertz
0E	A3	*
0F	30 11	*
11	31 16	If Q on, turn
13	7B	* it off, if
14	30 19	* off, turn it
16	7A	* on
17	30 19	*
19	83	Variable delay
1A	FF 01	* to balance
1C	3A 1A	* half cycles
1E	82	Repeat if cycle
1F	FF 01	* count not
21	A2	* zero
22	32 2E	*
24	F8 07	Fixed delay
26	FF 01	*
28	3A 26	*
2A	30 2C	*
2C	30 11	*
2E	86	End of mark
2F	3A 3D	* test
31	3F 09	If IN up,
33	F8 01	* continue mark
35	A6	*
36	F0	Start by getting
37	A4	* byte,
38	F8 09	* setting shift
3A	A5	* count in reg 5,
3B	30 4B	* going to space
3D	87	Branch if end
3E	3A 5A	* of byte
40	85	If not, reduce
41	FF 01	* shift count and
43	A5	* branch if end
44	32 53	* of byte
46	84	Go to mark or
47	76	* space according
48	A4	* to bit of byte
49	33 09	*
4B	F8 08	Generate a space,
4D	A2	* 4 cycles at
4E	F8 0E	* 1200 hertz
50	A3	*
51	30 11	*
53	F8 01	At end of byte,
55	A7	* set eob switch
56	F8 20	* and start double
58	30 0B	* mark
5A	F8 00	When end of byte,
5C	A7	* display byte and
5D	64	* if end of core,
5E	81	* go to mark, if not,
5F	32 01	* get next byte
61	30 36	* and return

Table 2. Program to write to cassette from Elf

Loc	Instr	Remark	Loc	Instr	Remark
00	30 02	Optional branch	32	FC ED	Branch if mark
02	E1	X=1	34	3B 3F	*
03	F8 68	First "load-to"	36	7A	If space, turn
05	A1	* address	37	F0	* Q off,
06	F8 80	Put a one in	38	F6	* shift output
08	A7	* reg 7	39	51	* byte and set
09	F8 08	Put shift count	3A	F8 59	* delay for
0B	A5	* in reg 5	3C	A2	* 2.5 ms
0C	7B	Turn on Q	3D	30 49	*
0D	F8 00	Clear byte to	3F	7B	If mark, turn
0F	51	* be loaded	40	F0	* Q on, add one
10	35 10	Loop on marks	41	F6	* to byte, shift
12	3D 12	* in header of	42	51	* it, restore
14	F8 00	* tape or	43	87	* it and
16	FC 01	* between	44	F4	* set delay
18	35 16	* bytes	45	51	* for 2.9 ms
1A	FC ED	*	46	F8 68	*
1C	3B 10	*	48	A2	*
1E	F8 B3	If space, start	49	85	Continue if not
20	A2	* 5 ms delay	4A	FF 01	* all 8 bits in
21	7A	*	4C	A5	* byte
22	82	Do delay using	4D	3A 22	Continue if not
23	FF 01	* duration in	4F	64	* all core loaded,
25	C4	* reg 2	50	81	* display byte,
26	3A 23	*	51	3A 09	* and return
28	35 28	Determine if	53	7A	If core loaded,
2A	3D 2A	* mark or	54	3F 54	* turn off Q and
2C	F8 00	* space at	56	37 56	* execute program
2E	FC 01	* sampling	58	30 68	* when IN down
30	35 2E	* time			

Table 3. Program to read from cassette into Elf core.

Loc	Instr	Remark	Loc	Instr	Remark
68	E5	X=5	89	3A 87	* indicates
69	F8 AC	Put first data	8B	89	Repeat as often
6B	A5	* in reg 5	8C	FF 01	* as tempo
6C	F0	Stop if data	8E	A9	* indicates
6D	3A 70	* is 00	8F	3A 99	*
6F	00	*	91	88	Repeat as often
70	A8	Store duration	92	FF 01	* as duration
71	15	* in reg 8	94	A8	* indicates
72	64	Display pitch,	95	3A 76	*
73	25	* store it in	97	30 A1	*
74	F0	* reg 7	99	C4	Delay to make
75	A7	*	9A	C4	* alternate
76	F8 10	Store tempo	9B	30 9D	* paths take
78	A9	* in reg 9	9D	30 9F	* same time
79	87	Stop alternating	9F	30 79	*
7A	FC B4	* Q if a rest	A1	7A	When note done,
7C	33 86	*	A2	15	* turn off Q,
7E	31 83	If Q on, turn	A3	F8 0E	* and insert
80	7B	* it off; if	A5	B3	* short quiet
81	30 86	* off, turn it	A6	23	* interval
83	7A	* on	A7	93	* between
84	30 86	*	A8	3A A6	* notes
86	87	Repeat as often	AA	30 6C	Get next note
87	FF 01	* as pitch			

Table 4. Program to play music.

Note	Dur	Pitch	
D	93	12	
C#	8B	14	
C	83	15	
B	7B	17	
A#	75	19	
A	6E	1B	
G#	68	1D	
G	62	1F	
F#	5D	22	
F	57	24	
E	52	27	
D#	4E	2A	
D	49	2D	
C#	45	30	
*C	41	33	*Middle C
B	3E	37	
A#	3A	3B	
A	37	3F	
G#	34	43	
G	31	47	
Rest	2D	4C	

Table 5. Hex values for whole note duration and pitch of musical notes.

Loc	Data	Loc	Data
AC	31 1F	D6	31 1F
AE	2C 24	D8	29 2A
B0	83 24	DA	83 24
B2	25 2D	DC	25 2D
B4	52 2A	DE	29 2A
B6	2C 24	E0	57 24
B8	25 2D	E2	25 2D
BA	94 2D	E4	94 2D
BC	17 4C	E6	17 4C
BE	1D 3B	E8	31 1F
C0	31 1F	EA	31 1F
C2	29 2A	EC	29 2A
C4	57 24	EE	57 24
C6	25 2D	F0	25 2D
C8	2C 24	F2	2C 24
CA	2C 24	F4	2C 24
CC	57 24	F6	57 24
CE	3A 19	F8	49 12
D0	75 19	FA	49 12
D2	57 24	FC	AF 19
D4	2D 4C	FE	00

Table 6. Data for portion of Simon's "Feeling Groovy".

Loc	Instr	Remark	Loc	Instr	Remark
00	F8 28	Delay subr exit	37	F8 76	Else set delay
02	A6	* in reg 6	39	30 1E	* for 9 ms
03	7B	Turn on Q	3B	64	Display byte
04	E1	X=1	3C	81	*
05	F8 70	First read-to	3D	3A 0B	Read next byte
07	A1	* address	3F	F8 70	First read-from
08	F8 80	Put a one in	41	A1	* address
0A	A7	* reg 7	42	7A	Start space
0B	F8 00	Clear mx	43	F0	Store output
0D	51	*	44	A4	* byte
0E	F8 08	Shift count to	45	F8 09	Shift count to
10	A5	* reg 5	47	A5	* reg 5
11	3E 19	Start if EF3=0	48	F8 4F	Set subr exit
13	3F 11	Loop if IN up	4A	56	*
15	37 15	Loop if IN down	4B	F8 76	Set delay for
17	30 3F	Branch otherwise	4D	30 1E	* 9 ms
19	F8 29	Set subr exit	4F	84	Modify output
1B	56	*	50	76	* according to
1C	F8 B2	Set 13.5 ms delay	51	A4	* bit in byte
1E	C4	Delay loop subr	52	33 57	*
1F	C4	*	54	7A	*
20	C4	*	55	30 58	*
21	C4	*	57	7B	*
22	C4	*	58	85	Return if not
23	FF 01	*	59	FF 01	* all 8 bits
25	3A 1E	*	5B	A5	*
27	30 00	Subr exit	5C	3A 4B	*
29	F0	Form byte	5E	7B	Start mark
2A	F6	* according	5F	F8 66	Set subr exit
2B	51	* to EF3 at	61	56	*
2C	3E 31	* sampling	62	F8 ED	Set delay for
2E	87	* time	64	30 1E	* 18 ms
2F	F4	*	66	64	Display byte
30	51	*	67	81	* just put out
31	85	Branch if all	68	32 05	All core go to read
32	FF 01	* 8 bits read	6A	3F 42	Else start next byte
34	A5	*	6C	37 6C	Loop if IN down
35	32 3B	*	6E	30 05	Return to read

Table 7. Program to read from Teletype to Elf and vice versa.

Loc	Instr	Remark	Loc	Instr	Remark
00	3F 00	Wait for IN to	25	C4	*
02	37 02	* be operated	26	30 15	*
04	F8 2C	Store values	28	F8 01	Zero side,
06	A2	* for one	2A	A3	* no count
07	F8 32	* second	2B	30 2D	*
09	B2	*	2D	30 15	*
0A	F8 00	Clear freq	2F	14	One side,
0C	A4	* count	30	F8 00	* add to freq
0D	B4	*	32	A3	* count
0E	A3	*	33	30 15	*
0F	3E 13	Wait for 0-1	35	E1	Display high
11	36 11	* transition	36	F8 46	* order
13	3E 13	*	38	A1	* byte of
15	22	Exit if end	39	94	* freq
16	92	* of second	3A	51	* count
17	32 35	*	3B	64	*
19	83	Monitor if on	3C	21	*
1A	32 22	* zero or	3D	3F 3D	Wait for IN to
1C	36 2F	* one side	3F	37 3F	* be operated
1E	C4	One side,	41	84	Display low
1F	C4	* after count	42	51	* order byte
20	30 15	*	43	64	* freq cnt
22	3E 28	One side,	44	30 00	Start over
24	C4	* no count			

Table 8. Frequency count program.

Loc	Instr	Remark	Loc	Instr	Remark
00	3F 00	Wait till IN	16	F8 26	Display high
02	37 02	* sw depressed	18	A1	* hex position
04	F8 00	Clear registers	19	92	* of interval
06	A2	*	1A	51	* count
07	B2	*	1B	64	*
08	A3	*	1C	21	*
09	3E 0D	Wait for first	1D	3F 1D	Wait for IN
0B	36 0B	* 0 to 1	1F	37 1F	* sw depressed
0D	3E 0D	* crossing	21	82	Display low
0F	12	Add when	22	51	* hex position
10	36 0F	* EF3=1	23	64	* of interval
12	12	Add when	24	30 00	Repeat
13	3E 12	* EF3=0	26		Storage
15	E1	Exit at end			

Table 9. Program to measure time intervals.

PROMABLE 1K OPERATING SYSTEM FOR RCA ELFS

Edward McCormick
P O Box 863
Norco, CA 91760

Received: 77-Sept-22

There are three significant limitations to the basic Elf microcomputer. They are (1) limited memory (2) lack of an operating system and (3) volatility of memory. This article describes an overall system for overcoming these limitations at a cost of $25 more than the basic Elf itself. It is based on improvements given in Part 3 of the series on the COSMAC Elf which appeared in the March 1977 *Popular Electronics.* The system uses 1K RAM, a 256 byte EPROM operating system and an ordinary cassette to load and unload RAM. It operates immediately when turned on. The operating system facilitates program execution and manipulation of the data in RAM.

The circuit of Figure 1 is the interface between the cassette unit and the micro. The Q output of the micro goes to the *Mike* input of the cassette unit through a suitable dropping resistor and also to a transistor amplifier to drive a small monitoring speaker. The *Speaker* (or *Earphone*) output of the cassette unit goes to another transistor amplifier which provides input to the micro through EF2 (pin 23 of the 1802).

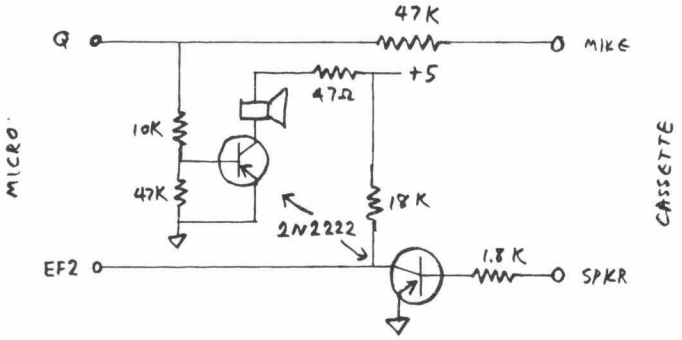


Figure 1. Hardware interface between Elf and Cassette unit.

The 1K 2102 RAM memory is that described in Part 3 of the *Popular Electronics* article. It is addressed as pages 04 through 07. Incidentally, there is an error in Figure 1 (page 65) of Part 3. A10 connects to pin 11 and $\overline{A10}$ to pin 12, not the reverse as shown.

The operating system is based on ETOPS also in Part 3 of the series. It performs five functions (1) start program execution at any place in memory, (2) read from memory, (3) write to memory, (4) read from cassette to RAM, and (5) write from RAM to cassette. The function codes (entered from the toggle switches) are 00 through 04 respectively. To start execution at any memory location the switches are first set to 00 and the RUN switch turned on. Next, the high order address byte is entered in the toggle switches and the IN switch is depressed. The low order byte is then set in the toggle switches and the IN is depressed a second time. Finally, depressing the IN switch the third time causes program execution to start at the designated address. (Register F is used as the program register.) Functions 01 and 02 also require similar entry of the starting address. The 01 function code results in displaying the contents of successive locations starting at the designated address. Code 02 causes the data in the switches to be loaded into successive memory locations. The Q LED will turn on indicating that memory is being overwritten. The operating system uses 07FE and 07FF of RAM.

The cassette functions do not require the address bytes since they read or write all 1024 bytes of RAM. When the switches are set at 04 (or more) and the RUN switch is then turned on, the output is a constant 2400 hertz tone which can be heard on the monitor speaker. About 10 seconds of this should be recorded as a header. Depressing the IN switch causes the 1024 bytes of memory to be read out in KC standard format in about 39 seconds. This is followed immediately by the 2400 hertz tone again to complete the process.

To enter data from the cassette unit, the switches are set at 03. However, the RUN switch should not be turned on until the unit is playing back in the header record. When the data is encountered it will be read into RAM and the process can be heard on the speaker and observed in the display.

If the cassette unit does not have autolevel, it may be necessary to experiment to find the correct input recording level. On output, it was found necessary to set the volume at maximum to assure the clean 0 to 1 and 1 to 0 crossings required by the read program. The timing in the cassette programs presume a 2 megahertz frequency to comply with the KC standard. Any other crystal frequency would require program modification.

The operating system is given in Table 1. This could be stored in the original 256 bytes of 2101 memory but would be volatile there. Using a 1702 EPROM eliminates this problem. A 1702 is used because they are inexpensive and programming services are readily available. The erasable feature is useful when one wants to change the operating system.

Loc	Contents															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	F8	07	B1	F8	FE	A1	E1	6C	64	FC	FD	33	3B	3F	0D	37
10	0F	6C	64	21	BF	3F	15	37	17	6C	64	21	21	AF	F0	EF
20	3F	20	37	22	32	34	F6	33	2B	7B	6C	64	3F	2C	37	2E
30	31	2A	30	2B	DF	00	00	00	00	00	00	3A	3E	7B	F8	04
40	B1	F8	00	A1	B2	B3	B4	B5	B6	B7	39	9E	F8	80	A7	F8
50	08	A5	7B	F8	00	51	35	56	3D	58	F8	00	FC	01	35	5C
60	FC	ED	3B	56	F8	B3	A2	7A	82	FF	01	C4	3A	69	35	6E
70	3D	70	F8	00	FC	01	35	74	FC	ED	3B	85	7A	F0	F6	51
80	F8	59	A2	30	8F	7B	F0	F6	51	87	F4	51	F8	68	A2	85
90	FF	01	A5	3A	68	64	91	FF	08	3A	4F	7A	00	00	7A	F8
A0	00	A6	A7	F8	10	A2	F8	01	A3	30	AB	31	B0	7B	30	B3
B0	7A	30	B3	83	FF	01	3A	B4	82	FF	01	A2	32	C8	F8	07
C0	FF	01	3A	C0	30	C6	30	AB	86	3A	D7	3F	A3	F8	01	A6
D0	F0	A4	F8	09	A5	30	E5	87	3A	F4	85	FF	01	A5	32	ED
E0	84	76	A4	33	A3	F8	08	A2	F8	0E	A3	30	AB	F8	01	A7
F0	F8	20	30	A5	F8	00	A7	64	91	FF	08	32	9E	30	D0	00

Table 1. Operating system for 1K Elf.

The pin connections for the 1702 are shown in Table 2. For convenience, they are shown in terms of pin connections to the 1802 itself. Note that a nine volt source is required by the 1702; however, this can be an ordinary battery since the current drain is about 15 milliamperes. Provision must be made to disconnect the battery since there is a three milliampere drain when the micro is turned off.

Since a memory dump takes only one minute even with header and trailer records, many can be recorded on a cassette. Good quality tapes should be used. Verbal explanations can also be included. A footage indicator is also quite useful. This system was developed using a Panasonic 212 unit but should work with other cassette units.

(Table 2→)

SUGGESTIONS FOR BETTER TINY LANGUAGES

Dear Dennis, June 2, 1977

I like tiny PILOT, especially the added Editing and control features compared to WANG PATB which I use. Several comments come to mind:

1. I don't like the intraline editing on PATB, echoing a Rubout with a backslash and no way to echo the corrected line except LIST. I would rather rubout echos the deleted character with the next rubout deleting and echoing the character before that, etc. Then I would use control R to give a CRLF and an echo of the corrected line thus far. Clear?

2. There are several ways for interfacing calculator chips to micros. (See National's publication: AN-176.) It seems to me a few bucks and a half dozen chips is a cheap way to get a whale of a lot of floating point math capability. Dr. Suding's interface 76 Jan) shows other possibilities. Why not design a language around such hardware or at least make it easy to add it later on an optional basis. It would be very slow but the benefits are great.

3. Perhaps two optional break points after RUN would help debugging. That is, execution would stop at the first of the line numbers encountered.

4. How about borrowing the GET, PUT and CALL (assembly language subroutines) from Livermore BASIC (*DDJ* 77 Jan)?

5. Why do you propose GOTO after your earlier statements (*DDJ* 76 Feb)? Is GOTO not a dirty word?

Keep up the good work.

John Bunting
2444 N. Delaware Apt K4
Springfield, MO. 65803

1702	1802	Other
1	27	
2	26	
3	25	
4	15	
5	14	
6	13	
7	12	
8	11	
9	10	
10	9	
11	8	
12		+5v
13		+5v
14		Pin 11 of 4042
15		+5v
16		-9v
17	32	
18	31	
19	30	
20	29	
21	28	
22		+5v
23		+5v
24		-9v

Table 2: Pin connections for the 1702 EPROM as related to the 1802 unless otherwise noted.

SIMPLIER VDM FIX

Dear Jim, *Received: 77 Jul 13*

Dan Parker, in the April issue, is quite right about the problem of the VDM display not necessarily corresponding with its storage in memory.

However, a simpler solution is to start the print program with a routine that outputs a row of spaces (64) at the beginning of the page.

Jim Callas, M.D. 631 No. San Pedro Road
San Rafael, CA. 94903

NOTES ON CP/M's BASIC-E

Dear Mr. Warren, *Received: 77 Jul 18*

I would like to give you and your readers some background on the current status of a public domain disk Basic known as BASIC-E. BASIC-E was developed at the Navy Postgraduate School, Monterey, California for my Masters thesis. The implementation is comprised of two subsystems, a compiler which generates code for a hypothetical stack machine and a run-time monitor which interprets this code. The system was written in PL/M and runs with Digital Research's CP/M.

BASIC-E provides standard floating point arithmetic processing features following the proposed ANSI standard. Additional extensions include multi-dimensional numeric and string arrays, logical operators (AND, OR, XOR, NOT) and string processing. Also, disk support methods include sequential and random access to files. Strings are dynamically allocated as required and may contain up to 255 characters. Concatenation and all standard functions are implemented. Logical operations use 32 bit two's complement binary representation of the integer portion of the variables. Files may be organized as either sequential or direct. Sequential files are a linear sequence of data items. Each reference to a sequential file retrieves the next data item or writes another data item. A file may also be declared with a user selected blocksize. These files may be accessed either sequentially or randomly.

The original version (version 1) was distributed by a number of companies including IMSAI and Digital Systems. There were several bugs in version 1 of BASIC-E. They involved file operations, string processing and some of the predefined functions! Also, because of the nature of the floating point package, integer numbers were often printed out with an error in the seventh digit. I have subsequently fixed the known bugs in version 1 and released version 2 of BASIC-E. Both version are completely in the public domain. Version 2 fixes the integer problem mentioned above by printing only six digits of the seven which are maintained internally. Unfortunately, there are still some copies of version 2 in use which have not had the integer problem corrected. Structured Systems Group (Oakland, CA.) has a patched version of BASIC-E which retains 7 digits without the integer problem (versions K2.0 of the run-time package).

I strongly recommend that anyone using BASIC-E obtain a copy of version 2. It is available through several distributors and at many retail outlets. I would be glad to provide a copy to dealers who need one for $30 copy charge. The source is also available in machine readable form for a $50 copy charge.

I would like to thank the many people who have shown an interest in BASIC-E and by using it have helped debug it. Special thanks go to Gary Kildall of Digital Research for converting it to the resident PL/M, and to Alan Cooper and Keith Parsons of Structured Systems Group who continued to find that last elusive bug.

Gordon E. Eubanks, Jr. 1333 No. Camino Alto, #151
Vallejo, CA. 94950

MULTIPLE COLUMN ACCOUNTING

Journal Balancing Program

Charlie Pack
25470 Elena Rd.
Los Altos Hills CA 94022

You are the owner of a small business. It is past the end of the month and it's 1:00 AM and you're trying to balance last month's income and expenses in a multi-column check register journal. All of the debit and credit columns have been added with a small calculator. But they don't balance. You're off by 13 cents! So you add the columns and rows all over again, trying to find the error. Finally, you find it, and re-add the column to get a new total. Now you're off by $57.62 and there is another error! Have you ever been in a situation like that? Then this program is for you!

The purpose of this program is to total and balance any kind of multi-accounting journal that has dollar amounts arranged in columns and rows (lines). The program adds columns vertically and rows horizontally (totals and crossfoots), and then prints various types of reports including a trial balance report and a statement of cash income and expenses. It also functions (with a video display terminal) as a powerful desktop adding machine.

There are several advantages in using a computer instead of an adding machine or calculator to balance a multi-column journal. It's true, of course, that in either case all of the numbers must be keyed in. But there the simillarity ends. The computer can "remember" every single debit and credit entry! It's like a calculator with an almost unlimited number of directly addressable memories, any of which can be recalled individually or by column or row instantly. There is no need to compute column totals and then have to re-enter them to get a trial balance (the sum of all the debit and credit column totals should balance, or add up to zero). The computer will do it for you automatically at the touch of a key. Errors are easier to find and correct because numbers don't have to be re-entered. Just enter only the corrected debit or credit amount and new totals can be computed immediately. Furthermore, the terminal shows a neat, formatted display which is much easier to read than 10 feet of adding machine tape. And, if you have a video display, it's silent and doesn't use up paper. So, let's put that home computer (almost anything that can run BASIC) to work!

As far as personal applications are concerned, I developed this program initially to total and crossfoot a 24-column Economik check register which I use in a personal double-entry (debits and credits) accounting system. In this respect, two of its greatest values have been to save time finding errors and to automatically provide data that I would not otherwise have time to do the calculations for. More specifically, each time I balance the check register I get an income statement, a cash flow budget vs. actual statement, and a running balance total on my checking account. I also use it any time I need an adding machine!

Version 1.0 of the multi-column accounting journal balancing program has 322 lines (including REMarks) and is optimized to run under MITS Altair 8K BASIC, revision 4.0. However, it is intended to run under any BASIC which supports 2-dimensional and string arrays, and supports string processing using the LEFT$, MID$ and RIGHT$ functions. A formatted print routine using the string functions is included in the program.

A great convenience that 4.0 BASIC provides is the ability to save data (separately from the program) on a cassette and later re-load it. This ability permits the updating of existing data, so I no longer feel the need to take time to write down the trial balance totals every month.

The journal balancing program itself requires about 9100 bytes of memory, but about 3300 bytes can be saved by removing all REMark statements. About 500 bytes additionally are needed for temporary variables and work space, not including the data matrix. The data matrix requires C*4*(R+1) bytes, where C is the number of columns and R is the number of rows. If 8K BASIC occupies 6378 bytes, it can be seen that 16K would be the minimum useful memory requirement.

I am running the program on a 24K Altair system. My personal check register journal requires 21 columns and I find that 90 rows are adequate to store three months transactions. The report formats are intended for a video display with lines at least 64 characters in length. Any kind of serial printer with adequate line length can also be used.

The journal balancing program "thinks" of a multi-column journal as a matrix containing a number of vertical columns and a number of horizontal rows. A column represents a column in the journal, for example "Office rent" or "Checking account deposits". A row normally represents a particular transaction or a line number on a journal page; or each row within a column can be used sequentially. In other words, the data matrix can be thought of as a number of elements, each element representing the intersection of a row and a column. Each element may contain a positive or a negative number, or zero. When the program is first started, all the elements contain zero values. The program allows the user to put numerical data into any element, after which it can compute row and column totals and display several types of reports. In normal use the contents of each element represents dollar amounts from some type of accounting journal or work sheet; but the program can be used almost anywhere an adding machine is required. All results are computed to two decimal places.

When the program is being used to balance an accounting journal or work sheet, all debits are identified as positive numbers and all credits are identified as negative numbers. A debit is an increase in an expense account, a decrease in an income account or an increase in an asset account (for example, a deposit to a bank checking account). A credit is a decrease in an expense account, an increase in an income

account or a decrease in an asset account (for example, a withdrawal from a bank account). Normally, expense items are debits and income items are credits. When a transaction is entered into a double-entry journal, the sum of the debit amounts must be equal to the sum of the credit amounts. The journal is in balance when the total of all the debits is equal to the total of all the credits. Since the program considers debits as positive values and credits as negative values, the grand total of all debits and credits entered into the data matrix should be zero. However, if a dollar amount is entered incorrectly, the journal won't balance. The computer can then help the user to find the error very quickly, by displaying the individual column and row totals without the necessity of re-entering any data.

In the journal balancing program, columns and rows may optionally be assigned special functions which determine how the data is processed by the program and how it appears on reports. The assignment of these special functions is controlled by assigning constant values to several subscript variables at the beginning of the program (this is explained in more detail later). The available special column functions are:

a) expense items
b) income items
c) transfers to another entity, such as a fixed asset ledger
d) transfer identification (in my system this is an account number in a fixed asset ledger). This number is an integer value and is not used in any calculations.
e) transaction identification (in my system this is a transaction date in the check register). This number is also an integer value and is not used in any calculations.
f) bank checking account deposits (debits)
g) bank checking account withdrawals (credits)
h) permanent asset and liability accounts. This includes bank checking account deposits and withdrawals, plus other accounts such as petty cash, savings account, credit card acounts payable, etc. Normally, asset accounts have debit balances and liability accounts have credit balances.

Two or more columns can be assigned to expense items, income items and permanent asset and liability accounts. Only one column can be assigned to each of the other special functions. All columns for a given function must be contiguous, and any one column can only have one special function.

The available special row functions are:

a) Cash flow estimate or budget entries. In this row, the numbers have opposite from normal signs; this is, expense items are credits, income items are debits, etc. With this facility, the program can tell how much you are over or under budget.
b) Beginning balances of asset and liability accounts. This allows the program to monitor and report cash flow and also to display the running balance on a bank checking account.

Only one row can be assigned to each of the special row functions. The program will not work correctly if both of the special row functions are assigned to the same row. Assignment of the special row functions is optional.

If the available special row and column functions are not assigned, the program functions as a simple adding machine with many directly addressable memories and the ability to add rows and columns. The income statement function does not work under these conditions.

The journal balancing program allows the user to manipulate elements of the data matrix by means of a simple command structure. The functions that can be performed are: add numbers algebraically to a matrix element; enter numbers into contiguous rows within a column; set elements or columns to zeroes; display the contents of any element or all elements of a row or column; display any or all row or column totals; and display a trial balance report or an income statement showing all of the column totals and grand totals. Particular attention is given to the ability to crossfoot and balance the matrix.

The general rules for entering commands are as follows: After loading the program and executing RUN, each command is prompted by the message: COMMAND (E, A, C, L, R, T, I, S, Q)? The user must enter a 1-letter function code followed, if required, by one or two 2-digit numbers which represent column or row numbers. The user must enter 1, 3 or 5 characters as required; examples are: E0426 L07 T. No embedded extra spaces or other characters are allowed. Column and row numbers must be two digits each and must not exceed the bounds of the data matrix. An appropriate error message will appear if a mistake is made. Following are the descriptions of each command. The letters cc mean a 2-digit column number and the letters rr mean a 2-digit row number. The first three commands are the data entry commands.

Accrr is used to accumulate numbers algebraically into the specified column/row element. The entry of each value is prompted by the message COL. cc ROW rr ACCUMULATE? Entry of a zero value will terminate the action. At that point the program rounds off the result to two decimal places. The program will not allow accumulation into the transfer ID or transaction ID columns (use the Enter command).

Eccrr is used to enter numbers into consecutive rows of the specified column cc , beginning at row rr . Thus, the first entry is stored in D(cc, rr) where D is the name of the data matrix. Subsequent entries will be stored in D(cc, rr+1), D(cc, rr+2), . . . , and so on. Each entry is prompted by the message COL. cc ROW rr ENTRY? . When a zero is entered, or when the row counter exceeds the maximum value allowed by the program, the user is automatically returned to command mode. Only positive whole numbers can be entered into the transfer ID or transaction ID columns (if assigned); anything to the right of the decimal point is ignored. Other data is rounded to two decimal places.

If an invalid entry such as an alphabetic character or embedded space is made, BASIC will display an error message such as REDO FROM START and the data may be re-entered. In the beginning balance row (if assigned), data may be placed in only the asset/liability account columns.

Cccrr clears the specified column/row element by setting it to zero.

Ccc clears the specified column, by setting its row elements to zeroes. The cash flow estimate (budget) entries and the asset/liability beginning balance entries are not cleared. It is intended that the Cccrr form of the command be used for that purpose.

Lccrr causes the program to display the value contained in the specified column/row element. Any element in any column or row may be examined regardless of its assigned special function.

Lcc causes the program to display all non-zero items within column cc, along with the column total. For each row item within the specified column, the associated column and row numbers, transfer ID, transaction ID and dollar amount are given. The beginning balances of asset accounts are not shown and not included in the column total (use the Lccrr or Rrr commands for that). If the specified column represents transfer ID or transaction ID, the dollar amounts are suppressed. Otherwise, the column total will include the estimated cash flow or budget entry for that column. Any column may be examined with this command.

Rrr causes the program to display all non-zero items within

row rr, along with the row total. In this manner any row may be examined.

R causes the program to display row totals for all rows except the row assigned to asset account beginning balances. For each row which contains at least one non-zero entry, the row number, transfer ID, transaction ID, row total and a running balance of the bank checking account are shown. The values of this multi-purpose command are that it will show immediately an out-of-balance condition in a double-entry system (all row totals should be zero); and it can be used to help reconcile the check register with bank statements. The Rrr form of the command may be used to display asset or liability account beginning balances.

T causes the program to list the column totals in the form of a trial balance report. For each column which contains at least one non-zero entry, this report shows the column number, column name, account balance, estimated cash flow or budget amount and column total. The algebraic sum of the account balance and the budget entry is equal to the column total. Columns which represent transfer ID or transaction ID are not shown as their contents are not dollar amounts. The last line of the report shows the grand totals. In a double-entry system if there are not errors the grand totals will be all zeroes. This is the trial balance.

I causes the program to display a statement of cash receipts and disbursements (similar to an income statement). This report is similar to the trial balance report, except the items are displayed in a different order (expenses, income items, transfers and asset or liability accounts) are included. For each of these columns which contains at least one non-zero entry, this report shows column number, column name, actual amount, budget amount and variance. The variance is the algebraic sum of the actual amount plus the budget amount (remember the latter has its sign opposite from normal). Columns which represent transfer ID or transaction ID are not shown. The last lines of the report show a reconciliation of the ending and beginning net cash balance. The difference between these two numbers added algebraically to the sum of expenses, income and transfers should be zero.

S (Stop) causes the program to stop execution and return to command mode in BASIC. The user may subsequently return to the journal balancing program with the contents of all variables left intact by typing CONT. The purpose of the Stop command is to allow the user to dynamically re-assign special column or row functions, change column names, etc. Warning: Do not create a syntax error when doing this, or BASIC will re-set itself and destroy all of the data previously entered into the data matrix. It is not possible to change the size of the data matrix using this method.

Q (Quit) causes the program to end and return to command mode in BASIC. This command is not intended to allow a return to the journal balancing program without typing RUN and re-initializing all of the variables. However, some versions of BASIC allow the user to execute a GOTO in immediate mode, which would allow a successful re-start.

A special function, such as "checking account deposits", is assigned by deciding which columns or rows are to be used for that function and then giving the appropriate column numbers or row numbers to special subscript values in the program. Columns and rows are numbered consecutively by ones starting with zero (most versions of BASIC allow the use of the 0th element), i.e. 0, 1, 2, 3, . . . , etc., up to the maximum allowed by the program. The maximum column number value is contained in variable C9 and the maximum row number is contained in variable R9. Therefore, if you want 20 columns and 50 rows, and you do not wish to use the 0th column or the 0th row, line 5040 in the program should read:

```
C9=20:R9=50
```

Not all columns and/or rows have to be assigned a special function; nor do all the special functions have to be used. If a special function is not wanted, its associated subscript variables must be given a value larger than the maximum allowed. It is suggested to use C9+1 or R9+1 because those values will work regardless of the values of C9 and R9. For example: if the data matrix is not to contain any asset or liability items, line 5070 should read:

```
A1=C9+1:A2=C9+1:B1=C9+1:B2=C9+1
```

Whenever the program wants to process a special function, a test is made first to determine if that speical function has been assigned.

The names of the subscript variables for special functions is given in the list of variable names included at the beginning of the program; refer to lines 1240 through 1350 and lines 1440 and 1450. The values for these variables are set in lines 5050 through 5080; the user should customize this part of the program for his or her particular application. When assigning special functions or setting the bounds of the data matrix, no other program changes are required other than those given at the beginning of the program. Special functions may be assigned permanently by changing the program (before RUNning it), or assigned temporarily by RUNning the program and then issuing a Stop command, making the changes, and CONTinuing the program.

The program contains a string array T$(C9) which has names (10 characters or less) for all the columns. T$(1) has the name for column 1, T$(2) has the name for column 2, and so on. When the program is first loaded for a particular application, the column name array should be set up as well as setting up the special functions and setting up the bounds of the data matrix. Refer to line 5120 in the program.

Structured logic and "top-down" programming techniques should make it easier to follow and maintain the program. In addition, a cross-reference listing of all referenced line numbers and variable names is provided to aid in making changes.

The main program loop is encompassed in lines 5430 through 5480. This routine looks up the command in a table (line 5390) and then does a GOSUB to the appropriate subrouting. Everything from line 5500 to the end of the program is invoked by a GOSUB.

The formatted print routine in lines 9000–9380 is designed to extract additional precision from floating-point variables by handling the integer and fractional parts separately. MITS BASIC will only print 6 digits of precision, even though over 7 digits of accuracy can be obtained. The largest amount that can be handled without losing precision is approximately $130,000.

Special function assignments used in run samples

```
LIST

5040 CLEAR 100
5040 C9=21:R9=91
5050 E1=1:E2=12:I1=13:I2=14
5060 X1=15:X2=16:X3=17
5070 A1=18:A2=21:B1=20:B2=21
5080 R1=0:R2=1
5090 DIM D(C9,R9)
5110 DIM T$(C9)
5120 T$(1)="AUTOMOBILE":T$(2)="LIC.&FEES"
5130 T$(3)="OFF. RENT":T$(4)="SUPPLIES"
5140 T$(5)="UTILITIES"
5180 T$(13)="SALES INC.":T$(14)="SERVICES"
5190 T$(15)="XFR ACCT #":T$(16)="XFR AMOUNT"
5200 T$(17)="TRANS. ID":T$(18)="CHG ACCTS"
5210 T$(19)="PETTY CASH":T$(20)="W/DRAWALS"
5220 T$(21)="DEPOSITS"
5390 F$="EACLRTISQ"

BREAK
OK
```

Run sample showing use of E, L and R commands

```
RUN
COMMAND (E,A,C,L,R,T,I,S,Q)? E0102
COL. 1 ROW 2 ENTRY? 1.02
COL. 1 ROW 3 ENTRY? 1.03
COL. 1 ROW 4 ENTRY? 1.04
COL. 1 ROW 5 ENTRY? 0
COMMAND (E,A,C,L,R,T,I,S,Q)? E0203
COL. 2 ROW 3 ENTRY? 2.03
COL. 2 ROW 4 ENTRY? 2.04
COL. 2 ROW 5 ENTRY? 2.05
COL. 2 ROW 6 ENTRY? 0
COMMAND (E,A,C,L,R,T,I,S,Q)? E0803
COL. 8 ROW 2 ENTRY? 8.02
COL. 8 ROW 3 ENTRY? 8.03
COL. 8 ROW 4 ENTRY? 0
COMMAND (E,A,C,L,R,T,I,S,Q)? E1702
COL. 17 ROW 2 ENTRY? 1702
COL. 17 ROW 3 ENTRY? 1703
COL. 17 ROW 4 ENTRY? 1704
COL. 17 ROW 5 ENTRY? 1705
COL. 17 ROW 6 ENTRY? 0
COMMAND (E,A,C,L,R,T,I,S,Q)? L0803
          8.03
COMMAND (E,A,C,L,R,T,I,S,Q)? 302          (Oops!)
INVALID COMMAND FUNCTION
COMMAND (E,A,C,L,R,T,I,S,Q)? L02
COL.#--ROW #--XFR ID--TXN ID----$ AMOUNT
    2       3             1703         2.03
    2       4             1704         2.04
    2       5             1705         2.05
                COLUMN TOTAL             6.12
COMMAND (E,A,C,L,R,T,I,S,Q)? R04
COL.#--ROW #----$ AMOUNT
    1       4          1.04
    2       4          2.04
  ROW BALANCE          3.08
COMMAND (E,A,C,L,R,T,I,S,Q)? R
ROW #--XFR ID--TXN ID--ROW TOTALS---CH ACC BAL
    0                                      .00
    2             1702         9.04
    3             1703        11.09
    4             1704         3.08
    5             1705         2.05
       TOTAL ALL ROWS         25.26
COMMAND (E,A,C,L,R,T,I,S,Q)? Q
OK
```

Run sample showing Trial Balance and Income Statement reports

```
COMMAND (E,A,C,L,R,T,I,S,Q)? T
COL.#---NAME---------ACCT BAL.------BUDGET---COL. TOTALS
   1   AUTOMOBILE       182.56        200.00-        17.44-
   2   LIC.&FEES         43.75         45.00-         1.25-
   3   OFF. RENT        350.00        350.00-          .00
   4   SUPPLIES          78.54         75.00-         3.54
   5   UTILITIES        139.82        120.00-        19.82
  13   SALES INC.    1,259.35-     1,100.00       159.35-
  14   SERVICES        184.55-       225.00         40.45
  20   W/DRAWALS       794.67-       790.00          4.67-
  21   DEPOSITS      1,443.90      1,325.00-       118.90
*****GRAND TOTALS         .00           .00          .00



COMMAND (E,A,C,L,R,T,I,S,Q)? I
COL.#---NAME----------AMOUNT-------BUDGET-----VARIANCE
  13   SALES INC.    1,259.35-     1,100.00       159.35-
  14   SERVICES        184.55-       225.00         40.45
*****TOTAL INCOME     1,443.90-     1,325.00       118.90-
   1   AUTOMOBILE      182.56        200.00-        17.44-
   2   LIC.&FEES        43.75         45.00-         1.25-
   3   OFF. RENT       350.00        350.00-          .00
   4   SUPPLIES         78.54         75.00-         3.54
   5   UTILITIES       139.82        120.00-        19.82
***TOTAL EXPENSES      794.67        790.00-         4.67
**SURPLUS/DEFICIT      649.23-       535.00        114.23-
****NET CASH FLOW      649.23-       535.00        114.23-
NET CASH BALANCE (ENDING)      759.23
NET CASH BALANCE (BEGIN.)      110.00     NET CASH FLOW      649.23
COMMAND (E,A,C,L,R,T,I,S,Q)?
```

```
SYMBOL  LINE NUMBERS WHERE REFERENCED

 5430   5460 5480
 5470   5450
 5490   5470
 5530   5470
 5610   5560 5630
 5640   5620
 5690   5530 5550
 5730   5470
 5770   5750 5860 5880
 5830   5810
 5840   5820
 5850   5870
 5890   5730 5740 5780 5850
 5930   5470
 6010   5940
 6040   6020
 6090   5930 5950 5960
 6130   5470
 6230   6140
 6310   6270
 6350   6310
 6390   6250 6350
 6490   6130 6150 6180 6390
 6530   5470
 6750   6650
 6840   6530
 6910   6850
 7010   6930
 7050   7030
 7070   7050
 7130   7080 7090
 7140   6910 7010
 7190   6560 6780
 7230   5470
 7430   5470
 7470   7450
 7530   7510
 7710   7610
 7910   7820
 7960   7930 7940
 8090   7810
 8640   7250 7460 7520 7620
 8710   8680
 8780   8730
 8810   8660 8710
 8910   7270 7480 7540 7590 7720 8790
 8990   8920
 9030   6180 6260 6280 6320 6360 6420 6710 6720 6780 6860 6880 7020 7040
        7060 7070 7110 7160 8030 8050 8070 8780 8910 8950
 9080   9050
 9140   9120
 9150   9110
 9210   9080 9140
 9250   9230
 9310   9210 9220 9240
 9350   9130
 9360   9350
 9730   5530 5730 5930 6130
 9830   5550 5740 5940 6150 6560
 9910   5460
 9920   5540 9730 9830
 9930   9740 9750 9840 9850
 9940   9770
 9950   5540
 9960   9870
 9970   5570 5760
 9990   9760 9860 9910 9920 9930 9940 9950 9960
 9999   5470

RECORDS READ - 176    SYMBOL TABLE ENTRIES FOUND - 139

SYMBOL  LINE NUMBERS WHERE REFERENCED

 A1     5070 5560 5570 5750 5760 5860 5870 7810 7850 7910
 A2     5070 5560 5570 5750 5760 5860 5870 7810 7850 7910
 B1     5070 5850 7080 7090 7110
 B2     5070 5850 5860 7080 7090 7110
 C      5530 5540 5570 5610 5630 5640 5650 5660 5730 5760 5770 5810 5820
        5830 5840 5870 5930 5960 6030 6130 6170 6250 6260 6350 6360 6370
        6390 6640 6650 6710 6720 6750 6920 6930 6960 7010 7850 7850 7910
        7930 7950 7990 8650 8660 8680 8690 8750 8780 8790 8810 9730 9740
        9750 9760 9770
 C1     7250 7460 7520 7620 8650
 C2     7250 7460 7520 7620 8650
 C9     5040 5050 5060 5070 5090 5110 5560 5750 5860 6270 6310 6640 6850
        6920 7030 7050 7080 7250 7450 7510 7610 7810 9760
 D(     5090 5630 5643 5650 5660 5820 5830 5843 5960 6030 6170 6250 6280
        6320 6360 6370 6650 6720 6860 6930 6960 7040 7060 7090 7110 7850
        7930 7950 8683 8690 8750
 E1     5050 7510 7520
 E2     5050 7510 7520
 F$     5390 5440 5450
 G1     7490 7550 7590 7630 7720
 G2     7490 7550 7590 7630 7720
 G3     7490 7550 7590 7630 7720
 H$     6550
 I1     5050 7450 7460
 I2     5050 7450 7460
 J      5440 5450 5460 5470
 K      5440 5940 6140 6530 6540 6560 9730 9830
 L      6170 6260 6280 6320 6360 6420 6710 6720 6780 6860 6880 7020 7040
        7060 7070 7110 7160 8030 8780 8910 8950 9040 9360
 L9     9350 9360
 M      6170 6260 6360 6420 6710 6720 6780 6860 6880 7020 7070 7160 8030
        8780 8910 9030 9050 9080 9110 9210
 N      6920 6960 7010 7920 7950 7960 8670 8690 8710
 P1     7270 7480 7540 7590 7720 8720 8750 8910
 P2     7270 7480 7540 7590 7720 8750 8950
 P3     7270 7480 7540 7590 7720 8750 8950

RECORDS READ - 176    SYMBOL TABLE ENTRIES FOUND - 269
```

```
SYMBOL  LINE NUMBERS *HERE REFERENCED

A       5550 5560 5610 5630 5640 5650 5660 5740 5750 5770 5820 5830 5840
        5850 5860 5950 5960 6010 6020 6030 6040 6150 6170 6240 6250 6260
        6280 6320 6360 6370 6390 6550 6650 6710 6720 6910 6930 6960 7020
        7040 7060 7080 7090 7110 7140 7920 7930 7940 7950 7960 8670 8680
        8690 8690 8710 9830 9840 9850 9860 9870
A1      5080 5550 5750 5860 6020 6250 6850 6860 6910 7080 7820 7850 7940
        8680
A2      5080 6020 7080 7940 8690 8730 8750 8920
A9      5040 5080 5090 5850 6010 6240 6850 6910 7080 7820 7920 8670 8730
        8920 9860
A$      5430 5440 5450 6550 6560 9740 9750 9760 9840 9850 9860
S2$     9740
S3$     9750
S4$     9840
S5$     9850
I       5620 5630 5640 5650 5660 5780 5820 5830 5840 6240 6370 6420 6640
        6720 6780 6920 6960 7070 7920 7950 7960 8670 8690 8720
I$(     5110 8750
I1      7270 7450 7480 7490 7510 7540 7550 7630 8640 8720
I2      7270 7450 7480 7490 7510 7540 7550 7630 8640 8750
I3      7270 7450 7480 7490 7510 7540 7550 7630 8640 8750
I4      6860 6880 7110
I5      6910 7070 7150
I6      7820 7850 8310 8050
I7      7910 7960 8310 8070
I8      8310 8330
J       6170 6260 6280 6320 6360 6420 6710 6720 6780 6860 6880 7020 7040
        7060 7070 7110 7160 8030 8050 8070 8780 8910 8950 9050 9060 9130
        9310
J$      9130 9320 9350 9380
J1      9050 9060 9070 9080 9110 9120
J1$     9080 9140 9150 9320
J2      9050 9060 9070 9220 9230
J2$     9210 9220 9240 9250 9320
J9$     9110 9140 9150 9230 9240 9250 9310 9320
X1      5060 5540 5810 6170 6270 6280 6350 6390 6650 6930 7030 7040 8660
X2      5060 7610 7620
X3      5060 5540 5810 6170 6310 6320 6350 6390 6650 6930 7050 7060 8660

RECORDS READ - 176    SYMBOL TABLE ENTRIES FOUND - 272
```

```
1010 REM******MULTIPLE COLUMN CHECK REGISTER BALANCING PROGRAM
1020 REM      VERSION 1.0   09-16-77
1030 REM      WRITTEN BY CHARLIE PACK
1040 REM                25470 ELENA ROAD
1050 REM                LOS ALTOS HILLS, CA.  94022
1060 REM                PHONE (415) 941-0495 EVENINGS
1070 REM******THIS PROGRAM MAY BE COPIED AND/OR
1080 REM      USED BY ANYONE WHO WANTS IT.
1100 REM
1110 REM******COLUMN SUBSCRIPT VARIABLES
1120 REM
1130 REM  C       COLUMN SUBSCRIPT
1140 REM  C1,C2   RANGE OF COLUMNS TO BE PROCESSED BY A SUBROUTINE
1150 REM          (C1 LOWEST, C2 HIGHEST)
1160 REM  C9      HIGHEST ALLOWABLE COLUMN NO. (CAN BE CHANGED)
1200 REM
1210 REM      THE FOLLOWING VARIABLES ASSIGN COLUMNS TO THE
1220 REM      INDICATED FUNCTIONS.
1230 REM
1240 REM  A1,A2   LOWEST,HIGHEST ASSET OR LIABILITY ACCOUNT COLUMN
1260 REM  B1      BANK CHECKING ACCT. WITHDRAWALS COLUMN
1270 REM  B2      BANK CHECKING ACCT. DEPOSITS COLUMN
1280 REM          NOTE: COLUMNS B1 AND B2 MUST BE INCLUDED IN
1290 REM                THE RANGE A1,A2.
1310 REM  E1,E2   LOWEST,HIGHEST EXPENSE ACCOUNT COLUMN
1320 REM  I1,I2   LOWEST,HIGHEST INCOME ACCOUNT COLUMN
1330 REM  X1      TRANSFER I.D. (CATEGORY OR ACCOUNT NO.) COLUMN
1340 REM  X2      TRANSFER DOLLAR AMOUNT COLUMN
1350 REM  X3      TRANSACTION I.D. (DATE OR LINE NO.) COLUMN
1400 REM
1410 REM******ROW SUBSCRIPT VARIABLES
1420 REM
1430 REM  R       ROW SUBSCRIPT
1440 REM  R1      ASSET ACCOUNT BEGINNING BALANCE ROW
1450 REM  R2      BUDGET AMOUNT ROW
1490 REM  R9      HIGHEST ALLOWABLE ROW NO. (CAN BE CHANGED)
1500 REM
1510 REM******VARIABLES USED IN FORMATTED PRINT ROUTINES
1520 REM
1530 REM  L       LENGTH OF PRINT FIELD, INCLUDING LEADING SPACES
1540 REM  L9      TEMPORARY LENGTH VALUE
1550 REM  M       NO. OF PLACES BEYOND DECIMAL POINT (0 OR 2)
1560 REM  V,V$    NUMERIC VALUE TO BE PRINTED
1570 REM  V1,V1$  INTEGER OF VALUE TO BE PRINTED
1580 REM  V2,V2$  FRACTION OF VALUE TO BE PRINTED
1590 REM  V9$     TEMPORARY STRING VARIABLE
1600 REM
1610 REM******ACCUMULATORS
1620 REM
1630 REM  T    TEMPORARY ACCUMULATOR
1640 REM  T4   CHECKING ACCOUNT RUNNING BALANCE
1650 REM  T5   TOTAL OF ALL ROWS
1660 REM  T6   BEGINNING NET CASH BALANCE
1670 REM  T7   NET CASH FLOW
1680 REM  T8   ENDING NET CASH BALANCE
1700 REM
1710 REM  P1,P2,P3   VALUES TO BE PRINTED
1720 REM  T1,T2,T3   COLUMN SUB-TOTAL ACCUMULATORS
1730 REM  G1,G2,G3   COLUMN GRAND TOTAL ACCUMULATORS
1740 REM  P1,G1,T1   COLUMN BALANCE
1750 REM  P2,G2,T2   BUDGET AMOUNT
1760 REM  P3,G3,T3   VARIANCE
2000 REM
```

```
2010 REM******OTHER VARIABLES
2020 REM
2030 REM  D     DATA ARRAY OF DIMENSION D(C9,R9)
2040 REM  J     TEMPORARY SUBSCRIPT VALUE
2050 REM  K     LENGTH OF COMMAND STRING
2060 REM  N     COUNT OF NON-ZERO ELEMENTS
2110 REM  F$    COMMAND FUNCTION CODES
2120 REM  H$    TEMPORARY STRING HOLD AREA
2130 REM  T$    COLUMN TITLE STRING ARRAY OF DIMENSION T$(C9)
2140 REM  S$    COMMAND STRING
2150 REM  S1$,S2$,...,S5$  COMMAND CHARACTERS 1,2,...,5
5000 REM
5010 REM******PROGRAM INITIALIZATION
5020 REM
5030 CLEAR 100
5040 C9=19:R9=29
5050 E1=C9+1:E2=C9+1:I1=C9+1:I2=C9+1
5060 X1=C9+1:X2=C9+1:X3=C9+1
5070 A1=C9+1:A2=C9+1:B1=C9+1:B2=C9+1
5080 R1=R9+1:R2=R9+1
5090 DIM D(C9,R9)
5110 DIM T$(C9)
5120 REM******INSERT COLUMN NAMES HERE
5390 F$="BACLRTIS0"
5400 REM
5410 REM******MAIN PROGRAM LOOP BEGINS HERE
5420 REM
5430 INPUT"COMMAND (B,A,C,L,R,I,T,S,0)";S$
5440 K=LEN(S$):FOR C=1 TO LEN(F$)
5450 IF LEFT$(S$,1)=MID$(F$,C,1) THEN 5470
5460 NEXT C:GOSUB 9910:GOTO 5430
5470 ON C GOSUB 5740,5530,5930,6110,6930,7210,7410,5490,9999
5480 GOTO 5430
5490 STOP:RETURN
5500 REM
5510 REM******ACCUMULATE IN SPECIFIED COLUMN/ROW ELEMENT
5520 REM
5530 GOSUB 9730:IF C<0 THEN 5690
5540 IF C=X1 OR C=X3 THEN 9950
5550 GOSUB 9830:IF R<0 THEN 5690
5560 IF R<>R1 OR A2>C9 OR A1>A2 THEN 5610
5570 IF C<A1 OR C>A2 THEN 9970
5610 PRINT"COL.";C;"ROW";R;"ACCUMULATE";
5620 INPUT I:IF I=0 THEN 5640
5630 D(C,R)=D(C,R)+I:GOTO 5610
5640 I=D(C,R):I=INT(ABS(I)*100+.5)/100
5650 IF D(C,R)<0 THEN I=-(I)
5660 D(C,R)=I
5690 RETURN
5700 REM
5710 REM******ENTER IN CONSECUTIVE ROWS OF SPECIFIED COLUMN
5720 REM
5730 GOSUB 9730:IF C<0 THEN 5890
5740 GOSUB 9830:IF R<0 THEN 5890
5750 IF R<>R1 OR A2>C9 OR A1>A2 THEN 5770
5760 IF C<A1 OR C>A2 THEN 9970
5770 PRINT"COL.";C;"ROW";R;"ENTRY";
5780 INPUT I:IF I=0 THEN 5890
5810 IF C<>X1 AND C<>X3 THEN 5830
5820 D(C,R)=INT(ABS(I)):GOTO 5840
5830 D(C,R)=INT(ABS(I)*100+.5)/100
5840 IF I<0 THEN D(C,R)=-(D(C,R))
5850 R=R+1:IF R>R9 THEN 5890
5860 IF R<>R1 OR A2>C9 OR A1>A2 THEN 5770
5870 IF C<A1 OR C>A2 THEN 5850
5880 GOTO 5770
5890 RETURN
5900 REM
5910 REM******SET SPECIFIED ELEMENT OR COLUMN TO ZEROES
5920 REM
5930 GOSUB 9730:IF C<0 THEN 6090
5940 IF R=3 THEN 6010
5950 GOSUB 9830:IF R<0 THEN 6090
5960 D(C,R)=0:GOTO 6090
6010 FOR R=0 TO R9
6020 IF R=R1 OR R=R2 THEN 6040
6030 D(C,R)=0
6040 NEXT R
6090 RETURN
6100 REM
6110 REM******COLUMN LIST ROUTINES
6120 REM
6130 GOSUB 9730:IF C<0 THEN 6490
6140 IF R=3 THEN 6230
6150 GOSUB 9830:IF R<0 THEN 6490
6160 REM******LIST SPECIFIED COLUMN/ROW ELEMENT
6170 V=D(C,R):L=10:M=2:IF C=X1 OR C=X3 THEN M=0
6180 GOSUB 9030:PRINT:GOTO 6490
6210 REM******LIST ALL NON-ZERO ELEMENTS IN SPECIFIED COLUMN
6220 PRINT"COL.#--ROW #--ENTRY ----$ AMOUNT"
6240 T=0:FOR R=0 TO R9
6250 IF D(C,R)=0 OR R=R1 THEN 6300
6260 V=D(C,R):L=5:M=0:GOSUB 9030:V=T:GOSUB 9030
6270 IF X1>C9 THEN 6310
6280 V=D(X1,R):IF V>0 THEN L=9:GOSUB 9030
6310 IF X3>C9 THEN 6350
6320 V=D(X3,R):IF V>0 THEN L=7:GOSUB 9030
6330 IF C=X1 OR C=X3 THEN PRINT:GOTO 6490
6360 PRINT TAB(33):V=D(C,R):L=11:M=2:GOSUB 9030
6370 PRINT:T=T+D(C,R)
6390 NEXT R:IF C=X1 OR C=X3 THEN 6490
6410 PRINT TAB(16)"COLUMN TOTAL";
6420 PRINT TAB(33):V=T:L=11:M=2:GOSUB 9030:PRINT
6490 RETURN
6500 REM
6510 REM******ROW LIST ROUTINES
6520 REM
6530 IF R=1 THEN 6840
6540 IF R<>3 THEN 6990
6550 H$=S$:S$=LEFT$(H$,1)+"  "+RIGHT$(H$,2)
6560 K=LEN(S$):GOSUB 9830:IF R<0 THEN 7190
```

NATIONAL COMPUTER CLUB CONGRESS

Received: 77 July

Over forty-five computer club leaders and representatives, from about thirty computer clubs across the nation, met in Dallas for the first National Computer Club Congress. The Congress was held in conjunction with the 1977 National Computer Conference, which included a major segment on personal computing. Dr. Portia Isaacson attended one of the sessions of the Congress and received enthusiastic applause in recognition of her support as Chairman of the 1977 NCC.

The purpose of the Congress was to exchange ideas and discuss issues related to club activities and programs. The central focus of the group was directed toward the question of a national association. Pre-NCC planning sessions were held this spring at computerfests in San Francisco, Trenton, and Cleveland. A survey of club leaders was conducted by mail to develop information on prospective purposes and structures which might be appropriate for a national association. The results of that survey were reported to the Congress at the beginning of the first session, and ensuing discussion revealed widespread agreement on many potential purposes. There was, however, a divergence of opinions on how these purposes should be accomplished and on the structure of any national organization.

A committee to define and establish a national amateur computing association was formed June 15 the second session of the Club Congress. At its initial meeting held after adjournment of the Congress, the group decided to be known as the Committee. The Committee includes computer clubs members from the Northeast, Atlantic seaboard, south, midwest, southwest and west.

Throughout the next few months the Committee will be organizing for discussion of the services that should be provided by a national society for personal computing, as well as how it should be set up. These sessions will be held during functions attended by large numbers of amateur computer users. They will build on forums already held in San Francisco, Trenton, and Cleveland.

Members of the Committee will meet at every opportunity under a rotating presiding officer. Those unable to be present will be asked to submit comments on the subjects to be discussed. Results of Committee meetings will be publicized throughout the amateur computing community.

Further forums for discussion of issues raised by the Committee have been offered by *SCCS Interface* and *Dr. Dobbs Journal*.

Initial financial support of the Committee has been assured by pledges of support from a few of the large regional organizations of computer clubs.

The Committee is also soliciting opinions in personal contacts by individual members. The group may be reached through its secretary, Jim White, at 1202 River View Lane, Watertown, WI 53094, or by calling Richard Kuzmack, (703) 821-2873.

Also resulting from the Computer Club Congress are plans for the Southwest Federation of Computing Clubs. These were arrived at by members of nine clubs in Texas, Oklahoma, Arizona, and Louisiana with a total membership of about 700. The major purpose of the Federation is seen as communication among member clubs. Responsibility for Federation activities will be rotated among the clubs.

```
6610 REM*****LIST ALL NON-ZERO ELEMENTS WITHIN SPECIFIED ROW
6620 REM*****  (EXCEPT XFR ID AND TXN ID)
6630 PRINT"COL.#--ROW #----$ AMOUNT"
6640 T=0:FOR C=0 TO C9
6650 IV D(C,R)=0 OR C=X1 OR C=X3 THEN 6750
6710 V=C:L=5:M=0:GOSUB 9030:V=R:L=7:GOSUB 9030
6720 V=D(C,R):L=13:M=2:GOSUB 9030:PRINT:T=T+D(C,R)
6750 NEXT C:PRINT" ROW BALANCE";
6760 V=T:L=13:M=2:GOSUB 9030:PRINT:GOTO 7190
6810 REM*****LIST TOTALS OF ALL ROWS WITH ANY NON-ZERO ELEMENTS
6820 REM*****ALSO LISTS CHECKING ACCOUNT RUNNING BALANCE
6840 PRINT"ROW #--XFR ID--TXN ID--ROW TOTALS---CH ACC BAL"
6850 IF R1>R9 OR B1>C9 OR B2>C9 THEN 6910
6860 V=R1:L=5:M=0:GOSUB 9030:T4=D(B2,R1)
6880 V=T4:L=42:M=2:GOSUB 9030:PRINT
6910 T5=0:FOR R=0 TO R9:IF R=R1 THEN 7140
6920 N=0:T=0:FOR C=0 TO C9
6930 IF D(C,R)=0 OR C=X1 OR C=X3 THEN 7010
6960 T=T+D(C,R):N=N+1
7010 NEXT C:IF N=0 THEN 7140
7020 V=R:L=5:M=0:GOSUB 9030
7030 IF X1>C9 THEN 7050
7040 V=D(X1,R):IF V>0 THEN L=9:GOSUB 9030
7050 IF X3>C9 THEN 7070
7060 V=D(X3,R):IF V>0 THEN L=7:PRINT TAB(15);:GOSUB 9030
7070 PRINT TAB(23);:V=T:L=11:M=2:GOSUB 9030:T5=T5+T
7080 IF R1>R9 OR B1>C9 OR B2>C9 OR R=R2 THEN 7130
7090 IF D(B1,R)=0 AND D(B2,R)=0 THEN 7130
7110 T4=T4+D(B1,R)+D(B2,R):V=T4:L=13:GOSUB 9030
7130 PRINT
7140 NEXT R
7150 PRINT TAB(7)"TOTAL ALL ROWS";
7160 V=T5:L=13:M=2:GOSUB 9030:PRINT
7190 RETURN
7200 REM
7210 REM*****DISPLAY TRIAL BALANCE
7220 REM
7230 PRINT"COL.#---NAME---------ACCT BAL.";
7240 PRINT"------BUDGET---COL. TOTALS"
7250 C1=0:C2=C9:GOSUB 8640
7260 PRINT"*****GRAND TOTALS";
7270 P1=T1:P2=T2:P3=T3:GOSUB 8910
7390 RETURN
7400 REM
7410 REM*****DISPLAY INCOME STATEMENT
7420 REM
7430 PRINT"COL.#---NAME----------AMOUNT-";
7440 PRINT"------BUDGET-----VARIANCE"
7450 IF I2>C9 OR I1>I2 THEN T1=0:T2=0:T3=0:GOTO 7470
7460 C1=I1:C2=I2:GOSUB 8640
7470 PRINT"*****TOTAL INCOME";
7480 P1=T1:P2=T2:P3=T3:GOSUB 8910
7490 G1=T1:G2=T2:G3=T3
7510 IF E2>C9 OR E1>E2 THEN T1=0:T2=0:T3=0:GOTO 7530
7520 C1=E1:C2=E2:GOSUB 8640
7530 PRINT"***TOTAL EXPENSES";
7540 P1=T1:P2=T2:P3=T3:GOSUB 8910
7550 G1=G1+T1:G2=G2+T2:G3=G3+T3
7580 PRINT"**SURPLUS/DEFICIT";
7590 P1=G1:P2=G2:P3=G3:GOSUB 8910
7610 IF X2>C9 THEN 7710
7620 C1=X2:C2=X2:GOSUB 8640
7630 G1=G1+T1:G2=G2+T2:G3=G3+T3
7710 PRINT"****NET CASH FLOW";
7720 P1=G1:P2=G2:P3=G3:GOSUB 8910
7810 IF A2>C9 OR A1>A2 THEN 8090
7820 T6=0:IF R1>R9 THEN 7910
7850 FOR C=A1 TO A2:T6=T6+D(C,R1):NEXT C
7910 T7=0:FOR C=A1 TO A2
7920 N=0:T=0:FOR R=0 TO R9
7930 IF D(C,R)=0 THEN 7960
7940 IF R=R2 OR R=R1 THEN 7960
7950 N=N+1:T=T+D(C,R)
7960 NEXT R:IF N>0 THEN T7=T7+T
7990 NEXT C
8010 T8=T7+T6:PRINT"NET CASH BALANCE (ENDING)";
8030 V=T8:L=13:M=2:GOSUB 9030:PRINT
8040 PRINT"NET CASH BALANCE (BEGIN.)";
8050 V=T6:GOSUB 9030
8060 PRINT"   NET CASH FLOW";
8070 V=T7:GOSUB 9030:PRINT
8090 RETURN
8600 REM
8610 REM*****OUTPUT ROUTINES FOR TRIAL BALANCE OR INCOME STMT
8630 REM
8640 T1=0:T2=0:T3=0
8650 FOR C=C1 TO C2
8660 IF C=X1 OR C=X3 THEN 8810
8670 N=0:T=0:FOR R=0 TO R9
8680 IF D(C,R)=0 OR R=R1 THEN 8710
8690 N=N+1:IF R<>R2 THEN T=T+D(C,R)
8710 NEXT R:IF N=0 THEN 8810
8720 P1=T:T1=T1+P1
8730 IF R2>R9 THEN 8780
8750 P2=D(C,R2):T2=T2+P2:P3=P1+P2:T3=T3+P3
8780 V=C:L=5:M=0:GOSUB 9030
8790 PRINT TAB(7) T$(C);:GOSUB 8910
8810 NEXT C:RETURN
8910 PRINT TAB(19);:V=P1:L=11:M=2:GOSUB 9030
8920 IF R2>R9 THEN 8990
8950 V=P2:L=13:GOSUB 9030:V=P3:L=13:GOSUB 9030
8990 PRINT:RETURN
9000 REM
9010 REM*****FORMATTED PRINT ROUTINE
9020 REM
9030 IF M<>0 AND M<>2 THEN M=0
9040 L=INT(ABS(L)):IF L<2 OR L>72 THEN L=14
9050 V1=INT(ABS(V)):IF M=0 THEN V2=0:GOTO 9080
9060 V2=INT((ABS(V)-V1)*100+.5)/100
9070 IF V2>=1 THEN V2=0:V1=V1+1
9080 IF V1=0 AND M>0 THEN V1$="":GOTO 9210
9110 V9$=STR$(V1):IF V1<1000 OR M=0 THEN 9150
9120 IF V1<1E+06 THEN 9140
9130 V$=STR$(V):GOTO 9350
9140 V1$=MID$(V9$,2,LEN(V9$)-4)+","+RIGHT$(V9$,3):GOTO 9210
9150 V1$=MID$(V9$,2,LEN(V9$)-1)
9210 IF M=0 THEN V2$="":GOTO 9310
9220 IF V2=0 THEN V2$=".00":GOTO 9310
9230 V9$=STR$(V2):IF LEN(V9$)>3 THEN 9250
9240 V2$=MID$(V9$,2,2)+"0":GOTO 9310
9250 V2$=RIGHT$(V9$,LEN(V9$)-1)
9310 V9$=" ":IF V<=-.005 THEN V9$="-"
9320 V$=V1$+V2$+V9$
9350 L9=LEN(V$)
9360 IF L>L9 THEN PRINT" ";:L9=L9+1:GOTO 9360
9380 PRINT V$;:RETURN
9700 REM
9710 REM*****CHECK FOR VALID COLUMN #
9730 IF K<3 THEN C=-1:GOTO 9920
9740 S2$=MID$(S$,2,1):IF S2$<"0" OR S2$>"9" THEN C=-1:GOTO 9930
9750 S3$=MID$(S$,3,1):IF S3$<"0" OR S3$>"9" THEN C=-1:GOTO 9930
9760 C=VAL(MID$(S$,2,2)):IF C<=C9 THEN 9990
9770 C=-1:GOTO 9940
9810 REM*****CHECK FOR VALID ROW #
9830 IF K<5 THEN R=-1:GOTO 9920
9840 S4$=MID$(S$,4,1):IF S4$<"0" OR S4$>"9" THEN R=-1:GOTO 9930
9850 S5$=MID$(S$,5,1):IF S5$<"0" OR S5$>"9" THEN R=-1:GOTO 9930
9860 R=VAL(MID$(S$,4,2)):IF R<=R9 THEN 9990
9870 R=-1:GOTO 9960
9910 PRINT"INVALID COMMAND FUNCTION":GOTO 9990
9920 PRINT"WRONG # OF CHARACTERS TYPED":GOTO 9990
9930 PRINT"NON-NUMERIC ROW OR COLUMN #":GOTO 9990
9940 PRINT"COLUMN # OUT OF RANGE":GOTO 9990
9950 PRINT"ILLEGAL COLUMN # FOR THIS FUNCTION":GOTO 9990
9960 PRINT"ROW # OUT OF RANGE":GOTO 9990
9970 PRINT"ILLEGAL ROW # FOR THIS FUNCTION"
9990 RETURN
9999 END:GOTO 9999
OK
```

SOFTWARE & CORRECTION

Dear Jim, *Received: 77 Oct 28*

My 650X subroutine OPLEGL, published in the Aug. '77 *DDJ* let one illegal opcode (9E) slip through its net. So there 105 illegals, not 104. I am therefore submitting a rewrite that fixes this error and also spruces things up some. So that OPLEGL will be more than an academic exercise in sorting, I have added a primitive debugger program that uses both OPLEGL and BYTNUM. I am also submitting a new byte-count-computing subroutine (NUMBYT) that uses utterly different logic from any earlier one. It lacks the cool precision of BYTNUM, but has a fiery eccentricity and some novel ideas that will appeal to some users, and comes closer to what I feel should be the ultimate goal of subroutine design: to return the maximum possible amount of information to the main program, whether or not this is needed by the program it was written to serve.

Sincerely,

H.T. Gordon
Agricultural Experiment Stn.
University of California
Berkeley, CA 94720
©1977 by H.T. Gordon

P.S. While I wish to make the software in this paper freely available without restriction to individual users, I retain full copyright for any commercial use. My motivation is not greed (my fee would be trivial and usually waived) but curiosity. Some time ago I sent a binary-to-decimal conversion program to MOS Technology, stating that it was to be in the public domain. Recently I asked them whether they had used it in their PET. The reply was that they did not plan to release any information on its programs. Puzzling, since anyone who buys a PET will be able to read its ROM. I am an admirer of their 6502 and KIM designs, and I sympathize with their desire to avoid hassles. Still, this gave me food for thought. Software is a trickier thing than the books and magazine articles for which copyright law was written, and in which changing a few words won't affect value. A seemingly minor software revision may so enhance value that its original will (and should) vanish forever. Like it or not, every programmer is part of a collective mind, and progress demands that he educate and be educated by others. There is no precise answer to the question: what is the value of a piece of software, and who owns what part of the value? In the software-cost controversy argued in *DDJ* by Tom Pittman and others, I strongly favor keeping all costs at the bare minimum and legal conflicts at zero, since I am primarily a user. Program costs are not easy to calculate. In my present "package", OPLEGL took a lot of time, BYTNUM much less because I was only modifuing Larry Fish's logic. NUMBYT was even easier because I knew what needed to be done, and SIMBUG was child's play. Still, all these things were simultaneously working themselves out in my mind, and without the basic (originally more grandiose) concept of a scanning-debugger I would never have bothered with OPLEGL. This concept was inspired by Jim Butterfield's screening-out of the 64 "easy" illegals in his relocator program. I was receptive to this because of earlier problems caused by an illegal opcode. So in a way SIMBUG is a costly program.

The real value of anything is not in its cost, but in its utility. It may cost a lot to produce a white elephant, but the result is still worthless. In our social system, utility tends to be measured in the marketplace, in dollars. This has worked well for hardware, and for any software that can be inextricably linked to hardware. Unattached software can be valued in dollars only to the extent that users can be compelled either to buy it or do without. But it is so readily diffusible and copiable that many users will not buy it except at a price little above the cost (in money and time) of copying. Copyrighted printed matter is now photocopied illegally by individuals with complete impunity, because enforcement is not practical. Where law fails, we cannot expect too much of ethics. So the utility (in the economic sense) of even the most useful "software" is low. One could say the same of rainfall and sunlight!

OPLEGL CORRECTION, AND A 6502 SCANNING-DEBUGGER

H.T. Gordon

The following corrected version of subroutine OPLEGL adds 4 bytes to screen out opcode 9E but saves 5 bytes by improved logic and structure, so is down to 66 bytes total. The task of saving the accumulator is left to the main program. The accumulator will return modified by from 2 to 6 successive logical-shift-rights. Otherwise, only the status register is affected. The Z flag is set only by A2 and the 5 legals of type X_1(4, A, C); these also set the C flag. The C flag is set by X91, 5, 9, D)≥8D, by X(6,E)≥A6, by $X_0$0≥90, and by all X8, but cleared by all others. The N flag is always the complement of the C flag. The V flag is not affected. This status information may or may not be useful to the main program (who can tell?) but one ought to know that it exists.

Coding for OPLEGL

```
Ø27Ø 4A     OPLEGL LSR A (bit Ø to C)
Ø271 9Ø Ø8         BCC TYPEØ2 (even #s)
Ø273 4A            LSR A (bit 1 to C)
Ø274 BØ 17         BCS ILLEGA (all
                     X(3,7,B,F))
Ø276 C9 22         CMP #$22 (LSRed $89)
Ø278 FØ 13         BEQ ILLEGA (89 illeg.)
Ø27A 6Ø            RTS (other X(1,5,9,D))
Ø27B 4A     TYPEØ2 LSR A (bit 1 to C)
Ø27C 9Ø 1Ø         BCC TYPEØ (X(Ø,4,8,C))
Ø27E 4A            LSR A (bit 2 to C)
Ø27F 9Ø Ø5         BCC TYPE2A (X(2,A))
Ø281 C9 13         CMP #$13 (LSRed $9E))
Ø283 FØ Ø8         BEQ ILLEGA (9E illeg.)
Ø285 6Ø     LEGALA RTS  (other X(6,E))
Ø286 4A     TYPE2A LSR A (bit 3 to C)
Ø287 BØ 2Ø         BCS TYP4AC (all XA))
Ø289 C9 ØA         CMP #$ØA (LSRed $A2)
Ø28B FØ F8         BEQ LEGALA (A2 legal)
Ø28D ØØ     ILLEGA BRK (other X2 illeg.)
Ø28E 4A      TYPEØ LSR A (bit 2 to C)
Ø28F BØ Ø8         BCS TYPE4C (X(4,C))
Ø291 4A            LSR A (bit 3 to C)
Ø292 BØ Ø4         BCS ALLOK (all X8)
Ø294 C9 Ø8         CMP #$Ø8 (LSRed $8Ø)
Ø296 FØ ØC         BEQ NOTLEG (8Ø illeg.)
Ø298 6Ø      ALLOK RTS  (other XØ legal)
Ø299 4A     TYPE4C LSR A (bit 3 to C)
Ø29A FØ Ø8         BEQ NOTLEG ($Ø4, $ØC,
                     LSRed to ØØ)
Ø29C BØ Ø7         BCS TYPEC (other XC)
Ø29E 29 ØD         AND #$ØD (tests LSRed
                     $44, $64)
Ø2AØ C9 Ø4         CMP #$Ø4 (new = Ø4)
Ø2A2 DØ Ø5         BNE TYP AC (other X4)
Ø2A4 ØØ     NOTLEG BRK  (44, 64 illeg.)
Ø2A5 C9 Ø9  TYPEC  CMP #$Ø9 (LSRed $9C)
Ø2A7 FØ FB         BEQ NOTLEG (9C illeg.)
Ø2A9 4A     TYP4AC LSR A (bit 4 to C)
Ø2AA 9Ø Ø5         BCC LEGIT (X• legals)

Ø2AC 4A            LSR A (bit 5 to C)
Ø2AD C9 Ø2         CMP #$Ø2 (LSRed X=9 or
                     B new = Ø2)
Ø2AF DØ F3         BNE NOTLEG (≠, illeg.)
Ø2B1 6Ø      LEGIT RTS (X1 legals)
```

It is natural that 6502 programmers would be more concerned about legality than the 8080 workers (who have only a dozen illegal opcodes to worry about). In any program, a wrong-bit bug can arise by miskeying or accidental bit-change during transfer to or from storage. In an 8080 program, such an error is likely to change an opcode to a different (wrong but legal) one. With 105 illegals there is a higher probability that a legal opcode will be altered to an illegal; furthermore, if a branch or jump should be misdirected to an operand location (by miscalculation, miskeying, or bit-dropping), the operand may not be a legal opcode. Even if it is, I shall in a subsequent paragraph indicate a way of proving that it is wrong. The

underlying principle of a scanning debugger is that the inherent structural rigidities in a program make possible the automatic detection of certain errors, without any execution of the program. Execution of a program bug can cause a lot of damage in RAM – even *creating* bugs where there were none before – so that even an imperfect pre-execution debugging scan may well be worthwhile. The "opcode-type-counting" aspect of my program SIMBUG could be used for *any* microprocessor. The "legality-testing" aspect is especially valuable for the 650X. The 650X control unit, that knows all about the operands required by opcodes, knows nothing of legality and will cheerfully execute 93 of the illegals and be stopped cold by the other 12 (all type X2 except 82, A2, C2, and E2, of which only A2 is legal but all 4 are executed). Either way the result can be unpleasant, sometimes a subtle and far-reaching bug. It was such a bug that first aroused my interest in the executable illegals.

As an introduction to SIMBUG, I shall try to make the concept more concrete by analysing a short program segment, written in a schematic way as

$$L_1\ O_1\ O_2\ K_1\ O_3\ J_1\ K_2\ O_4\ B_1\ O_5\ K_3\ O_6$$

"O" indicates an operand, "J" a 1-byte opcode, "K" a 2-byte, and "L" a 3-byte. "B" indicates a branch opcode, a subclass of 2-bytes that is easy to detect (all 8 branches are type X_10) and has a very high usage frequency (10% or more of all the opcodes in most programs). All 4 types are *counted* by SIMBUG.

The debugger is initialized to pick up L_1. It calls OPLEGL to test its legality. If legal, it calls BYTNUM to determine where the next opcode is. If there are no illegality errors (that cause a program BREAK) it skips from code to code until it eventually encounters an illegal. No branches or jumps are taken (although more elaborate programming would make this possible). Operation can be terminated at any desired point by deliberately replacing a legal opcode by an illegal.

Let us now examine the effect of an opcode error (however caused) that is an alteration of a correct code to an incorrect but legal one, "W". If W specifies the *same* number of bytes, the error is undetectable. If W specifies 1 or 2 bytes, the debugger will pick up either O_1 or O_2 as the next opcode. Only if O_1 is a legal 2-byte, or O_2 a legal 1-byte, will it pick up K_1 as the third opcode and be back "on track". Such chance "error compensation" will sometimes occur. Note, however, that one 3-byte code has been converted into two (a 1-byte plus a 2-byte). In the example, the counts of 1-bytes and 2-bytes would be too high by 1, and the count of 3-bytes too low by 1. The programmer is alerted to search for wrong 3-byte. One can extend this analysis to every possible W-type error and see that it is highly improbable (though not impossible) that chance compensation will restore the correct counts. The difficulties are obvious: the programmer must *know* the correct counts, and in a long program the search for the opcode error will be tedious. The latter can be minimized if scanning is done in "chunks" of 256 bytes. This could be easily implemented in SIMBUG but I have not bothered, because this debugging concept is not yet (and may never be!) an accepted one. Improvement can wait until the simple version has (or has not) proved itself useful.

The only operand errors that a somewhat more complex version of SIMBUG could detect are the branch offsets, and then only if the error caused branching to an operand. The debugger could calculate the location to which the offset directed the operation, and start a scan from that address in the hope that mistracking would not be compensated but cause it to strike an illegal. A much more positive approach would be for a first pass of the debugger to record the address of every presumptive opcode in its search area, and in a second pass calculate the branch locations and compare them with the list of opcode addresses. Failure to find a match would be a guarantee of error, most probably in the offset. The gain in debugging power might not be worth the much higher memory cost. Only about 10% of all operands will be branch offsets, and the chance of an error detection would be roughly 50-50. It seems to be a characteristic of all debuggers (including human ones) that effectiveness is subject to severely diminishing returns, while costs increase exponentially.

Coding for SIMBUG

```
Ø2B3 AØ ØØ     SIMBUG LDY #Ø
Ø2B5 A2 Ø7            LDX #7
Ø2B7 94 Ø2     CLERIT STY CNTLO,X
Ø2B9 CA               DEX
Ø2BA 1Ø FB            BPL CLERIT
Ø2BC B1 ØØ     LOADIT LDA (BASAD),Y
Ø2BE 2Ø 7Ø Ø2         JSR OPLEGL
Ø2C1 B1 ØØ            LDA (BASAD),Y
Ø2C3 2Ø 1Ø Ø2         JSR BYTNUM
Ø2C6 F6 Ø2       COUNT INC CNTLO,X
Ø2C8 DØ Ø2            BNE TSTBRN
Ø2CA F6 Ø6            INC CNTHI,X
Ø2CC 29 1F     TSTBRN AND #$1F
Ø2CE C9 1Ø            CMP #$1Ø
Ø2DØ DØ Ø6            BNE INCAD
Ø2D2 E6 Ø2            INC CNTLO
Ø2D4 DØ Ø2            BNE INCAD
Ø2D6 E6 Ø6            INC CNTHI
Ø2D8 E6 ØØ      INCAD INC BASAD
Ø2DA DØ Ø2            BNE NOPINC
Ø2DC E6 Ø1            INC BASAD+1
Ø2DE CA        NOPINC DEX
Ø2DF DØ F7            BNE INCAD
Ø2E1 FØ D9            BEQ LOADIT

 Zero-page locations affected:
ØØ BASAD   low scan-start address
Ø1 BASAD+1  hi    "     "      "
Ø2 CNTLO   low branch-opcode count
Ø3 CNTLO+1  "  1-byte-  "      "
Ø4 CNTLO+2  "  2-   "    "      "
Ø5 CNTLO+3  "  3-   "    "      "
Ø6 CNTHI    hi branch-opcode    "
Ø7 CNTHI+1  "  1-byte-  "      "
Ø8 CNTHI+2  "  2-byte-  "      "
Ø9 CNTHI+3  "  3-   "    "      "
```

As befits a moronic main program, both logic and handling of SIMBUG are straightforward. The user inserts an illegal code at the point he wishes to scan to stop, keys in the start address in the BASAD zero-page locations, and runs SIMBUG. Almost instantly there will be a BREAK to whatever program-interrupt routine the user has decided on (this is system-dependent). The user checks the current address in BASAD. If it is a program-opcode location, this contains an illegal. This is fixed and SIMBUG is started again (the address in BASAD is right). If the illegal was at an operand location, derailing occurred at an earlier point, probably by a wrong but legal opcode calling for the wrong number of operand bytes. There is also a (presumably faint) possibility that a correct opcode was allotted the wrong number of operand bytes. This is

hunted down and fixed. To play safe, it is a good idea to rescan from the start address. Eventually the break will occur at the pre-set terminator illegal. At this point, a track of legals exists in the program, but possibly not the intended one. A user who has not yet had his fingers burned may decide to execute the program, without bothering to check the opcode-type counts. Those who have previously gambled and lost will compare the counts from their listing with the SIMBUG counts, remembering that the computer is always right. If they agree, SIMBUG can do no more. True gamblers can strip SIMBUG of its 25-byte counting logic (the CLERIT and COUNT segments) and cut it to a super-moronic 23 bytes. The total byte-count of this minimal operation, including the two subroutines, is then 124. A less drastic stripping would eliminate only the branch-testing segment TSTBRN, saving 12 bytes.

SIMBUG was tested on the KIM-1 monitor programs. In the section starting at 1800, it was stopped by the illegal 0F at 1871. This is a data word, not an instruction. The counts of opcode-types checked perfectly. Starting a new scan on the program opcode at 1873, SIMBUG stopped at the data word 6B at 1BFA. It had scanned so many bytes that I did not bother to check the type-counts. A scan from 1C00 was stopped by the data word 4B at 1FDF. The scan had ploughed through a string of data zeroes well beyond the end of actual programming, so the type-counts would not be correct (but what can one expect from a minuscule moron?). Needless to say, when turned on itself and its subroutines, SIMBUG found no flaws!

As what I hope will be my swan song in this software area, I submit a radically different alternative to BYTNUM. Subroutine NUMBYT is fully relocatable, equally byte-efficient, but a bit slower. It includes some tricks that could be useful in other contexts. Users of the 6800 and 650X know that both designs have direct transfers between stack (S), accumulator (A), and status register (P), implemented in a different way: S-A-P in the 6800 but A-S-P in the 650X. The TAP instruction of the 6800 sets all 4 testable flags in P to the pattern of the 4 low-order bits in A (because the P register is organized as xxxxSZOC), and the elaborate conditional-branch instructions allow sophisticated bit-analysis. However, although the pattern is easily stored in S for later re-use (by a PSH ACA), it has to pass through (and so destroy the content of) A in order to move into P.

The 650X needs 2 bytes (PHA, then PLP) to move the A bit-pattern to P, and the 4 testable flags then reflect the status of the 2 highest-order and 2 lowest-order bits (because P is organized as NVxxxxZC). However, the pattern can be stored in S (either by deferring the PLP until it is needed, or by a PLP and PHP for both immediate and later use), and moved to P without affecting A.

The price paid for these powerful flag-setting operations is that non-testable flags are also affected (e.g., the interrupt-inhibit flag). In the 6800, the stack must be reset to the pre-call state (if the trick is used inside a subroutine) so that the RTS can pick up the correct return address. However, the 650X has the unique capability of transferring one byte, stored in the stack by a subroutine, to the main program. If an RTI is used instead of the usual RTS, the stack-stored byte is moved into P before the return address is picked up. The main program can use the status information immediately, and/or store it by a PHP for later use. NUMBYT uses these tricks both in its inner operation and to return much more information in its status register at exit than BYTNUM. Since it destroys the accumulator, the main program has the task of saving and restoring it if necessary. P at exit has 7 of the original bits in A, rotated to 0x765432, where x is always a zero bit and the original bit 1 is lost. The N, Z, and C flags therefore are = bits 0, 3, and 2.

PRAISE FOR JAMES & MICROTRONICS

Dear Jim: *Received: 77 Mar 29*

I'm glad to see you printing the letters from satisfied customers of companies as well as the ones from dissatisfied customers. Obviously it is as important for us to know the good outfits to go to as the bad ones to avoid.

Along those lines, I echo the sentiments of your other readers on James Electronics. If I order something on Monday, I have it by Friday or the following Monday.

Another good outfit is right there at Box 7454 in Menlo Park - - Microtronics. I had despaired of getting turnaround of anything less than a month on 1702A EPROM programming, after experiences with two other outfits, both in the Bay area. Microtronics gets them back in a week, and it only costs $3 (plus postage) for programming from a hex coding form! The best service at the lowest prices. They sell some hardware and software (on PROMS) as well as doing programming, but I haven't checked those products yet.

Jim Wilson — 3250 Wing St. #402
Ketron, Inc. — San Diego, CA. 92110

An optional modification of NUMBYT might be useful. Just before its fifth (PHA) instruction, bit 1 is in the C flag and bit 0 in the N flag. A 5-byte insertion: BCC SKIP, BPL SKIP, BRK will cause a program break for the 64 type X (3,7,B,F) illegals. This does not provide the complete "insurance" against illegality that OPLEGL gives, but costs much less in bytes and time. An added bonus is that if the N flag is set (bit 0 = 1), one knows the opcode was one of 64 "odds" of which only 1 (89) can be illegal, so that one could forego a full legality test at small risk. If the N flag is clear, the probability of an illegal is 20 times greater and a call to OPLEGL might be justifiable.

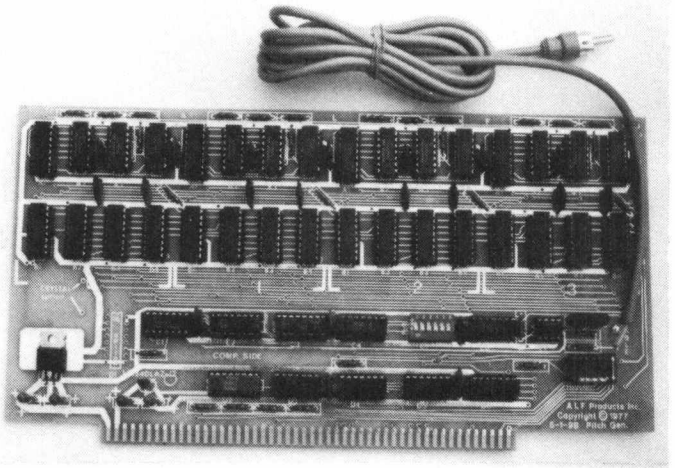
Coding for Subroutine NUMBYT

```
Ø24Ø A2 Ø1 NUMBYT LDX #1 (sets X reg.)
Ø242 18           CLC (clears carry)
Ø243 6A           ROR A (bit Ø to C)
Ø244 6A           ROR A (bit 1 to C)
Ø245 48           PHA (A to stack)
Ø246 6A           ROR A (bit 2 to C)
Ø247 6A           ROR A (bit 3 to C)
Ø248 BØ ØD        BCS NUMHAF (all
                     type X(8-F))
Ø24A 6A           ROR A (bit 4 to C)
Ø24B BØ 13        BCS 2BYTE (X1(Ø-7))
Ø24D C9 Ø4        CMP #4
Ø24F BØ ØF        BCS 2BYTE (all but
                     (6,4,2,Ø)Ø)
Ø251 C9 Ø1        CMP #1
Ø253 FØ ØA        BEQ 3BYTE  ($2Ø)
Ø255 4Ø           RTI ((6,4,Ø)Ø)
Ø256 28     NUMHAF PLP (stack to P)
Ø257 Ø8           PHP (P to stack)
Ø258 BØ Ø5        BCS 3BYTE (bit 2=1,
                     X(C,D,E,F))
Ø25A 1Ø Ø5        BPL 1BYTE (bit Ø=Ø,
                     X(8,A))
Ø25C 6A           ROR A (bit 4 to C)
Ø25D 9Ø Ø1        BCC 2BYTE (bit 4=Ø,
                     X•(9,B))
Ø25F E8      3BYTE INX
Ø26Ø E8      2BYTE INX
Ø261 4Ø      1BYTE RTI
```

ALF MUSIC BOARDS

news release *Received: 77 Jul 13*

The 10-5-9 and 10-5-10 Quad Chromatic Pitch Generator boards are designed to be a low-cost start in computer-controlled music generation. The single board Pitch Generator produces one to four tones simultaneously, two boards can be used to produce eight simultaneous tones in stereo. Each of the four tones are separately controlled and can produce any of 96 tones which form an 8 octave range. This covers the entire standard piano range, plus 8 higher pitches. Special connections allow later expansion with accessory boards to control various sound parameters. Using the optional on-board crystal oscillator or a 2MHz source (external or pin 49 on the S-100 bus) all pitches are within 0.1% of the A-440 Hz standard. The 10-5-9 is S-100 compatible, and the 10-5-10 is compatible with parallel output ports. Kit prices for both versions range from $111 to $159 (depending on the number of simultaneous tones), the assembled price is $185. Oscillator is an additional $16. Available for produce evaluation: data sheet (free), demonstration record ($1), and owner's manual ($3 plus $1 postage). ALF Products Inc.; 128 South Taft; Denver, CO 80228. (303) 234-0871.



6502 GOODIES

Dear Mr. Warren, *Received: 77 Jul 15*

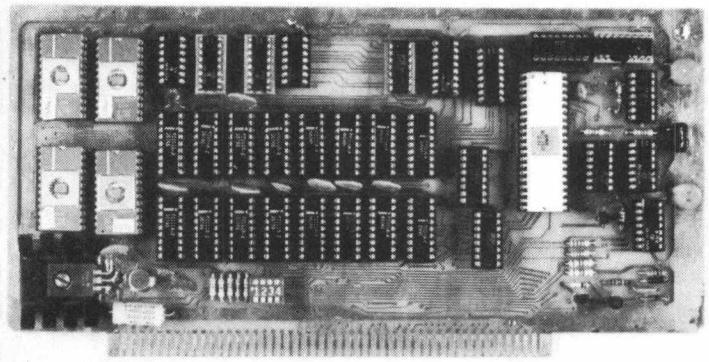
CGRS Microtech has finished a 6502 computer board for the S100 bus. Our board contains the 6502, TTL support circuitry, 2K bytes of RAM, 4K bytes of 2708 ROM, and a connector for our DMA control panel.

We have also developed a T.I.M. I/O card to go with our computer board.

We will soon have available a complete monitor/operating system along with editor/assembler/dissassembler/debugger.

J. Swope
President
CRGR Microtech

P.O. Box 368
Southampton, PA. 18966
(215) 757-0284



ZAPS CASSETTE OPERATING SYSTEM

Received: 77 Oct 17

The ZAPS Cassette Operating System consists of a full Z-80 assembler, text editor, in-memory file system, labeled cassette tape storage system with full cyclic redundancy checks on all tape operations, and a variety of other software utilities. The system is entirely resident in 14K of memory including all buffers and 1K bytes of symbol table space. It will run on most 8080 and Z-80 processors. It is distributed on Tarbell, Digital Group, and TDL standard tapes.

The ZAPS text editor is a full context editor, no line numbers are required. Only the actual data bytes are stored. Editor commands include input, insert, delete, replace, change, global change, up, down, top, bottom, string search, print and print current line number.

The assembler processes the ZILOG recommended form of mnemonics. A wide variety of pseudo operands are available including origin, end, define byte, define word, define string, and equate. Assembler variable names can be any length. All characters in the names are used to determine variable uniqueness. Features that the assembler shares with the rest of ZAPS include English error messages and the ability to accept numerical values in hex, octal, decimal, or binary form.

The system comes with a ninety page user manual, and is available for $60. It can be purchased from local computer stores or directly from Algorithmics, P.O. Box 56, Newton Upper Falls, MA 02164 (617) 965-0545.

3000 HOLE GENERAL PURPOSE PROTOTYPING BOARD FOR $70

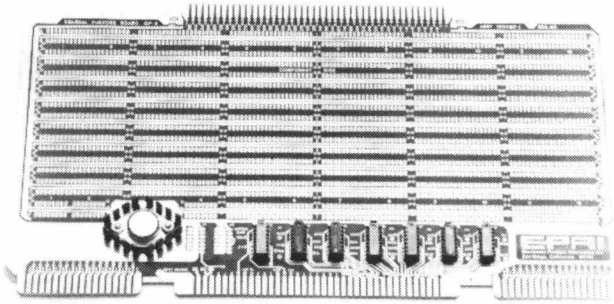
News Release

Electronic Product Associates, Inc., announces the availability of a new General Pupose Prototyping board for use in the Micro-68 Microprocessor systems. The 8" X 14.8" (20.32cm X 37.59cm) GP-2 board is exorcisor bus compatible and has complete bus buffering already established on-board with 8833 driver/receivers. The GP-2 board contains Vcc and ground busing, 3000 holes worth of blank dip patterns for thirty-five 24, 40 or 42 pin dip packages, or 107 14 or 16 pin dip packages.

Price is $170.00 and they are available from stock.

For additional details contact:

Patti Neumann
Vice-President
Electronic Product Associates, Inc.
1157 Vega Street
San Diego CA 92110



dr. dobb's journal of

COMPUTER Calisthenics & Orthodontia

Running Light Without Overbyte

Number 20 **November/December 1977** **Volume 2, Issue 10**

Product Review

NEW VOICE SYNTHESIS & GRAPHICS PRODUCTS – *Kenneth Young* 6

ZAPS Operating System and Assembler for the Z-80 – *Jef Raskin* 29

Product Evaluation

Heathkit's H-8: A Consumer's View – *Paul R. Poduska, Ph.D.* 9

a positive evaluation by a user

Software

For 6502's

A KIM-1 Disassembler – *Theodore E. Bridge* 12

MATHPAC: A Kimath Supplement – *John Eaton* 15

For 6800's

GPM for the 6800 – *Frits van der Wateren* 22

complete documentation & annotated source code

SWTPC Editor Format Update – *Phil Hughes* 30

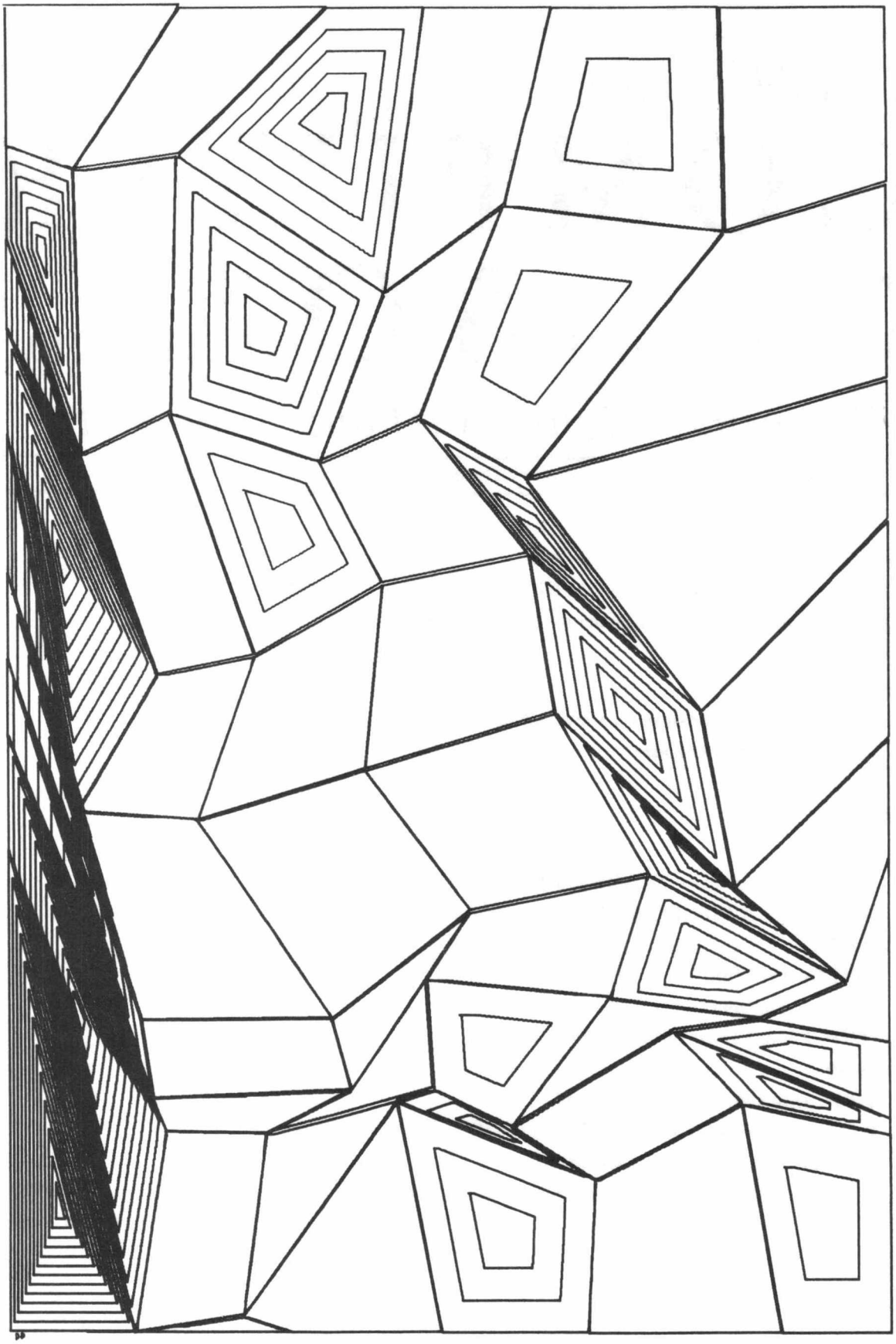
For 8080's

VDM-1 Driver Mods – *John Moorhead* 13

KAPIAR: AN 8080 MACROPROCESSOR – *Steve Newberry* 31

complete documentation & source code

code is in PL/M and *the resulting assembler & machine code*



© David Dameron 1979

Product Review

NOTES ON THE DC HAYES DATA COMMUNICATIONS ADAPTER

Jef Raskin *Received 77 Nov 17*

DC Hayes "Data Communications Adapter" model 80-103A. Available from D. C. Hayes PO Box 9884, Atlanta, GA 30319.

The testing reported on here was performed by Ron Crane, who is chairperson of the System Design Subcommittee and member of the Protocol Subcommittee of the PCNET group. As reported on in these pages, the PCNET people are trying to establish a set of communications guidelines that will help ease communication between personal computer users. This reviewer has attended a few of their meetings, and read their proposals, and if he was writing a review of their work would give it very high marks.

But this is a review of the DC Hayes board. This board does allow two way communication between computers over phone lines under full computer control. This makes it one of the products that were wished for in the review of the Pennywhistle modem. However, like most products in our marketplace, there are a number of problems that the naive user wouldn't anticipate, and that the advanced user must deal with. The situation is especially complicated since a number of regulatory agencies and their regulations (also called "tariffs") affect the use of this device. For example, though it is electrically possible to connect this device directly to the phone lines, the present tariffs require that the user purchase or rent a DAA (Data Access thingamabob) from the phone company or other vendor. The DC Hayes board then attaches to the DAA which then attaches to the phone line. This means extra bread. If you call the phone company, they may even tell you that you cannot use it at all. Don't call them. In any case (to make a pun) the rules are being challenged in court, and in congress, and at the FCC (Federabobble Communications Commission) and a lot of things are up in the air at present. Unfortunately, the 80-103A board does not quite meet phone company specs. The board should provide a 600 ohm load, but is only 500 ohm. Then you need a 600 to 900 ohm matching transformer (this required transformer and some protective and filtering circuits are what you get in a DAA). Ron points out (and has told the manufacturer) that changing R8 to 6800 ohms, and R11 to 470 ohms, with a .0039 microfarad capacitor across R8 to filter out harmonics, will make the device have a better output impedance, and will cut down on the high frequencies that the phone company deplores.

It should be mentioned that this review only covers the *analog* side of the board. The S-100 portion of the interface has not been tested by the reviewer or by Ron. However some evidence has been accumulated showing that it *does* work with the computer.

The manufacturer has promised to make the changes Ron has suggested. He notes that similar suggestions have been received from other purchasers. The overall impression is that this device may well prove satisfactory for inter-computer communication at a hobbyist level. Its specs are not as tight as commercial units–but then its price is far less. It has many features, such as automatic phone answering, that are otherwise unavailable. It can operate in both originate and answer modes (see the Pennywhistle review in the August 1977 issue for definitions). It will probably work under good line conditions, but it may not work over a long distance circuit reliably. One reason is that the board only puts out –7.7db instead of the –6db (referenced to 1 mw) claimed. If you don't understand the jargon, in plain English this means that it's feeble. When you look at the circuit (if you like such things) you will see that the 100 ohm series resistance of the transformer has been neglected. Oops.

Another oops is that the board depends on getting 1 MHz from the computer. If it gets something else, the tones it puts out will be something else. IMSAI and MITs computers have the right clock rate. Polymorphic systems' is 8% too low, which will make the signals borderline quality, but just how bad it is hasn't been tested. Z-80 systems will also sometimes be incompatible, especially if they are running at 4MHz. The proper way to build boards is to use an onboard crystal oscillator. We have noted this in the past. That way, even if you crank up the performance of your computer by goosing the clock, all the peripherals will still run at the right rate.

To conclude: the Pennywhistle was 50% of the way to personal computer communications. The DC Hayes is 80% of the way there. Will someone please give us an affordable 100% effort?

WILL WE MAKE IT?

Dear Jim Warren,

I am renewing my subscription with some reservation. It is November 10 and I just received the August issue 2 days ago. You have slightly more than 1½ months to get out three issues in order to give your subscribers their money's worth.

I think it is inexcusable for you not to meet your deadlines when you act as a consumer advocate for computer hobbyists. If you do not get out your 10 issues before Jan 1, is that not false advertising? Maybe you will get all of the issues out–I think not.

However, since you have the only publication of its kind I'm willing to take a chance on another year with you. A couple of issues ago you had quite an article on Wayne Green. At least he gets his magazine out on time.

Bill Eisinger

Bill–It looks close, but I'm betting we'll get it in the mail under the wire. The rest is up to the Post Awful.–TRW

NORTH STAR SOFTWARE PROFUSION

Dear Jim: *Received: 77 Dec 12*

Some information of interest to all personal computer owners utilizing a North Star Disc Drive and compatible North Star Software. Additional software packages, from varied independent sources, are now appearing. In fact, at my last count, there were 7 independent firms offering North Star compatible Software. I have personally purchased several packages.

George Rompot, Business Computer Systems, 216 Collier Dr., Springfield, Illinois, offers two: a package of basic programs covering Business, Financial, Real Estate, etc., 18 programs, $50.00, and a set of Utility programs in machine code, $25.00. The Utility package includes a COMPRESS, SORTDIR, and two other programs. Both packages are reasonably priced and I consider them to be excellent Software.

The XEK assembler, disassembler and monitor package as offered by the Byte Shop of Westminister, California, is a good Software package to have available to help realize some of your micro's power and capabilities (with good Software). At $48.00 including a good set of documentation, but not quite as complete as might be required to understand the full power of XEK package, this Software is well worth purchasing.

H. B. Rothberg
Box AF
Woodbridge, CT 06525

New Voice Synthesis & Graphics Products at the Mini/Micro Show

Dear Dr. Dobb's, Received: 77 Dec 19

Enclosed is a copy of a letter I recently sent Bill Fuller and Neil Ferguson of the Computer Hobbyist Group of North Texas. It is about some of the new personal computing products I saw at the Mini/Micro show and my preliminary evaluation of them. I also sent a similar letter to Sol Libes and Russell Gorr of the Amateur Computer Group of New Jersey. I've been particularly impressed with the work these people have been doing. They really do sincerely care *about computer hobbyists. That's why I wrote them* and *also the reason why I am writing you now. If you feel my letter to be of sufficient general interest to fellow computer enthusiasts, please feel free to use it in your publication.*

Sincerely,
Kenneth Young *3311 W. 3rd St., Apt. 1-319*
(213) 383-9666 *Los Angeles, CA 90020*

Dear Bill and Neil, *Dated 77 Dec 14*

I just attended the recent Mini/Micro show held December 6-8 in Anaheim, California. Anaheim is south of Los Angeles and takes approximately 35 minutes to drive there. The turnout was moderate and it wasn't really aimed at hobbyists. However, there were four new S-100 products introduced which I felt might be of interest to fellow CHGNT members: two speech synthesizers and two computer tablets. I've enclosed what little literature they had available. All these manufacturers said they'd be delivering their newly announced products by mid-January. I'm a little skeptical, but we'll see.

Votrax is introducing a $650 assembled S-100 speech synthesizer. It looks very similar to the $595 Digital Group/Votrax speech synthesizer except that there will be no voice input circuitry. The board I saw was on a Vector S-100 prototyping board. Most of the board was covered with a ½ dozen support chips I could see. The computer-synthesis-by-rule (CSR) facilities of this product are embedded in the hardware and are therefore fixed and inflexible. Votrax also plans to make software available at additional cost. Since they have not written any software yet, they don't know yet what they plan to charge for it. With this product, Votrax should be giving Computalker some competition. Unfortunately, Votrax's unit is more expensive and, like Computalker, you have to get used to the way it produces user-generated words before you can recognize what it is saying. It is possible with the Computalker to create very natural sounding speech. However, you must have a fairly deep understanding of voice synthesis techniques and access to very specialized equipment like an audio spectrum analyzer. I've owned a Computalker for about a year now and I am semi-serious about obtaining a Votrax unit. My reasons are that the CSR facilities on the Votrax *seem* to be a little easier to use and that Votrax's CSR generated speech *might* be a little better quality than Computalker's. However, it's difficult for me to tell which is really better without having the units side-by-side for comparative analysis purposes.

Speech Technology Corporation of Santa Monica, California, is introducing a $495 assembled speech synthesizer also. This company's approach to speech synthesis is a little different from that of Computalker and Votrax in that no CSR facilities are provided so that a user can create his own vocabulary. Fixed vocabularies of clearly understood words are provided on 2708 and 2716 EPROMs. These vocabularies range in number from 30 words to 60 words. It wasn't clear to me how much these vocabularies would cost, except they'd be expensive. I asked the company representative if they planned to make any CSR software available and the answer was no. The only advantage I can see that this unit may have over Computalker and Votrax is that the speech synthesizing circuitry is not epoxy-encased. Perhaps this manufacturer, unlike other companies in the same field, might therefore make available to the public complete schematics of how an artificial voice is generated.

Two established computer tablet companies, Summagraphics and Talos, are now entering the personal computing market. I personally think that tablets (sometimes referred to as "x-y digitizers") represent a very exciting new peripheral for computer enthusiasts. In my opinion, tablets are much superior and more accurate than light pens to work with. However, it would take me several pages to explain why I personally believe this. Maybe some computer club member might take some time at the next meeting to explain to interested people what a tablet can and cannot do for you. I should add that because of the lack of tablet applications-oriented software in the microcomputer market, I cannot recommend these marvelous devices to everyone, especially businessmen and computer novices.

Summagraphics told me that they manufactured all the tablets for all of Tektronix's high resolution computer graphic terminals. I verified this claim with my company's Tektronix representative. This really impressed me since Tektronix standards are known to be quite high. I personally like the Summagraphics tablet and have ordered one. Supposedly, it'll arrive by January 15 of next year. I am obviously biased towards this product. You should take this into consideration in my brief evaluation of tablets.

Summagraphics tablet is $555 and is called the"Bit Pad". The resolution on this device is either 0.005" of 2200 x 2200 lines *or* 0.1mm of 2794 x 2794 lines on an 11 inch square active area. You specify which resolution mode (inches or metric) you want when you order. I spoke to Bill Kiss, one of the designers of the Bit Pad, and he told me there is a jumper on the PC board which'll allow us to have *either* resolution mode. This is good to know especially since this capability is not described in their literature. Why Summagraphics didn't incorporate a switch to allow a user to readily change modes is a mystery to me.

The Bit Pad does *not* come with a power supply. This device requires +5v @ 500ma regulated, +12v @ 70ma regulated, and −12v @ 50ma regulated. There is an optional power supply, which is an external one, that costs $95.00. It sort of looks like the external power supply on Radio Shack's Z-80 computer. For $30.00 you can put regulators onto the Summagraphics PC board to that you can tap the unregulated voltages directly off your computer power supply.

The normal Bit Pad interface to your computer is through an 8-bit parallel port. Supposedly, the Bit Pad is a latched parallel device. Therefore, we should be able to use it with unlatched parallel devices such as Processor Technology's 3P+S. (I sure hope the pulse width of the strobe signal is of the right length to allow me to do this.)

There is an RS232 serial option for $125.00. I personally consider the serial and power supply options to not be good deals, and therefore did not purchase them. There is a $75.00

single button cursor option which I did order. There is a little ½" diameter piece of glass with cross-hairs on this device. Supposedly, this allows one to input x-y coordinates more accurately than with the stylus, which comes standard with the Bit Pad. I'm not sure how valuable this option will be, but I guess I'll find out.

At the show, Summagraphics had two Bit Pads on display. One of them was hooked-up to the latched parallel input port of Polymorphics video terminal interface (VTI) board. Two programs were demonstrated on the Poly setup: one of them was to decorate a Xmas tree with ornaments and the other was a paper keyboard. What's a "paper keyboard"? Remember that in place of a keyboard, Summagraphics attached their tablet to the input port of Poly's VTI board. To simulate a keyboard, Summagraphics drew a keyboard on a piece of paper and then taped it to their tablet. To enter a character, you touched the appropriate spot on the paper keyboard and the desired character would then immediately be sent to the computer! Of course, with a set-up like this, you could custom-make your own keyboards. The other Bit Pad was attached directly to a high resolution Tektronix storage display scope. You could draw anything you wanted to on the scope by moving the stylus on the Bit Pad. This proved that the Bit Pad resolution was high enough to draw circles and curves without any visible break in the figures. With the Tektronix demo, I discovered that when your stylus was outside the 11" x 11" active area of the tablet that zeros were sent to the computer. Summagraphics explained to me that this was not a flaw in their equipment, but the way it was designed. I was quite impressed with these demos.

A tablet such as the Bit Pad is a marvelous input device for high resolution graphic displays. However, most hobbyists do not have high resolution graphic output devices (yet). Are there *interesting* things that can be done without a high resolution graphic display? I think so. A ball-point pen cartridge can be used in the tablet's stylus. (I'm hoping this ball-point pen cartridge is a regular one that I can buy inexpensively at any department store.) This will allow you to write. Therefore, if you had something like a multiple-choice questionnaire taped to the tablet, the computer could record the user's responses and score it (if necessary) in real-time. With some software pattern recognition routines, it *might* even be possible to recognize characters (such as numbers, letters, or Oriental hieroglyphics) printed on the tablet. There would then be little need to type-in user responses via a keyboard, which I personally consider an awkward input device anyway (*especially* for those people who can't type). I have a friend, who is an orthodontist and who has also recently ordered a Bit Pad, who plans *to try* to use his tablet as a precise measuring device for determining distances on dental x-rays. Even if we fail in our attempts to implement these initial applications, we feel that a tablet is such a potentially powerful tool that we should *eventually* be able to find something useful and interesting for it to do.

Commercial 11" x 11" tablets are several times more expensive than the Bit Pad. I did a little research and found the costs of commercial 11" x 11" tablets from the following four companies to be:

1) $3200 from Tektronix
2) $2400 from Summagraphics
3) $2500 from Talos
4) $1600 from GTCO Corp. (qty 25-49)

The natural question to ask is why is there such a big difference between the Bit Pad and its commercial counterparts, so I did. The answer I got was that the Bit Pad was not as precise on input. Supposedly, there is a "one-point wobble". If you try to input the coordinate from the same spot on the tablet several times, you won't always get the same values. For example, one time the x-coordinate may be 1000, the next time it may be 999 or 1001. With a commercial unit, there is absolutely no wobble at all, but this additional reliability really costs a lot! Also the Bit Pad is a byte-oriented device. Everytime a coordinate is obtained, 5 bytes of information are sent to the computer. In commercial tablets, it is possible to send coordinate information to a computer like the IBM 370 all at one time. I personally consider these "acknowledged limitations" to be relatively trivial for my purposes and consider the Bit Pad to be an excellent buy for the money. Of course, I won't *really* know how good a buy this product is until after I've used it for a while.

Talos offers a $495 assembled tablet called the "Personal One". The resolution is only 256 x 256 lines on an 8 inch square active area. It requires two parallel ports: one for the x-coordinate and one for the y-coordinate. The input power requirement is 12v to 15v @ 100ma. I thought this device looked cheap and did not particularly like it. The resolution is too low and at $495 the price is too high.

At the show, the Talos tablet was attached to an IMSAI which contained a Dazzler. The Talos software demo used Cromemco's game of "Track" where the object of the game is to move a beam of light into the center of a spiral maze. Instead of using a joystick as Cromemco does to move the beam of light, Talos used their tablet. When I played this game, I sometimes saw several random spots appear simultaneously on the TV when I inputted a single coordinate value. I asked Talos about this and they said random noise was being generated on the AC line by the equipment of adjacent vendors at the show and that this was coming into the IMSAI. They said I wouldn't have this problem in the privacy of my home. Maybe this is true, but I am skeptical.

Talos also plans to introduce a $300 assembled color graphics board called the "DV-S100". The "DV" is an abbreviation for "Display Video". This product is briefly described in their literature. However, it was not shown at the show. This output device was designed to go hand-in-hand with their tablet. In addition to its color graphics capabilities, this board contains a microprocessor, two parallel ports, and software drivers in EPROM to operate their tablet and color graphics board. I personally think an individual might have a hard time using the Talos tablet without this second product.

The Talos people told me that they thought the color resolution of their DV-S100 would be 256 x 256! This is better than the color resolution on a Compucolor, which is 160 x 192 on a 15 megahertz color monitor!!! One of my hardware friends was skeptical of this claim. He felt the practical resolution on a color TV was about 128 x 128 since a color spot on a normal 5 megahertz color TV takes up 3 points. On a black-and-white Sanyo 4092 monitor (which a lot of us hobbyists own), the maximum number of horizontal points is about 400. Maybe our skepticsm is based on ignorance of color graphics technology (& lack of a working demo) and is therefore unjustified. Anyway, this product supposedly will be available in January. If this product is for real, I would seriously consider getting it. I would like to have a higher resolution computer graphics interface to replace my Dazzler.

Now for some miscellaneous information. At the show, I saw Micro Peripherals. Micro Peripherals is a new floppy disk manufacturer in Southern California. There are so many floppy disk manufacturers here that I'm tempted to refer to my section of the country as "Floppy Flats"!!! Anyway, Micro Peripherals had a full page advertisement in the October 1977 issue of *Byte*. I asked them when their S-100 minifloppy system would be available and they said about three months. Therefore, they *might* start delivering in March. I'm personally a little wary of this company. I think they may have entered this market a little too late.

After the show, I checked-up on Associated Electronics of Fullerton, California as requested by Dick Tillbrook in the

MICROFLOPPY™ DISK DRIVE WITH DOUBLE DENSITY RECORDING INTRODUCED AT NCC

news release *Received: 77 Jun 18*

DALLAS, Texas -- The first flexible disk drive for 5-¼" diskettes to offer double density recording of 250,000 bytes on each side of a diskette, was introduced at the National Computer Conference here today by the Pertec Division of Pertec Computer Corporation.

The new drive, designated the Pertec Model FD200 Microfloppy™, is mechanically and functionally interchangeable with the diskette drive introduced earlier by Shugart Associates but permits writing on 40 tracks versus a 35-track limit for the Shugart SA400 unit, in addition to the double density feature. The Pertec FD200 also allows recording on both sides of a diskette, whereas other units are limited to one side, according to Pertec.

Both the signal interface connector and the dc power connector are compatible with the Shugart equipment, and mounting holes and outline dimensions are the same.

The complete FD200 measures just 5.75" by 3.25" by 8" and is priced to be competitive with small tape systems such as the Philips or 3M mini-cassette drives. As a result, applications are expected in many types of intelligent terminals, data capture devices, personal computing systems, word processing units and other applications where very compact, low-cost data storage with random access is desirable.

Aside from added capacity and two-sided data recording, the Pertec FD200 reduces interface electronics to a single circuit board using IC components. Other manufacturerers' units typically use two circuit boards and discrete components. The single-board approach is expected to reduce spares stocking requirements and simplifiy field maintenance for OEM customers, as well as improve overall reliability through the reduced components count.

Up to four FD200 drives can be "daisy-chained" on a single 34-line ribbon cable.

The new Microfloppy™drive also employs an IBM-compatible read/write head with "tunnel erase". Head life is estimated at 20,000 hours. A special design feature prevents the head from disengaging from the positioning cam mechanism.

The FD200 incorporates a molded plastic faceplate and door. Light emitting diodes on the faceplate indicate the drive that has been selected. As with Pertec's larger FD400 and 500 models, the new unit conforms to ECMA standards.

Formatted capacity is 102,440 bytes per side. The data transfer rate of the diskette drive is 125,000 bits per second; average access time is 463 milliseconds. Recording density at the inside track is 2580 bits-per-inch.

The FD200 operates with standard 5- and 12-volt dc power supplies. Power dissipation is 15 watts continuous. Meantime between failure is rated at 8000 hours with typical error rates of 1 per 10^8 bits for "soft" read errors and 1 per 10^{11} bits for "hard" read errors. Write-protect features are incorporated in the design.

The new FD200 Microfloppy™disk drive is marketed to OEM users by PCC/Pertec and will be incorporated in both OEM and personal computing products offered by Pertec's Microsystems Division. Delivery of sample quantities is slated for August 1977 with production quantities expected in September. Unit price of the FD200 is $405.

For further information on the new floppy diskette unit contact PCC/Pertec, 21111 Erwin Street, Woodland Hills, CA, 91367; phone (213) 999-2020.

A SELECTRIC TERMINAL FOR THE HOBBY MARKET

News Release *Received: 77 Aug 18*

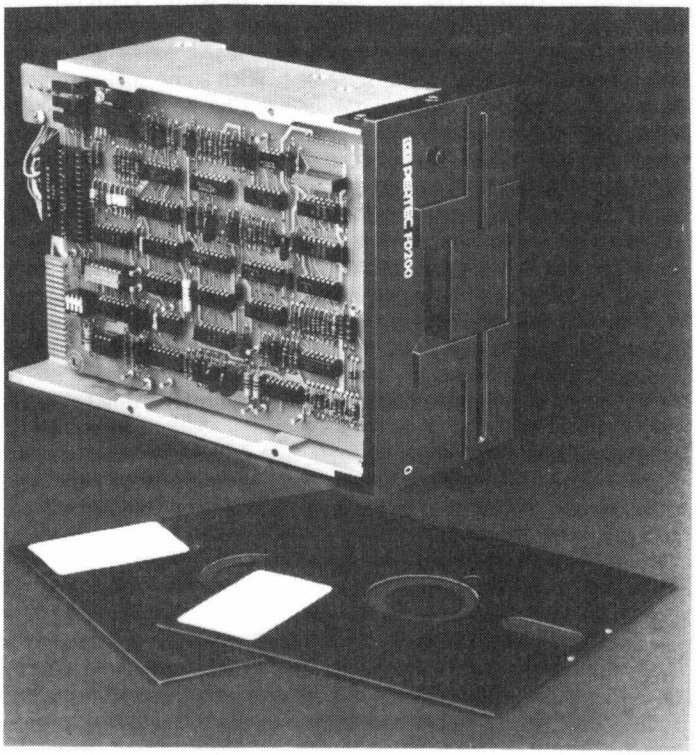
Anderson Jacobson, Inc. (AJ) plans to enter the hobby market with a professional quality I/O terminal at a price attractive to the serious hobbyist.

Originally selling for over $4,000, the AJ 841 is an IBM Selectric terminal and off-line typewriter that has been completely refurbished. It includes a built-in ACVII interface and numerous other features attractive to the professional and hobbyist alike.

Anderson Jacobson plans to sell this unit through the medium of direct mail for an introductory price of $995 plus a small shipping charge. The standard price of the unit is $1195 plus shipping charges. The terminal is fully warranted for 30 days.

Customers will be able to pick up their terminals at specified AJ service locations where units will be thoroughly tested before delivery. These locations will also provide warranty or repair service as required.

For further details contact Steve Tritto at Anderson Jacobson, Inc., 521 Charcot Avenue, San Jose, CA 95131, (408) 263-8530.



The Pertec Model FD200 Microfloppy is the first flexible disk drive for 5-1/4" diskettes to offer double density recording of 250,000 bytes on each side of a diskette.

December issue of the Amateur Computer Group of New Jersey newsletter. Dick had sent in money to this company several months ago for a 16K memory board and has not heard from the company since. I visited their "office" and this company is no longer in existence. According to neighbors, this company went out of existence in the beginning of November 1977. All kinds of people are supposedly after them for money. I'm telling you this because Associated Electronics has a full page ad on page 109 of the January 1978 issue of *Kilobaud. Don't buy anything from this company!!!* I've already written to Dick and told him what I've found. I hope he can get his money back, but I doubt it. I understand how the poor guy feels. I've been there. . .

Sincerely,
Kenneth Young

HEATHKIT'S H-8

A Consumer's View

Copyright, 1977
by Paul R. Poduska, PhD
114 Warwick Street
Lowell, MA 01851
(617) 453-2904

Editorial Note: If nothing else, the Heath H-8 system is certainly sparking a great deal of comment. To those who may have missed it, we refer to the "H-8: Pro & Con" article in Dr. Dobb's no. 19. *Here now is another offering from a user who seems quite pleased with the system–TRW*

Microcomputers–that is to say, microprocessor-based computers–have, over the past several years, evolved into several rather distinct species: the single-board small system, the mother-board-based medium size system, and the full mainframe system. Within this last group, buyers can find several different styles or mainframe configurations. Some models have elaborate front panels with rows of toggle switches and LED indicators, others have no front panel devices at all, while still a third group incorporates numeric keypads and seven-segment LED displays.

None of the main-frame types is *a priori* better than the other. Each system has its own assets depending upon the kinds of uses to which it will be put. And, in this regard, the Heathkit H-8 eight-bit microcomputer might be viewed as a truly general purpose system. By this I mean a system which has wide applicability to a broad range of users as well as functions. From the outset it should be made clear that the H-8 is not intended to rival machines which can process COBOL programs. Nor is it intended for extensive timesharing applications or dedicated use in manufacturing applications. Its appeal lies in its usefulness in general purpose applications–that is, for the non-scientist who may want to build and operate his own personal computer, or the scientist who may need a small yet rather powerful system for his or her laboratory work. With this in mind, I will give my views as a consumer of this fine product.

The front panel of the H-8 has been designed to provide users with easy input and output of data and system control commands. The nine seven-segment LEDs display octal code (split-octal to be exact) which is typically easier for the beginner to learn and use than hexadecimal code. Similarly, all data and control words are entered in octal code through a convenient four-by-four keypad on the right-hand side of the panel. Both of these convenient I/O devices have a number of user-oriented features which make the system easy to access and understand.

LED Display Capabilities:

- Display of register pair contents,
- Display of accumulator and flag register contents,
- Display of any memory location and its contents,
- Display of the program counter or stack pointer,
- Display of input and output ports and associated data,
- Display of cassette tape data record types,
- Display of data and addresses during input from tape,
- Display of remaining blocks to be output during a dump,
- Display of any program running activities, e.g., the contents of registers or the program counter, and
- Display of user generated patterns

On the input side of the front panel, the console keypad provides the user with the following functions:

- Input of octal characters,
- Designation of display registers,
- Designation of memory locations to be displayed,
- Input of memory locations/data to be altered,
- Increment or decrement the memory location displayed,
- Altering of register contents,
- Control of the program counter,
- Display or alteration of accumulator contents,
- Cancel any keypad input,
- Load or dump from or to any desired port,
- Execution of a stored program,
- Single step through a program,
- Hard restart which resets the system logic (see below),
- Soft restart which stores the contents of the program, registers and accumulator on the stack,
- Keys to enable input of decimal data under special control routines, and
- Input or output of port numbers and data.

Taken together, the numeric keypad and octal displays provide users with full control of the system. There is no need to interpret the rows of single-element LED bit patterns found on other main-frame systems. Similarly, the user does not have to go through a binary-to-octal translation process to enter data into memory location or registers. It has always amazed me why this design was not adopted from the outset by other small systems manufacturers, for it is much easier to use and would seem to have much broader appeal than the toggle switch, single-led indicator approach still in use.

Perhaps the reason why the H-8 is the first system to incorporate such comprehensive features is that the firmware that is needed to operate these features is costly to develop. Here I am speaking of the system front panel console monitor for the H-8 which is stored in 1K of ROM on the front panel control board. The ROM contains a very sophisticated set of systems utility routines which do everything from providing keypad debounce logic to calculating parity checksums for tape input and output. Designed for Heathkit by the Wintek Corporation, the ROM monitor not only includes a real-time clock, but also provides the user with access to a number of memory control cells located in a small portion of the user's RAM. For example, you can disable the display update by storing the octal value "002" at memory location ".MFLAG" (040 010 split octal) and then display any bit pattern you desire in the nine LEDs by storing appropriate octal values in the display segment control memory cells (040 013 through 040 023). Or, you can turn on the horn, which is a convenient built-in speaker for user feedback, by executing a simple assembly language program that loads the speaker enable value of "160" into the "CTLFLG" (Control Flag) memory cell at "040 011". (This should be done from a program because altering the contents of CTLFLG from the front panel disables front panel input of the last two octal digits and thereby prevents access to the system).

The ROM monitor also includes routines for inputting and outputting data through the cassette interface logic on the serial interface board via the USART (Universal Synchronous/Asynchronous Receive/Transmit device). This feature provides the user with easy access to the USART status register from the front panel via the port input keypad command as well as the ability to output blocks of data during a tape dump without having to set up the software to control the USART. I could go on and on about the many advantages of this integrated design. At this point, however, it seems sufficient to state that the front panel LEDs and keypad in conjunction with the ROM monitor provide a very efficient and effective (that is, "user-oriented") structure for operating the H-8 even without a teletypewriter or CRT I/O device.

At this point it might be interesting to consider what the buyer gets for his or her dollars, since I mentioned the cost of designing such an integrated system previously. Here is a comparison of the H-8 with a "typical" competitor:

The H-8	Brand "X"
9 seven-segment LED displays	30± single LED indicators
Data/Control Keypad	Rows of toggle switches
10-slot motherboard	22-slot motherboard
10 amp power supply	28 amp power supply
Resident ROM monitor	Optional monitor in user RAM
8080A CPU	8080A CPU
50-pin Bidirectional Bus	100-pin Unidirectional Bus
Not compatible with S-100 Bus	Uses S-100 Bus
Easy access to system data	Somewhat difficult access to system data
Switch selectable operation for 110 or 220VAC	110 VAC only.
Audio feedback for user	None
Fully assembled and tested CPU board	Unassembled, untested CPU board
Quiet, convection cooling	Fan-cooled
Superb instruction manuals	Archaic instruction manuals
Fully tested and operational software	Unreliable software (as reported to me by owners *and dealers*)
Above-average operation manuals	Sketchy operation manuals
Clear assembly instructions	Confusing instruction manuals
Full factory service and warranty	Limited service and warranty
Low-cost peripherals that work	Rather high-cost peripherals that often do not work (again as reported to me by owners and dealers)
Price: $375	Price: Over $800

The above is just a brief summary comparison but does tell a lot about the buyer's dollar value. If you must have an S-100 bus system, then price is not an issue. But if you believe, as I do, that many companies will be coming out with H-8 compatible boards and need dependability, reliability, and a cost-effective system, then the H-8 would seem to be the best buy. Not only does the buyer get superior hardware and hardware manuals, but he/she also receives *without charge* a complete software system with excellent manuals. Included in the software package are the following:

- "PAM-8", the front panel console monitor. A full listing of the ROM monitor is provided for users who may need to utilize its sophisticated features.
- "BUG-8", the software debugger program which provides users with easy access to and ability to examine and modify "core-resident" programs,
- "TED-8", the H-8 Text Editor, which is useful for writing assembly language program source code or for any personal text editing projects,
- "HASL-8", the 8K assembler which provides users with a rather sophisticated set of assembly language commands for developing 8080A source code programs, and
- Benton Harbor Basic, an 8K Basic incremental compiler which incorporates many sophisticated features not found on other systems.

All of the software is interlinked with the front panel monitor through the 1K ROM, which provides users with console routines which are used throughout all of the software (e.g., tape dump).

If the reader thinks that I am working for Heathkit because of all the positive statements that I've made thus far, then he/she should know that I don't. The H-8 is just a very well designed, well executed product that cannot help but sell itself. It goes way beyond being a slick image of a computer which is very important for the inexperienced hobbyist or first-time user.

But what would a story be without some disappointing parts? Here are a few. It is unfortunate that the text editor and assembler were not linked together (or user linkable through a linking loader). There are often times during the course of writing and editing a program that the user needs to go back and forth from one to the other until the final program runs. The repeated process of going back and forth from the text editor to the assembler is annoying and could have been prevented if they had been designed to be linked in core (user RAM).

A second misfortune is that the text editor runs in either upper or lower case, but not both. This is a real minus for anyone who wants to do serious text editing of manuscripts or who just likes to see better system software design. The problem also extends to the Heathkit H-9 Video Display Terminal which cannot use both upper and lower case characters. I can't help but hink that this must have been an oversight (of major impact) of the software/hardware designers—a costly one.

A third problem for anyone who is a moderately adept typist is the annoyance of "command completion", a feature used in all H-8 software. For the novice who can't type fast or the expert who hasn't mastered the keyboard this feature is nice. But for those who are used to computers in one way or another, or who can type, this "feature" is a real drawback. How many times I've had to reenter a line of text or program because I typed a "RUN" instead of "RU", the result being "RUN N", because the system adds (completes) the command with the last "N". Hopefully Heathkit will offer a second version of their software which will let this "feature" be an option.

A fourth (and last) regret is that the H-8 1K ROM monitor could have been expanded to include operating system commands which would have made software handling much easier. Similarly, I would have liked to see more than two bytes of RAM for the real-time clock so that these bytes could have been used directly for a twenty-four hour run-

INFO 2000 DISK SYSTEM FOR HEATHKIT H8

Press Release *Received: 77 Dec 23*

INFO 2000 Corporation, announces availability of a disk system for Heathkit H8 computers. The complete INFO 2000 Disk System includes PerSci Dual Diskette Drives, power supply, case, intelligent controller, adapter, cables and disk monitor in EPROM. The INFO 2000 Adapter Board contains the Z80 microprocessor and all support chips, 7K of EPROM, 1K of scratchpad RAM for the Disk Monitor, and all necessary logic for interfacing the disk system to the Heathkit H8.

With the addition of the INFO 2000 Disk System and installation of its Z80/Disk Adapter Board (by simply plugging it in) the H8 computer can operate in either of two switch-selectable modes. One mode enables continued use of the H8 EPROM monitor with the existing Benton Harbor software. No modification is required, and the H8 will perform at Z80 CPU speed and use the Z80 instruction set.

The second mode supports the INFO 2000 Disk Monitor, and other software adapted by INFO 2000 for use with all their Disk Systems, including the TDL software library and CP/M.

INFO 2000 Corporation offers individual consultation regarding their Z80 Disk Systems. Dealer inquiries are welcomed. Cost for factory assembled and tested 2000 Disk System for Heathkit H8 is $2,750. Contact INFO 2000 Corporation, 20630 S. Leapwood Avenue, Carson, CA 90746. Telephone (213) 532-1702.

time clock. Hopefully (again) Heathkit is making plans to come out with a second version of their ROM which will add such goodies.

Overall, and in perspective, these misfortunes and misfeatures are not serious at all. They do not in any way impair my ability to use the H-8. But they would enhance the system's utility considerably. The important services have been provided in a well designed package at a very reasonable cost. We can look forward to seeing (I am told by Heathkit) an excellent microfloppy disc system next year ('78), A to D/D to A converters, new Video Terminals, software packages, an EPROM board, and probably a host of devices for ham radio buffs who are also using RTTY systems. I still have not heard a thing from the HUG (Heathkit Users' Group) although they were quick to cash my membership check over a month-and-a-half ago. But these are new services bing provided by Heathkit, and like the somewhat inconsistent hardware service at many of the stores, will, I am sure, be resolved in the near future. All in all, Heathkit's initial venture has been one of excellence and in keeping with their long history of quality electronic kits. I am sure that all of us will be hearing a lot more from and about their computer products for years to come.

DIABLO TO 8080 INTERFACE CHALLENGE

Dear Jim; *Received: 77 Nov 28*

I am again standing in need of assistance. Having just acquired a Diablo series 1345 HyType II Printer, I am in the process of interfacing it to my Processor Technology SOL 20 with 32K of memory.

In order to get it operating as quickly as possible it will be interfaced initially as described for the "Qume" by Jef Raskin in the September issue of *SCCS Interface* using a software driver. Hopefully, I can later get a copy of the driver that is said to be available through the Intel 8080 Users Group.

This kind of interface will not fully utilize the potential of this printer, especially for graphics, and will also occupy a considerable chunk of memory. In order to more fully utilize the printer's potential and to minimize I/O pin assignments to a PTC 3P + S Module I would like to purchase, construct or otherwise acquire a set of PC boards which will plug into the two spare 56-contact sockets on the printer motherboard. This would then comprise an 8080 terminal microprocessor and interface similar to the Diablo HPR02 and 8080 Interface and would in effect convert the printer into a complete terminal with pretty extensive capabilities. The very expensive set of boards available through Diablo also converts the many wire parallel interface into a five wire RS-232 interface.

This type of printer has recently become widely available in volume on the surplus market. It would seem to me that it may well be the wave of the future for the serious hobbyist. Such a PC board set might make a good series of articles for *Dr. Dobb's Journal* especially if accompanied by a soldertail PC board kit. There isn't room for wire wrap.

It is possible that some-one has already done this. If so, I would like to get into contact with him. Failing that, with some-one who is interested in developing such a board. Any advice that you can give or assistance that you may be willing to offer will be greatly appreciated.

Sincerely,
Peter Nevius

2419 Brookshire Drive
Schenectady, NY 12309

Editorial Note: There you are, hardware hackers. Anyone out there care to take up the challenge? –TRW

A NOTE FROM DAY ON DMA

Received: 77 Nov 28

As Peter Traneus Anderson points out in his letter published in the August '77 issue of *DDJ*, the usual way of doing DMA on a microprocessor leaves considerable room for improvement. This is especially true in the case of machines using the S-100 bus, which really wasn't designed with DMA in mind. Fortunately, so far as I know, no one is under any obligation to implement DMA in such an awkward and inefficient manner.

Many alternative schemes are possible, almost all of which are better than those currently in use. For example, one could use *interleaved memory* with odd bytes in one physical bank and even bytes in another. Then the MPU and a video display could both access memory with minimal delay. For other memory access techniques, see Don Lancaster's forthcoming book on video displays.

Jim Day

17042 Gunther St.
Granada Hills, CA 91344

A KIM-1 DISASSEMBLER

Received: 77 Oct 28

I have just got my KIM1 up and running with a FORETHOUGHT motherboard kit, and an 8K ECONORAM kit from GODBOUT. I was impressed with the quality of both kits. The ECONORAM kit was a little too crowded. I had to unsolder some of the despiking capacitors that were too fat to fit between the 2102s. Next time, I will put the thin ones in these locations.

My first job was to get "A 6502 DISASSEMBLER FROM APPLE", (*DDJ* VOL 1 no. 8) to work. I had more than a little trouble, till I discovered that the KIM1 Outch and Prtbyt routines were clobbering the Y register that the disassembler was using as a counter. You might like to publish my version of this program which is enclosed. If anyone wants to avoid the problem of hand keying it in, I would be glad to furnish a copy on paper tape for $1.00 or on cassette for $3.00.

I am mainly interested in engineering applications. I would think that the micro computer is so cheap now that it would displace many applications now on time-sharing. Of course, the answer is in the cost of programming. That is where we computer buffs could perform a real service and have a good time doing it. Time sharing is good but the overhead is high. There is nothing like having your own computer right there when you want it.

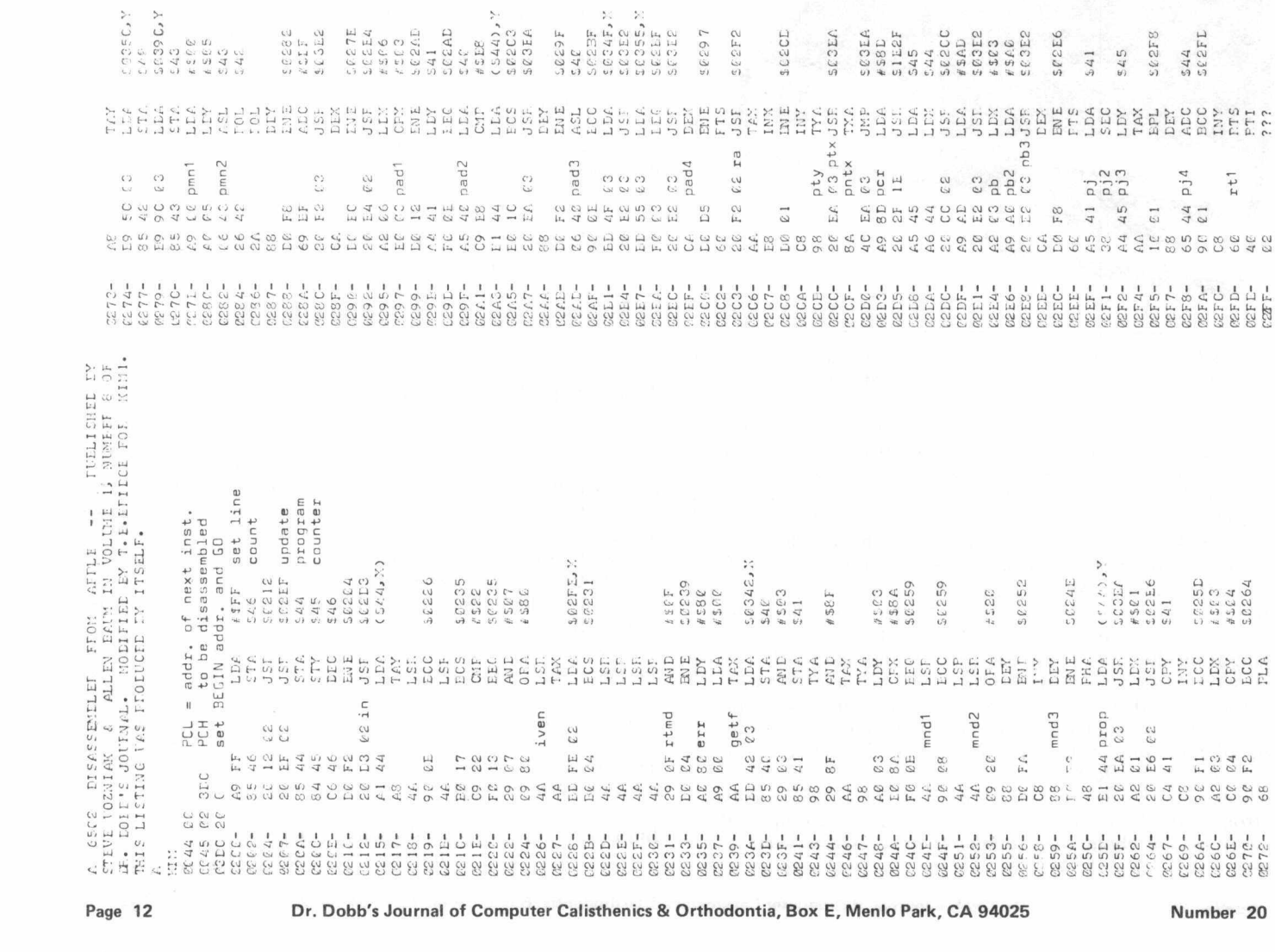
I would like to use BASIC, but I fear it will be too slow for some of my problems. For example, a single problem in PIPE STRESS may need to invert a 6 X 7 matrix 20 times. If we could only debug with a small problem on BASIC, and then compile the source tape into machine language for running large problems. The construction of such a compiler is far beyond my capability. Does anyone market a BASIC with floating point, and arrays that will run on my KIM1?

To work on any big engineering program, you need at least two people. One must be knowledgeable with the problem. You need feedback in working up the documentation, and in other ways too. I am 70, retired and available. I have a degree and 40 years experience in mechanical engineering. I can run a typewriter. Let me know if I can help you with a problem. I submitted the following programs to the DECUS library: PIPE STRESS, CRITICAL PATH METHOD, REGRESSION ANALYSIS, BUILDING HEAT LOSS. Having only 4 K of core on my PDP8f that I recently sold, BASIC was out of the question. Having no feedback from anybody, my documentation was bad, in spite of very great effort.

Very truly yours,
Theodore E. Bridge

54 Williamsburg Dr.
Springfield, MA 01108

PS In writing the above mentioned programs for the DECUS library, I made good use of a FLOATING POINT INTERPRETER in the DECUS library. It uses much less memory than BASIC and runs very much faster. Would anyone like to help me write a similar interpreter for my KIM1? I worked up a trace routine for the interpreter. I grant that debugging with BASIC would be much easier. Also only a few people can communicate with you if you are using your own interpreter. However, a BASIC interpreter is not really useable on a very large engineering application.



KICKING THE 8080 HABIT

Received: 77 Oct 17

Maybe it's just my imagination, but it seems that a lot of people aren't utilizing the Z-80 to its fullest. Everyone is so used to writing code for the 8080 that they don't seem to bother upgrading their software when they upgrade their CPU. Or maybe, as in my case, since we've been eating and sleeping in Intel Octal for so long that we just can't seem to catch on to all the enhancements of the added Z-80 functions and instructions. I would like to see you guys (or, someone else) come out and explain all the nifty Z-80 tricks. I know I can't be the only one that is stuck in the rut of 8080 code. (Please!! Don't tell me I swapped my CPU board JUST for speed–the software potential is fantastic.)

Michael Settle
751 Washington #115
Arlington, TX 76011

How about it? Anyone care to send us something on "Taking full advantage of Z-80's inner mysteries"?–TRW

MICROCOMPUTER SOFTWARE INDEX

News Release *Received 77 Nov 28*

A great quantity of software has been published in microcomputer books and magazines since late 1975. These programs range from complete BASICs to assemblers, routines, games and direct application programs.

The Schreier Software Index, An Index to Published Microcomputer Software, indexes hundreds of published microcomputer programs. The S S I contains cross references and over 130 program divisions. Many of the cited programs specify chip description. In addition, the S S I features complete publication documentation and bibliographic data. Locating a random number routine for a 6800 or an 8080, for example, takes but seconds.

Direct orders may be placed with S S I, 4327 East Grove Street, Phoenix, Arizona 85040. Price postpaid in the United States, Mexico and Canada is $5.00. The S S I is also available from a number of microcomputer dealers across the U.S.

```
THESE CHANGES WERE ADDED TO SAVE
THE  Y  REGISTER WHEN CALLING
KIM ROUTINES  OUTCH & PRTBYT

KIM
03DC 20 G
03DC-   20 00 02    JSR   $0200  begin
03DF-   4C 4F 1C    JMP   $1C4F  start
03L2-   84 EE       STY   $EE    savey
03E4-   20 A0 1E    JSR   $1EA0  outch
03E7-   A4 EE       LDY   $EE    gety
03E9-   60          RTS
03EA-   84 EE       STY   $EE    savey
03EC-   20 3B 1E    JSR   $1E3B  prtbyt
03EF-   A4 EE       LDY   $EE    gety
03F1-   60          RTS

THESE ARE THE TABLES

KIM
03DC 20 44
0044 F2 DC.
0045 03 3DC
03DC 20 17F7
17F7 FF DC.     end of tables
17F8 FF 03.
17F9 LA 0300    start of tables
0300 45 C
;1803004503D0084009302284533D0084009400245533D0084009400058A
;1803184045D0084000000224433D08C44C011224433D08C449A10220782
;1803304433D0084009102244433D0084009621378A900218182000000667
;180348594D9192864AAA59DACA9ACA5A8A4A9900D8A4A4001C8A1C230C0C
;18036005D8E1EA19D8A1D239D8E1DA1000219AE69A8192224531E230B5E
;1803782455319A1001A5E5EA5692424AEAEA8AD29007C00159C6D00085E
;18039A5692958413341A56923A0D8625A4826629485444C8540A16
;1803A86844E894000E44884784D4286E74F4CC4A72F2A48A00AAA2A20CE7
;1803C074747472446BE232E2000220C1A1A2600727288C8C4CA26480997
;1803D84444A2C8220000240C4F1C84EEE2CAC1EA4EE6084EF203E1EA40A8F
;00000A0000A
```

VDM-1 DRIVER MODS

by John Moorhead

Received: 77 Dec 2

```
0005 *  THIS IS THE VDM-1 DRIVER ORIGINATED
0010 *  BY LI CHEN WANG (DR. DOBB'S JOURNAL,
0015 *  VOLUME #1, ISSUE #6).
0020 *
0025 *  MODIFIED TO INCLUDE SPEED CONTROL AND
0030 *  GENERALLY EMULATE THE SOFTWARE DRIVER FOR
0035 *  PROCESSOR TECHNOLOGY'S VDM-1 VIDEO DISPLAY.
0040 *  THIS PROGRAM WILL ALSO ACT AS THE DRIVER
0045 *  FOR SOLID STATE MUSIC'S VB-1 DISPLAY.
0050 *
0055 *  WRITTEN BY JOHN MOORHEAD, 928 J STREET,
0060 *  DAVIS, CA.  95616  TEL NO 916-758-2495
0065 *
0070 *  MAY 5, 1977
0075 *
0080 *  BACKSPACE = CONTROL H OR B.S. KEY
0085 *  SCREEN CLEAR = "CANCEL" = CTRL-X
0090 *  AUTO CRLF ON INPUT OF A CARRIAGE RETURN
0095 *  CHANGE SPEED BY TYPING NUMBER DURING OUTPUT
0100 *  STOP DISPLAY BY TYPING SPACE BAR
0105 *  START DISPLAY BY TYPING ANY CHARACTER
0110 *
0115 *  NOTE - A SCREEN CLEAR MUST BE THE FIRST
0120 *  CHARACTER SENT IN ORDER TO INITIALIZE THE
0125 *  CURSOR AT THE BOTTOM LEFT CORNER OF THE
0130 *  SCREEN AND ENSURE PROPER SCROLLING.
0135 *
0140 *
0145 *  ENTER WITH CHARACTER IN THE ACCUMULATOR
0150 *
0155 *
0160 STRT   PUSH   H       SAVE SYSTEM REGISTERS
0165        PUSH   D
0170        PUSH   B
0175        PUSH   PSW     CHAR IS IN THE ACCUM
0180        LHLD   VDMP    GET SCREEN POSITION POINTER
0185        CPI    0DH     IS THE CHAR A CARRIAGE RET?
0190        JZ     CR      YES - SCROLL, OUTPUT A CR
0195        CPI    08H     IS IT A BACKSPACE?
0200        JZ     BS      YES - MOVE CURSOR BACK
0205        CPI    18H     IS IT A CTRL - X (SCREEN CLEAR)?
0210        JZ     CLEAR   YES - ERASE ENTIRE SCREEN
0215        CPI    7FH     IS IT A RUBOUT CHARACTER?
0220        JZ     SPEED   DON'T DISPLAY (FOR ALS-8 ONLY)
0225        CPI    20H     DON'T DISPLAY CONTROL CHARS
0230        JC     SPEED   EXIT TO CHANGE SPEED
0235        MOV    M,A     IT HAS TO BE DATA
0240        INX    H       UPDATE CHAR POSITION ON SCREEN
0245        MVI    M,0A0H  PUT CURSOR ON SCREEN
0250        JMP    LINOV   TEST FOR LINE OVERFLOW
0255 BS     MVI    M,20H   REMOVE CURSOR
0260        DCX    H       BACK UP POINTER
0265        JMP    BS-5
0270 CR     MVI    M,20H   CHAR IS A CARRIAGE RETURN
0275        MOV    A,L     UPDATE NEXT CHAR POSITION
0280        ANI    0C0H
0285        ADI    40H     SETTING UP FOR NEW LINE
0290        MOV    L,A     ADDRESS OF NEW LINE
0295        MVI    A,0
0300        ADC    H       ADD WITH CARRY
0305        MOV    H,A
0310 LINOV  SHLD   VDMP    SAVE POINTER FOR NEXT CHAR
0315        MVI    A,7FH
0320        ANA    L
0325        JNZ    SPEED   EXIT
0330        MVI    M,20H
0335        LXI    H,0CFC0H  SET UP DATA LINE (LINE 16)
0340        SHLD   VDMP    SAVE POINTER
0345        LXI    H,0CC40H  15 LINES OF SCREEN DATA
0350        LXI    D,0CC00H  TOP OF SCREEN.  SET UP
0355        LXI    B,03C0H  TO SCROLL 15 LINES
0360 SCROL  MOV    A,M     START SCROLLING UP
```

```
0365          STAX   D
0370          INX    H
0375          INX    D
0380          DCX    B       COUNT DOWN
0385          XRA    A
0390          CMP    B       15 LINES SCROLLED?
0395          JNZ    SCROL   NOT YET COMPLETE
0400          CMP    C
0405          JNZ    SCROL   STILL NOT COMPLETE
0410          LXI    H,0CFC0H  SET DATA LINE (LINE 16)
0415 ERASE    MVI    M,20H   PUT BLANK ON SCREEN
0420          INX    H       ERASE ENTIRE DATA LINE
0425          MOV    A,L
0430          ANI    3FH
0435          JNZ    ERASE   LINE NOT YET ERASED
0440          LXI    H,0CFC0H  SET DATA LINE (LINE 16)
0445          MVI    M,0A0H  PUT CURSOR ON SCREEN
0450 SPEED    CALL   STATS   LOOK AT KEYBOARD STATUS BIT
0455          CNZ    GETIT   IF BUSY, GET ASCII DATA
0460          LDA    NEWSP   LOAD NEW SPEED BYTE
0465          MOV    H,A     CHANGE SPEED WITH ASCII 1-9
0470          MVI    L,80H
0475 DELAY    DCX    H       DELAY ROUTINE FOR
0480          MOV    A,H     VARIABLE OUTPUT SPEED
0485          ORA    A
0490          JNZ    DELAY   JUMP FOR MORE TIME
0495          POP    PSW     DELAY COMPLETE
0500          POP    B       GET READY TO RETURN
0505          POP    D       TO USER PROGRAM
0510          POP    H
0515          RET    .       RETURN
0520 GETIT    CALL   DATA    INPUT ASCII DATA FROM KBD
0525          CPI    '9'+1   IS THE CHAR AN ASCII 1-9?
0530          JNC    WAIT    TOO BIG
0535          CPI    '1'
0540          JC     WAIT    TOO SMALL
0545          ANI    0FH     REMOVE ASCII BIAS
0550          PUSH   B       SAVE REGISTER
0555          MOV    C,A     SAVE DELAY NUMBER
0560          XRA    A       CLEAR ACCUMULATOR
0565          STC    .       INITIALIZE DELAY CARRY BIT
0570 LESS     DCR    C       DECREMENT DELAY NUMBER
0575          JZ     FOUND   STOP ROTATING DELAY BIT
0580          RAL    .       SHIFT DELAY BIT LEFT
0585          JMP    LESS    NEXT ROUND
0590 FOUND    POP    B       RESTORE REGISTERS
0595          STA    NEWSP
0600          RET
0605 WAIT     CPI    20H     IS IT A SPACE BAR?
0610          RNZ    .       NO - CONTINUE
0615 WAIT2    CALL   STATS   WAIT TILL A KEY IS PRESSED
0620          JZ     WAIT2   TO CONTINUE SCREEN OUTPUT
0625          RET
0630 CLEAR    LXI    H,0CC00H  CLEAR SCREEN
0635          MVI    A,0D0H  BOTTOM OF SCREEN CHECK
0640 CLER1    CMP    H       TEST FOR END OF SCREEN
0645          JZ     DONIT   YES - DONE
0650          MVI    M,20H   PUT BLANK ON SCREEN
0655          INX    H       CLEAR MORE SCREEN
0660          JMP    CLER1
0665 DONIT    LXI    H,0CFC0H  SET DATA LINE (LINE 16)
0670          MVI    M,0A0H  PUT CURSOR ON SCREEN
0675          SHLD   VDMP    SAVE POINTER TO DATA LINE
0680          XRA    A
0685          OUT    0C8H    CLEAR VDM-1 STATUS LATCH
0690          JMP    SPEED
0695 VDMP     DS     02H     CURRENT CURSOR POSITION
0700 NEWSP    DW     00H     CURRENT OUTPUT SPEED
0705 *
0710 *
0715 *  THESE ARE AUXILLIARY ROUTINES
0720 *  FOR INPUT AND OUTPUT OF THE
0725 *  DISPLAY SPEED INFORMATION FROM THE
0730 *  KEYBOARD.  THEY ARE NORMALLY
0735 *  PART OF THE CALLING PROGRAM.
0740 *
0745 *
0750 STATS    IN     00H     INPUT FROM STATUS PORT
0755          ANI    40H     LOOK AT KEYBOARD STATUS BIT
0760          RET    .
0765 DATA     IN     01H     INPUT FROM DATA PORT
0770          ANI    7FH     REMOVE PARITY BIT
0775          RET    .
```

$1250 MICROKIT ICE UNIT

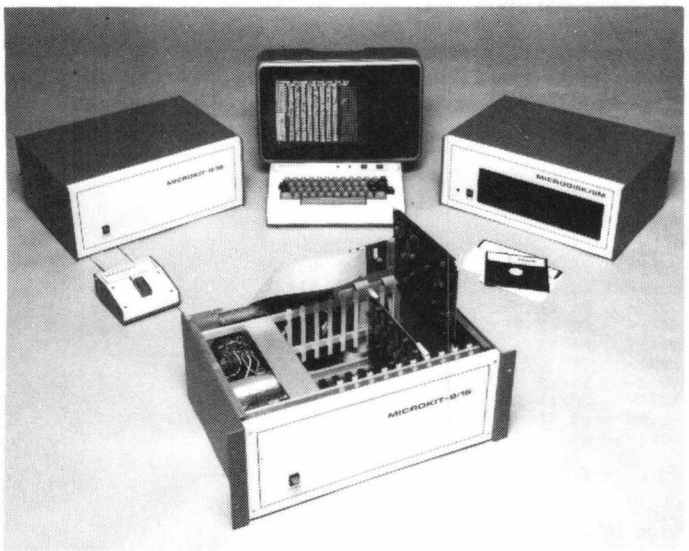
news release *Received: 77 Jul 5*

Los Angeles, California – Microkit, Inc., a leading manufacturer of ready-to-use, low-cost, universal microcomputer systems, has announced availability of a new in-circuit emulator/EPROM programmer.

The new MICROEMULATOR extends the editing, assembling and debugging capabilities of Microkits's 8080/6800/Z-80 Product Development Systems directly into a prototype or production system. The MICROEMULATOR probe plugs directly into a prototype's CPU socket and allows debugging of the prototype in its own environment. Programs residing in the microcomputer's RAM memory can execute and access the memory and I/O devices, emulating actual usage. In addition to basic monitor commands, provision is made for enabling emulation mode, single step and trace execution, hardware breakpoints and 2708/2704 EPROM programming.

Applications include hardware/software development and integration, production test and depot maintenance of microprocessor systems. MICROEMULATORS are offered for 8080, 6800 or Z-80 microprocessors and are compatible with any of Microkit's tape-based or disk-based systems, including high-speed QUICKRUN "in-memory" operating systems which provide the only co-resident assembly and interactive debugging system in the industry.

The complete system package consisting of an M8-40 MICROEMULATOR and M8-41 Debug and EPROM programmer is priced at $1250.00. Availability is from stock. Manufacturer: MICROKIT, INC., 11205 So. La Cienega Blvd., Los Angeles, CA 90045. Sales Contact: Bob Schaaf (213) 641-7700.



MATHPAC: A KIMATH SUPPLEMENT

by John Eaton
435 W. Padre #W10
Santa Barbara, CA 93105

Received: 77 Nov 28

The MOS Technology KIMATH program is an arithmetic program that handles 16 digit floating point operations. It can add, subtract, multiply and divide any two 16 digit numbers. KIMATH handles numbers in a BCD format; there is no conversion to any type of binary number. The numbers are stored in registers that are 18 bytes wide. The first byte in each register is the sign byte. Bit 7 is the sign of the mantissa and bit 6 is the sign of the exponent (0=+,1=–). The next 16 bytes contain the mantissa with one digit per byte. Each mantissa byte contains a BCD digit in the lower four bits of the byte with 0 in the upper four. The 18th byte contains the exponent in BCD which can be from 00 to 99. The entire register is in scientific notation so that the mantissa must be between 1 and 10.

The user has three registers in page 02: Rx, Ry, and Rz. When you call any operation, KIMATH will take the values from Rx and Ry and perform the operation and return the result to Rz.

KIMATH provides several routines for moving data from these registers to others in page 02. The Move routines are all labeled after the general format MVSD. The S is the source register and the D is the destination register. If you see a "JSR MVZX" it means that the number in Rz is moved into Rx.

The 18 byte format is a very inefficient way to store a large number of variables in memory. KIMATH has routines (PSTRES, PGTARG) that can store or recall a 16 digit number from user memory and only require 10 bytes per number.

KIMATH has several functions but most of them are limited to a range of 0 to 1. MATHPAC expands these functions to a far greater range and adds several other useful functions.

KIMATH is designed to increase the power of a 6502 system. Although capable of 16 digit operations, it is not able to directly drive a user's I/O device. Extra routines are required to take the power of KIMATH and give it to the user. MATHPAC does just that. It takes the user's ASCII I/O device and turns it into a scientific calculator.

Using MATHPAC

MATHPAC was designed to conform to the user's format instead of forcing the user to conform to the computer's. To use MATHPAC you simply type in commands in the form of assignment statements. For example if you type: @=1.234/56.78 followed by a carriage return then MATHPAC will figure out the value of the right side of the "=" and display it. (@ indicates display.) There are no restrictions on entering data since the program is designed to accept data in the same manner as a scientific calculator. You can use a "=" to enter a negative number and an "E" if you want to use scientific notation. An example is if you want to multiply 250 microamps by 11.75K ohms you type: @=250E–6*11.75E3. The program will respond by outputting 2.9375. With MATHPAC you can add (+), subtract (–), multiply (*), divide (/) or raise to a power (↑) any two 16 digit numbers.

Most scientific calculators have some form of memory where you can store results for later use. MATHPAC can store up to 26 sixteen digit numbers that can be identified by the letters A–Z. This is done simply by using a letter instead of the "@" in an assignment statement. Once a letter is defined it can be used in other calculations wherever a number is called for. For example if you type:

```
A=1234    cr*
B=345     cr
C=A–B     cr
@=C       cr          *carriage return
```

The computer will respond by outputting 889. It will also leave the numbers stored under the labels of A, B, And C. These letters can now be used in place of numbers in any other calculations.

Table 1. MATHPAC Functions:

Code	Address	Result
ABS	3513	Absolute value of Arg is found. No limitation on size.
ACS	343E	Arc cosine of arg is found. Result is in degrees from 0 to 180. Arg should be less than or equal to ±1.
ALG	32F9	Antilog base 10 is found. Arg should be greater than -99 and less than +100.
ASN	3454	Arcsin of arg is found. Result in degrees from -90 to +90. Arg should be less than or equal to ±1.
ATN	346D	Arc tangent of arg is found. Result is in degrees. No limit on size of arg. Result is from -90 to +90.
COS	3399	Cosine of arg(degrees) is found. No limit on size of arg.
DEG	351E	Argument in Radians is converted into degrees. No limit on the size of arg.
INV	354A	1/Arg is found. No limit on size of arg.
LOG	3210	Log base 10 of arg is found. Arg must be positive and non zero.
RAD	3526	Argument in Degrees is converted into radians. No limit on the size of arg.
SIN	3354	Sin of arg(degrees) is found. No limit on the size of arg.
SQR	3272	Square root of the absolute value of arg is found. No limit on the size of arg.
TAN	3370	Tangent of arg(degrees) is found. No limit to size of arg.

MATHPAC also has internal functions. Table 1 lists 13 functions and their ranges. You will notice that most of them are extended over the entire range of the KIMATH registers whenever possible (0 to ± 9.999999999999999 E±99). To call a function you simply type its three letter code and the argument in parenthesis. The argument can be either a number or a letter. Example:

```
@=SQR (23.45) cr
A=TAN (9.789E23) cr
B=LOG(A) cr
```

Each line typed into MATHPAC must contain one assignment per line. It can be one of three types: (1) SIMPLE assignment such as @=A, A=34 or B=A. (2) FUNCTION assignment such as B=SIN (A) (3) OPERATIVE assignment such as @=34.5/67, A=56+89, or C=A–3.

Two variable operative assignments cannot be mixed with functions on the same line. Use letters to store the results of calculations if mixed operations are required. If the program does not understand or is unable to carry out your command then it will respond with a "WHAT".

Placing MATHPAC in Your System

Your system must have at least 5K of memory in addition to I/O routines. 1K of RAM is required from 0000 to 03FF. MATHPAC itself needs 2K from 3000 to 37FF. KIMATH needs 2K from F800 to FFFF. Refer to Table 2 for a breakdown of the memory used. The entire system will work in a KIM-1 with an additional 4K of memory. The user must obtain his own copy of MOS Technology's KIMATH program and have single character input and output routines that pass data thru the accumulator.

Table 2. Memory Requirements:

0000--001C	Page zero use
0040--007F	I/O buffer for ASCII characters
0200--029A	KIMATH Page 02 requirements
0300--03FF	Number storage
3000--37FF	MATHPAC
F800--FFFF	KIMATH

All the codes used by MATHPAC are ASCII. Place the address of your character input routine in the jump command at 3600. The address of the character output routine will go in the jump command at 3603. Address 3606 must contain either an OD (carriage return) or an OA (linefeed). If your terminal does not have an automatic linefeed with carriage return then use an OA, otherwise use an OD.

Page 03 is where MATHPAC stores its data and must be cleared to 00. The amount of memory used is variable. Place a block of 11 bytes of FF where you want the memory to end. If you fill the last 11 bytes of page 03 with FF then MATHPAC will be able to store 22 sixteen digit numbers.

The byte at address 0000 must be set to 10. This sets the length of all operations to 16 digits. All functions are automatically rounded off to 8 digits and all other operations are rounded off to 14 digits. You must start the MATHPAC program at 3607.

Expanding the Functions

You may want to add some of your own special functions to MATHPAC. All functions take their argument from KIMATH's Rx register and leave the result in Rz. If you have a routine that does this then you may add it to MATHPAC by placing its starting address in TAB2. If you read through TAB1 and TAB2 you will see that there are three functions (FNA, FNB and FNC) that call the KIMATH routine MVXZ(FCFO). If you substitute your starting address for the first address of FCFO then calling FNA will call your function. If you want to get fancy and give it its own three letter code then you will have to reassemble both tables and insert your code in alphabetical order.

Extra Uses for MATHPAC

KIMATH is useful when it can be called by other programs to perform arithmetic operations. It consists of a series of routines and is useful to any of your other programs. MATHPAC has many similar uses when called on as subroutines. Tables 1 and 3 show many of the different routines that can be called by the user programs to perform operations on the KIMATH registers.

Table 3. MATHPAC support routines:

Name	Address	Result
PACKER	3000	Packs the ASCII data at ARGYL,ARGYH into Ry. No restrictions on format.
UNPACK	30F9	Converts Rz into readable number and stores it at RES,RES+1
STORE	3182	Stores Rz in memory under the ID in the accumulator. Returns with FF in accumulator if there is not enough room.
RECALL	31BB	Finds number in memory with ID in accumulator. Loads it into Ry.Sets accumulator to FF if number not in memory.
FORGET	31D8	Erases number from memory,ID from accumulator.
INT	329D	Largest interger less than or equal to Rx is found.
ONEX	350A	Rx is set to one.
PIE	3553	Ry is set equal to Pi
HEXDEC	3558	CNT (0003) is converted from a HEX number to a BCD number.
SETCON	3568	Constant from table at 37C0 is loaded into Ry. Accumulator determines which one.
CHOPIT	3575	Rz is scanned and PREC is set to cover only non zero digits. -0 is also corrected for.
PACADD	3589	Y index is added to ARGYL,ARGYH
RNDF	3597	Rx is rounded off the the lenght in the X index register.

After you use MATHPAC and KIMATH for a while you may notice a quirk in the system. If you type @=.5–0 the computer will respond with –9.5. Not quite the right answer. This is caused by an error in KIMATH that affects the subtraction of zero from a positive number that is less than one. If you have KIMATH in RAM then you can correct it by changing FCBB to DO and FCBD to FO.

Table 4. Assignment Statement Format:

@ (display) A-Z (Save in memory)	=	Simple assignment single letter or number.
		Function Arg in parenthesis can be number or letter
		Operation two variables can be either letter, number or both

```
                ; Calculator supplememnt for KIMATH
                ; see KIMATH manual for undefined labels
0000 10         N                       set to 10 or lenght
0017            PER
0018            QUADCT
0019            ID
001A            SIGN
001B            CAL1
001C            CAL2
0040            LR                      I/O buffer 64 Bytes
0300            ; Page 03 used for numeric storage.
                ; clear all bytes to 00. Set last 11
                ; bytes of page 03 ( or first 11 of
                ; page 04)to FF.

3000 20 7C FD   PACKER  JSR CLRY        routine to load  raw
3003 A2 00              LDX#00          number at (ARGYL,
3005 A0 00              LDY#00          ARGYH) into Ry.
3007 84 17              STY PER
3009 84 03              STY CNT
300B 84 1A              STY SIGN
300D B1 08              LDA(ARGYL),Y    1st character
300F C9 2B              CMP#2B          "+"
3011 F0 08              BEQ PACK1
3013 C9 2D              CMP#2D          "-"
3015 D0 07              BNE PACK2
3017 A9 80              LDA#80
3019 85 1A              STA SIGN        set sign neg
301B C8         PACK1   INY
301C B1 08              LDA(ARGYL),Y
301E C9 2E      PACK2   CMP#2E          "."
3020 D0 0F              BNE PACK4
3022 A9 40              LDA#40          decimal point found
3024 24 17              BIT PER
3026 30 05              BMI PACK3       stop counting exponent
3028 05 1A              ORA SIGN        start counting down
302A 85 1A              STA SIGN
302C 0A                 ASL A
302D 85 17      PACK3   STA PER
302F D0 EA              BNE PACK1       unconditional
3031 C9 30      PACK4   CMP#30          test for 0-9
3033 90 2F              BCC PACK8       non-digit
3035 C9 3A              CMP#3A
3037 B0 2B              BCS PACK8       non-digit
3039 24 17              BIT PER
303B 10 0D              BPL PACK5       not counting exp
303D E6 03              INC CNT
303F 70 15              BVS PACK7       counting up
3041 C9 30              CMP#30          zero?
3043 F0 D6              BEQ PACK1       place setting zero
3045 48                 PHA
3046 A9 40              LDA#40          stop counting
3048 D0 09              BNE PACK6       unconditional
304A 70 0A      PACK5   BVS PACK7       counting stopped
304C C9 30              CMP#30
304E F0 CB              BEQ PACK1       leading zero
3050 48                 PHA
3051 A9 C0              LDA#C0          start counting up
3053 85 17      PACK6   STA PER
3055 68                 PLA
3056 29 0F      PACK7   AND#0F          mask off digit
3058 9D 48 02           STA SY+1,X      store in Ry
305B E8                 INX
305C E0 11              CPX#11          16 digits?
305E 90 BB              BCC PACK1       not yet
3060 A2 10              LDX#10          clamp X to 16
3062 D0 B7              BNE PACK1       unconditional
3064 8A         PACK8   TXA             X=0?
3065 D0 04              BNE PACK9       no
3067 86 1A              STX SIGN
3069 86 03              STX CNT
306B 20 58 35   PACK9   JSR HEXDEC      convert exp to BCD
306E 8D 58 02   EXOT    STA EY
3071 B1 08              LDA(ARGYL),Y
3073 C9 45              CMP#45          "E"
3075 F0 06              BEQ EXP
3077 A5 1A              LDA SIGN
3079 8D 47 02           STA SY
307C 60                 RTS
307D A5 03      EXP     LDA CNT         old exp
307F 48                 PHA
3080 A5 1A              LDA SIGN
3082 48                 PHA
3083 29 80              AND#80          preserve man sign
3085 85 1A              STA SIGN        new sign
3087 A9 00              LDA#00
3089 85 03              STA CNT         new exp
308B C8                 INY
308C B1 08              LDA(ARGYL),Y
308E C9 2B              CMP#2B          "+"
3090 F0 0A              BEQ EXP1
3092 C9 2D              CMP#2D          "-"
3094 D0 09              BNE EXP2
3096 A9 40              LDA#40          new exp sign neg
3098 05 1A              ORA SIGN
309A 85 1A              STA SIGN
309C C8         EXP1    INY
309D B1 08              LDA(ARGYL),Y
309F C9 30              CMP#30          test for 0-9
30A1 90 15              BCC EXP3        non digit
30A3 C9 3A              CMP#3A
30A5 B0 11              BCS EXP3        non digit
30A7 29 0F              AND#0F          mask off digit
30A9 06 03              ASL CNT
30AB 06 03              ASL CNT
30AD 06 03              ASL CNT
30AF 06 03              ASL CNT         shift exponent
30B1 05 03              ORA CNT         combine with digit
30B3 85 03              STA CNT
30B5 38                 SEC
30B6 B0 E4              BCS EXP1        unconditional
30B8 F8         EXP3    SED             adjust sign and exp
30B9 68                 PLA             old sign
30BA 48                 PHA
30BB 45 1A              EOR SIGN        test signs of the
30BD 85 17              STA PER         two exp's to see
30BF 24 17              BIT PER         if they are the same
30C1 50 21              BVC EXP6        sign's same
30C3 68                 PLA             old sign
30C4 85 17              STA PER
30C6 68                 PLA             old exp
30C7 C5 03              CMP CNT
30C9 90 09              BCC EXP4        new exp gtr
30CB E5 03              SBC CNT         difference of exp's
30CD 48                 PHA             adjusted exp
30CE A5 17              LDA PER         old sign
30D0 48                 PHA             adjusted sign
30D1 38                 SEC
30D2 B0 0C              BCS EXP5        unconditional
30D4 E5 03      EXP4    SBC CNT         difference of exp's
30D6 85 03              STA CNT         compensate subtracting
30D8 A9 00              LDA#00          larger number from
30DA E5 03              SBC CNT         small by subtracting
30DC 48                 PHA             from zero
30DD A5 1A              LDA SIGN
30DF 48                 PHA             adjusted sign
30E0 A9 00      EXP5    LDA#00
30E2 85 03              STA CNT
30E4 18         EXP6    CLC
30E5 68                 PLA             sign
30E6 85 1A              STA SIGN
30E8 68                 PLA             exponent
30E9 65 03              ADC CNT
30EB 48                 PHA
30EC D8                 CLD
30ED D0 06              BNE EXP7        exp not zero
30EF A9 BF              LDA#BF
30F1 25 1A              AND SIGN
30F3 85 1A              STA SIGN
30F5 68         EXP7    PLA
30F6 4C 8E 30           JMP EXOT
30F9 A0 6A 02   UNPACK  LDA EZ          routine to unpack
30FC 85 03              STA CNT         Rz and store at
30FE 20 C3 FB           JSR DECHEX      (RES,RES+1)
3101 A0 00              LDY#00
3103 2C 59 02           BIT SZ
3106 10 05              BPL UNPAC1      positive number
3108 A9 2D              LDA#2D          "-"
310A 91 0A              STA(RES),Y
310C C8                 INY
310D A2 00      UNPAC1  LDX#00
310F A5 03              LDA CNT
3111 C9 10              CMP#10          exp gtr 15
3113 B0 3B              BCS UNPAC7      use scientific notation
3115 2C 59 02           BIT SZ
3118 50 0E              BVC UNPAC3      exp is positive
311A A9 2E              LDA#2E          decimal point
311C 91 0A              STA(RES),Y
311E A9 30              LDA#30          zero
3120 C8         UNPAC2  INY             display place setting 0's
3121 91 0A              STA(RES),Y
3123 C6 03              DEC CNT
3125 10 F9              BPL UNPAC2
3127 88                 DEY
3128 BD 5A 02   UNPAC3  LDA SZ+1,X      fetch digit
312B 09 30              ORA#30          convert to ASCII
312D 91 0A              STA(RES),Y
312F E8                 INX
3130 C8                 INY
3131 24 03              BIT CNT
3133 30 09              BMI UNPAC4
3135 C6 03              DEC CNT
3137 10 05              BPL UNPAC4
3139 A9 2E              LDA#2E          decimal point
313B 91 0A              STA(RES),Y
313D C8                 INY
313E E4 10      UNPAC4  CPX PREC        all digits moved?
3140 D0 E6              BNE UNPAC3
3142 24 03              BIT CNT
3144 30 09              BMI UNPAC6
3146 A9 30              LDA#30
3148 91 0A      UNPAC5  STA(RES),Y      trailing zero's
314A C8                 INY
314B C6 03              DEC CNT
314D 10 F9              BPL UNPAC5
314F 60         UNPAC6  RTS
3150 A9 00      UNPAC7  LDA#00          scientific notation
3152 85 03              STA CNT
3154 20 28 31           JSR UNPAC3
3157 A9 20              LDA#20          blank
3159 91 0A              STA(RES),Y
315B C8                 INY
315C A9 45              LDA#45          "E"
315E 91 0A              STA(RES),Y
3160 C8                 INY
3161 2C 59 02           BIT SZ
```

```
3164 50 05            BVC UNPAC8      positive exponent
3166 A9 2D            LDA#2D          "-"
3168 91 0A            STA(RES),Y
316A C8               INY
316B AD 6A 02  UNPAC8 LDA EZ
316E 4A               LSR A
316F 4A               LSR A
3170 4A               LSR A
3171 4A               LSR A
3172 09 30            ORA#30          convert to ASCII
3174 91 0A            STA(RES),Y
3176 C8               INY
3177 AD 6A 02         LDA EZ
317A 29 0F            AND#0F
317C 09 30            ORA#30          convert to ASCII
317E 91 0A            STA(RES).Y
3180 C8               INY
3181 60               RTS

                ; routines to store ad recall numbers.
                ; numbers are taken from Rz and stored
                ; in page 03. Numbers are recalled to Ry.
3182 20 E2 31 STORE   JSR SRCH
3185 D0 0D            BNE STOR1       ID already in memory
3187 A5 19            LDA ID
3189 48               PHA
318A A9 00            LDA#00
318C 20 E2 31         JSR SRCH        look for empty cell
318F F0 26            BEQ STOR2       no room in page 03
3191 68               PLA
3192 91 0C            STA(PTR),Y      set ID in pg 03
3194 A5 0A     STOR1  LDA RES
3196 48               PHA
3197 A5 0B            LDA RES+1
3199 48               PHA
319A A9 01            LDA#01
319C 20 04 32         JSR ADDM        add one to address
319F A5 0C            LDA PTR
31A1 85 0A            STA RES
31A3 A5 0D            LDA PTR+1
31A5 85 0B            STA RES+1
31A7 A5 00            LDA N
31A9 85 10            STA PREC
31AB 20 3C FE         JSR PSTRES      Move Rz into Pg 03
31AE 68               PLA
31AF 85 0B            STA RES+1
31B1 68               PLA
31B2 85 0A            STA RES
31B4 A5 19            LDA ID
31B6 60               RTS
31B7 68        STOR2  PLA
31B8 A9 FF            LDA#FF          No room in pg 3
31BA 60               RTS
31BB 20 E2 31  RECALL JSR SRCH
31BE F0 17            BEQ RECAL1       not in memory
31C0 A9 01            LDA#01
31C2 20 04 32         JSR ADDM         add one to address
31C5 A5 00            LDA N            recall number into Ry
31C7 4A               LSR A
31C8 69 01            ADC#01
31CA 85 04            STA LENGHT
31CC 20 87 FD         JSR CLRZ
31CF 20 E1 FD         JSR PGTARG
31D2 20 10 FD         JSR MVZY
31D5 A5 19            LDA ID
31D7 60        RECAL1 RTS
31D8 20 E2 31  FORGET JSR SRCH
31DB F0 04            BEQ FORGE1
31DD A9 00            LDA#00
31DF 91 0C            STA(PTR),Y
31E1 60        FORGE1 RTS
31E2 D8        SRCH   CLD              search page 03 for
31E3 85 19            STA ID           ID or FF
31E5 A0 00            LDY#00
31E7 A9 02            LDA#02
31E9 85 0D            STA PTR+1
31EB A9 F5            LDA#F5
31ED 85 0C            STA PTR
31EF 20 FF 31  SRCH1  JSR ADDL
31F2 B1 0C            LDA(PTR),Y
31F4 C5 19            CMP ID
31F6 F0 04            BEQ SRCH2
31F8 C9 FF            CMP#FF
31FA D0 F3            BNE SRCH1
31FC C9 FF     SRCH2  CMP#FF
31FE 60               RTS
31FF A5 00     ADDL   LDA N            Add  lenght to address
3201 4A               LSR A
3202 69 03            ADC#03
3204 18        ADDM   CLC              add A to address
3205 65 0C            ADC PTR
3207 85 0C            STA PTR
3209 A9 00            LDA#00
320B 65 0D            ADC PTR+1
320D 85 0D            STA PTR+1
320F 60               RTS

                ; LOG base 10 of Rx is found and stored
                ; in Rz. Rx must be positive and non zero
3210 A5 00     LOGT   LDA N
3212 48               PHA              save lenght
3213 AD 35 02         LDA SX
3216 48               PHA              save sign
3217 AD 46 02         LDA EX

321A 48               PHA              save exponent
321B A9 00            LDA#00
321D 8D 35 02         STA SX
3220 8D 46 02         STA EX
3223 A9 09            LDA#09
3225 20 68 35         JSR SETCON       Ry=1/SQR(10)
3228 20 0B F9         JSR MUL
322B 20 0C FD         JSR MVZX
322E 20 E7 FA         JSR LOG
3231 20 0C FD         JSR MVZX
3234 20 7C FD         JSR CLRY
3237 A9 05            LDA#05
3239 8D 49 02         STA SY+2          Ry=+.5
323C 20 08 F8         JSR ADD
323F 20 0C FD         JSR MVZX
3242 20 7C FD         JSR CLRY
3245 68               PLA              exponent
3246 C9 10            CMP#10
3248 B0 09            BCS LOGT1        exp gtr 9
324A 29 0F            AND#0F
324C 8D 48 02         STA SY+1
324F A9 00            LDA#00
3251 F0 10            BEQ LOGT2        unconditional
3253 48        LOGT1  PHA
3254 4A               LSR A
3255 4A               LSR A
3256 4A               LSR A
3257 4A               LSR A
3258 8D 48 02         STA SY+1
325B 68               PLA
325C 29 0F            AND#0F
325E 8D 49 02         STA SY+2
3261 A9 01            LDA#01
3263 8D 58 02  LOGT2  STA EY           Ry now contains exp
3266 68               PLA
3267 0A               ASL A            adjust sign
3268 8D 47 02         STA SY
326B 20 08 F8         JSR ADD
326E 68               PLA
326F 85 00            STA N            lenght
3271 60               RTS
3272 20 13 35  SQRT    JSR ABS          square root routine
3275 20 A6 FC         JSR XZTST
3278 D0 01            BNE SQRT1
327A 60               RTS
327B 20 18 FD  SQRT1  JSR MVZN
327E 20 14 FD         JSR MVZM
3281 AD 46 02         LDA EX
3284 85 03            STA CNT
3286 20 C3 FB         JSR DECHEX       exp now hex
3289 4A               LSR A            divide by two
328A D0 02            BNE SQRT2
328C A9 01            LDA#01
328E 85 03     SQRT2  STA CNT
3290 20 58 35         JSR HEXDEC       exp now BCD
3293 8D 7C 02         STA EM
3296 A9 07            LDA#07
3298 85 01            STA NKON
329A 4C B5 FA         JMP SQRT0
                ; routine to find the largest interger
                ; less than or equal to Rx.
329D A5 00     INT    LDA N
329F 48               PHA              save lenght
32A0 AD 35 02         LDA SX
32A3 48               PHA              save sign
32A4 29 7F            AND#7F
32A6 8D 35 02         STA SX           set positive
32A9 20 F4 FC         JSR MVXM
32AC 2C 35 02         BIT SX
32AF 50 03            BVC INT1         Rx gtr than one
32B1 20 71 FD         JSR CLRX         Rx=0
32B4 AD 46 02  INT1   LDA EX
32B7 C9 15            CMP#15
32B9 90 02            BCC INT2         exp less 15
32BB A9 15            LDA#15
32BD 85 03     INT2   STA CNT
32BF 20 C3 FB         JSR DECHEX       exp now hex
32C2 85 00            STA N
32C4 E6 00            INC N
32C6 20 7C FD         JSR CLRY
32C9 20 87 FD         JSR CLRZ
32CC 20 08 F8         JSR ADD
32CF 68               PLA              sign
32D0 10 23            BPL INT4
32D2 20 0C FD         JSR MVZX
32D5 20 20 FD         JSR MVMY
32D8 A9 10            LDA#10
32DA 85 00            STA N
32DC 20 00 F8         JSR SUB
32DF 20 0C FD         JSR MVZX
32E2 20 00 FD         JSR MVYZ
32E5 20 A6 FC         JSR XZTST
32E8 F0 06            BEQ INT3
32EA 20 0A 35         JSR ONEX
32ED 20 08 F8         JSR ADD
32F0 A9 80     INT3   LDA#80
32F2 8D 59 02         STA SZ
32F5 68        INT4   PLA
32F6 85 00            STA N
32F8 60               RTS
                ; antilog base 10 routine. Rx must be
                ; gtr than -99 and less than +100
32F9 2C 35 02  ALOG   BIT SX
```

```
32FC 70 12          BVS ALOG2       Rx less than 1
32FE A5 46 02       LDA EX
3301 C9 02          CMP#02
3303 90 0B          BCC ALOG2       Exp less 2
3305 20 D2 FC ALOG1 JSR INFIN
3308 A5 35 02       LDA SX
330B 4A             LSR A
330C 8D 59 02       STA SZ
330F 60             RTS
3310 20 F8 FC ALOG2 JSR MVXN
3313 20 9D 32       JSR INT
3316 20 0C FD       JSR MVZX
3319 AE 6A 02       LDX EZ
331C E0 02          CPX#02
331E F0 E5          BEQ ALOG1       X=-100
3320 A5 00          LDA N
3322 48             PHA
3323 B5 59 02       LDA SZ,X
3326 0A             ASL A
3327 0A             ASL A
3328 0A             ASL A
3329 0A             ASL A
332A 1D 5A 02       ORA SZ+1,X
332D 85 17          STA PER
332F 48             PHA             save exponent
3330 A5 59 02       LDA SZ
3333 4A             LSR A           adjust sign
3334 48             PHA             save sign
3335 20 2C FD       JSR MVNX
3338 A5 17          LDA PER
333A F0 09          BEQ ALOG3       exp=00
333C 20 10 FD       JSR MVZY
333F 20 00 F8       JSR SUB
3342 20 0C FD       JSR MVZX
3345 20 41 FB ALOG3 JSR TENX
3348 68             PLA
3349 8D 59 02       STA SZ
334C 68             PLA
334D 8D 6A 02       STA EZ
3350 68             PLA
3351 85 00          STA N
3353 60             RTS
3354 20 A8 33 SIN   JSR TRIG5       SIN(Rx) found and
3357 20 08 F8       JSR ADD         placed in Rz
335A 20 76 33 TRIG1 JSR TRIG4
335D A5 18    TRIG2 LDA QUADCT
335F F0 0C          BEQ TRIG3
3361 C9 03          CMP#03
3363 F0 08          BEQ TRIG3
3365 A5 59 02       LDA SZ
3368 49 80          EOR#80
336A 8D 59 02       STA SZ
336D 4C 75 35 TRIG3 JMP CHOPIT
3370 20 A8 33 TAN   JSR TRIG5       TAN(Rx) found and
3373 20 00 F8       JSR SUB         placed in Rz
3376 20 10 FD TRIG4 JSR MVZY
3379 20 1C FD       JSR MVMX
337C 20 16 FA       JSR DIVIDE
337F A5 6A 02       LDA EZ
3382 C9 06          CMP#06
3384 90 E7          BCC TRIG3
3386 2C 59 02       BIT SZ
3389 70 E2          BVS TRIG3
338B A5 59 02       LDA SZ
338E 48             PHA
338F 20 D2 FC       JSR INFIN
3392 68             PLA
3393 8D 59 02       STA SZ
3396 4C 75 35       JMP CHOPIT
3399 20 A8 33 COS   JSR TRIG5       COS(Rx) found and
339C 20 00 F8       JSR SUB         placed in Rz
339F 20 14 FD       JSR MVZM
33A2 20 00 F8       JSR SUB
33A5 4C 5A 33       JMP TRIG1
33A8 A9 FF    TRIG5 LDA#FF          Rx can be any value
33AA 85 18          STA QUADCT
33AC 2C 35 02 TRIG6 BIT SX
33AF 30 0C          BMI TRIG7
33B1 20 18 34       JSR Y360        angle is pos
33B4 20 00 F8       JSR SUB
33B7 20 0C FD       JSR MVZX
33BA 4C AC 33       JMP TRIG6
33BD 20 18 34 TRIG7 JSR Y360        angle is neg
33C0 20 08 F8       JSR ADD
33C3 20 0C FD       JSR MVZX
33C6 2C 35 02       BIT SX
33C9 30 F2          BMI TRIG7
33CB 20 13 34 TRIG8 JSR Y90
33CE 20 00 F8       JSR SUB
33D1 20 0C FD       JSR MVZX
33D4 E6 18          INC QUADCT
33D6 2C 35 02       BIT SX
33D9 10 F0          BPL TRIG8
33DB A5 18          LDA QUADCT      angle between -90 and 0
33DD 4A             LSR A
33DE B0 09          BCS TRIG9
33E0 20 13 34       JSR Y90
33E3 20 08 F8       JSR ADD
33E6 20 0C FD       JSR MVZX
33E9 20 13 34 TRIG9 JSR Y90
33EC 20 16 FA       JSR DIVIDE
33EF 20 0C FD       JSR MVZX
33F2 A5 00          LDA N
33F4 48             PHA
33F5 20 5C FB       JSR TANX        Rz=TAN(X/2)
33F8 68             PLA
33F9 85 00          STA N
33FB 20 0C FD       JSR MVZX
33FE 20 10 FD       JSR MVZY
3401 20 0B F9       JSR MUL
3404 20 14 FD       JSR MVZM
3407 20 08 F8       JSR ADD
340A 20 20 FD       JSR MVMY
340D 20 14 FD       JSR MVZM
3410 4C 0A 35       JMP ONEX
3413 A9 1E    Y90   LDA#1E
3415 4C 68 35       JMP SETCON
3418 20 7C FD Y360  JSR CLRY
341B A9 03          LDA#03
341D 8D 48 02       STA SY+1
3420 A9 06          LDA#06
3422 8D 49 02       STA SY+2
3425 2C 35 02       BIT SX
3428 70 0D          BVS Y360A
342A F8             SED
342B AD 46 02       LDA EX
342E F0 07          BEQ Y360A
3430 38             SEC
3431 E9 01          SBC#01
3433 C9 02          CMP#02
3435 B0 02          BCS Y360B
3437 A9 02    Y360A LDA#02
3439 8D 58 02 Y360B STA EY
343C D8             CLD
343D 60             RTS
                    ; arctrig routines give results in degrees
343E 20 C9 34 ACOS  JSR ARCSET
3441 2C 47 02       BIT SY
3444 10 14          BPL ASIN1       angle in 1st quad
3446 20 5A 34       JSR ASIN1       angle in 2nd quad
3449 20 0C FD       JSR MVZX
344C A9 1B          LDA#1B
344E 20 68 35       JSR SETCON      Ry=180
3451 4C 08 F8       JMP ADD
3454 20 C9 34 ASIN  JSR ARCSET
3457 20 BF FC       JSR XSY
345A AD 35 02 ASIN1 LDA SX
345D 29 80          AND#80
345F 48             PHA
3460 20 16 FA       JSR DIVIDE
3463 20 0C FD       JSR MVZX
3466 68             PLA
3467 0D 35 02       ORA SX
346A 8D 35 02       STA SX
346D A5 00    ATAN  LDA N
346F 48             PHA
3470 AD 35 02       LDA SX
3473 48             PHA
3474 29 7F          AND#7F
3476 8D 35 02       STA SX
3479 20 BF FC       JSR XSY
347C 20 0A 35       JSR ONEX
347F 20 08 F8       JSR ADD
3482 20 BF FC       JSR XSY
3485 20 16 FA       JSR DIVIDE
3488 20 0C FD       JSR MVZX
348B 20 A6 FC       JSR XZTST
348E F0 26          BEQ ATAN2
3490 2C 35 02       BIT SX
3493 50 0A          BVC ATAN1
3495 AD 46 02       LDA EX
3498 D0 05          BNE ATAN1
349A A9 99          LDA#99
349C 8D 46 02       STA EX
349F 20 78 FB ATAN1 JSR ATANX
34A2 68             PLA
34A3 48             PHA
34A4 29 40          AND#40
34A6 D0 0E          BNE ATAN2
34A8 20 0C FD       JSR MVZX
34AB A9 12          LDA#12
34AD 20 68 35       JSR SETCON      Ry=Pi/2
34B0 20 BF FC       JSR XSY
34B3 20 00 F8       JSR SUB
34B6 68       ATAN2 PLA             sign
34B7 29 80          AND#80
34B9 0D 59 02       ORA SZ
34BC 8D 59 02       STA SZ
34BF 20 0C FD       JSR MVZX
34C2 20 1E 35       JSR DEG         convert to degrees
34C5 68             PLA
34C6 85 00          STA N
34C8 60             RTS
34C9 2C 35 02 ARCSET BIT SX
34CC 70 17          BVS ARC2        Rx less one
34CE AD 36 02       LDA SX+1
34D1 48             PHA
34D2 AD 35 02       LDA SX
34D5 48             PHA
34D6 20 71 FD       JSR CLRX
34D9 68             PLA
34DA 8D 35 02       STA SX
34DD 68             PLA
34DE F0 02          BEQ ARC1
34E0 A9 01          LDA#01
34E2 8D 36 02 ARC1  STA SX+1
```

```
34E5 20 EC FC   ARC2    JSR MVXY        -1 ls X ls +1
34E8 20 F0 FC           JSR MVXZ
34EB A9 01              LDA#01
34ED 20 82 31           JSR STORE
34F0 20 0B F9           JSR MUL         X^2
34F3 20 10 FD           JSR MVZY
34F6 20 0A 35           JSR ONEX
34F9 20 00 F8           JSR SUB         1-X^2
34FC 20 0C FD           JSR MVZX
34FF 20 72 32           JSR SQRT        SQR(1-X^2)
3502 20 0C FD           JSR MVZX
3505 A9 01              LDA#01
3507 4C BB 31           JMP RECALL       Ry * ARG
                ;
350A 20 71 FD   ONEX    JSR CLRX
350D A9 01              LDA#01
350F 8D 36 02           STA RX+1        Rx=1.000
3512 60                 RTS
                ;
3513 AD 35 02   ABS     LDA SX          Absolute value
3516 29 7F              AND#7F
3518 8D 35 02           STA SX
351B 4C F0 FC           JMP MVXZ
                ;
351E A9 00      DEG     LDA#00          convert to deg
3520 20 68 35           JSR SETCON      Pi/180
3523 4C 16 FA           JMP DIVIDE
                ;
3526 A9 00      RAD     LDA#00          convert to rad
3528 20 68 35           JSR SETCON      Pi/180
352B 4C 0B F9           JMP MUL
                ;
352E 20 00 FD   XRY     JSR MVYZ        raise Rx to Ry
3531 A9 01              LDA#01
3533 20 82 31           JSR STORE
3536 20 10 32           JSR LOGT
3539 20 0C FD           JSR MVZX
353C A9 01              LDA#01
353E 20 BB 31           JSR RECALL
3541 20 0B F9           JSR MUL
3544 20 0C FD           JSR MVZX
3547 4C F9 32           JMP ALOG
                ;
354A 20 EC FC   INV     JSR MVXY        find 1/Rx
354D 20 0A 35           JSR ONEX
3550 4C 16 FA           JMP DIVIDE
                ;
3553 A9 21      PIE     LDA#21          set Ry=Pi
3555 4C 68 35           JMP SETCON
                ;
3558 F8         HEXDEC  SED             convert CNT from
3559 E6 03              INC CNT         HEX to BCD
355B A9 99              LDA#99
355D 18         HEX1    CLC
355E 69 01              ADC#01
3560 C6 03              DEC CNT
3562 D0 F9              BNE HEX1
3564 85 03              STA CNT
3566 D8                 CLD
3567 60                 RTS
                ;
3568 85 01      SETCON  STA NKON        load constant in Ry
356A A9 C0              LDA#C0
356C 85 0E              STA KON
356E A9 37              LDA#37
3570 85 0F              STA KONH
3572 4C 92 FD           JMP LOOKUP
                ;
3575 A6 00      CHOPIT  LDX N           remove unnedded 0's
3577 Bd 59 02   CHOP1   LDA SZ,X        by adjusting PREC
357A D0 0A              BNE CHOP2
357C CA                 DEX
357D D0 F8              BNE CHOP1
357F 8E 59 02           STX SZ          man=0,clear sign,exp
3582 8E 6A 02           STX EZ
3585 E8                 INX
3586 86 10      CHOP2   STX PREC
3588 60                 RTS
                ;
3589 98         PACADD  TYA             add Y to ARGY
358A D8                 CLD
358B 18                 CLC
358C 65 08              ADC ARGYL
358E 85 08              STA ARGYL
3590 A9 00              LDA#00
3592 65 09              ADC ARGYH
3594 85 09              STA ARGYH
3596 60                 RTS
3597 A5 00      RNDF    LDA N           round off routine
3599 48                 PHA             round off to X
359A A9 10              LDA#10
359C 85 00              STA N
359E 2C 35 02           BIT SX
35A1 70 05              BVS RNDF1
35A3 CD 46 02           CMP EX
35A6 90 43              BCC RNDF3
35A8 AD 35 02   RNDF1   LDA SX
35AB 48                 PHA
35AC 29 7F              ANF#7F
35AE 8D 35 02           STA SX
35B1 8A                 TXA
35B2 48                 PHA
35B3 20 EC FC           JSR MVXY
35B6 20 0A 35           JSR ONEX
35B9 68                 PLA
35BA 48                 PHA
35BB AA                 TAX
35BC A9 05              LDA#05
35BE 9D 37 02           STA SX+2,X
35C1 20 08 F8           JSR ADD
35C4 20 10 FD           JSR MVZY
35C7 20 87 FD           JSR CLRZ
35CA 20 0A 35           JSR ONEX
35CD 20 BF FC           JSR XSY
35D0 68                 PLA
35D1 AA                 TAX
35D2 E8                 INX
35D3 86 00              STX N
35D5 20 00 F8           JSR SUB
35D8 20 0C FD           JSR MVZX
35DB 20 A6 FC           JSR XZTST
35DE F0 07              BEQ RNDF2
35E0 68                 PLA
35E1 48                 PHA
35E2 29 80              AND#80
35E4 0D 35 02           ORA SX
35E7 8D 35 02   RNDF2   STA SX
35EA 68                 PLA
35EB 20 F0 FC   RNDF3   JSR MVXZ
35EE 68                 PLA
35EF 85 00              STA N
35F1 60                 RTS
3600 4C 00 00   INVEC   JMP CHARIN      user input routine
3603 4C 00 00   OTVEC   JMP CHAROT      user output routine
3606 0D         ECHO    BYTE 0D         echo character
3607 A9 01      SCICAL  LDA#01          START OF ROUTINE
3609 85 1B              STA CAL1
360B C6 1B      BACK    DEC CAL1        backspace routine
360D 20 00 36   LOOP1   JSR INVEC
3610 C9 08              CMP#08          backspace?
3612 F0 F7              BEQ BACK        yes
3614 A6 1B              LDX CAL1        X points to open cell
3616 95 40              STA LR,X        store chars at 0040
3618 E6 1B              INC CAL1
361A C9 0D              CMP#0D          carriage return?
361C D0 EF              BNE LOOP1
361E AD 06 36           LDA ECHO
3621 20 03 36           JSR OTVEC       Echo character
3624 A5 40              LDA LR          assignment char
3626 48                 PHA
3627 A9 40              LDA#40
3629 85 08              STA ARGYL
362B 85 0A              STA RES
362D A9 00              LDA#00
362F 85 09              STA ARGYH
3631 85 0B              STA RES+1
3633 A0 02              LDY#02
3635 20 0C 37           JSR LOAD
3638 B0 12              BCS HAV1        number loaded
363A A5 43              LDA LR+3        letter found,test function
363C 20 1B 37           JSR LTRTST
363F 90 6D              BCC FUNCTN      function found
3641 A5 42              LDA LR+2
3643 20 BB 31           JSR RECALL      fetch number into Ry
3646 C9 FF              CMP#FF
3648 F0 17              BEQ WHATC       number not in memory
364A A0 03              LDY#03
364C 20 FC FC    HAV1   JSR MVYX
364F B1 08              LDA(ARGYL),Y
3651 48                 PHA             operation
3652 C9 0D              CMP#0D          carriage return
3654 F0 0D              BEQ OPS
3656 C8                 INY
3657 20 0C 37           JSR LOAD
365A B0 07              BCS OPS
365C 20 BB 31           JSR RECALL
365F C9 FF              CMP#FF
3661 F0 20      WHATC   BEQ WHAT
3663 68         OPS     PLA
3664 20 25 37           JSR OPERT       two number op
3667 20 0C FD           JSR MVZX
366A A6 00              LDX N
366C CA                 DEX
366D CA                 DEX
366E 20 97 35   OUT     JSR RNDF        result in Rz
3671 20 75 35           JSR CHOPIT      remove unwanted zero's
3674 68                 PLA             assignment
3675 C9 40              CMP#40          @?
3677 F0 0E              BEQ OUT1        display result
3679 20 1B 37           JSR LTRTST      assignment a letter?
367C B0 7B              BCS WHAT        non letter
367E 20 82 31           JSR STORE       save result
3681 C9 FF              CMP#FF
3683 F0 74      WHATD   BEQ WHAT        no room in pg 03
3685 D0 80              BNE SCICAL      unconditional
3687 20 F9 30   OUT1    JSR UNPACK      display Rz
368A A9 0D              LDA#0D          car ret
368C 91 0A              STA(RES),Y
368E C8                 INY
368F AD 06 36           LDA ECHO        echo character
3692 91 0A              STA(RES),Y
3694 A2 00      DISP    LDX#00
3696 86 1B              STX CAL1
3698 A6 1B      DISP1   LDX CAL1
369A B5 40              LDA LR,X
369C CD 06 36           CMP ECHO         last character?
```

MICROSOFT-MITS EXCLUSIVE LICENSE TERMINATED

News Release *Received: 77 Nov 23*

Microsoft's BASIC for the 8080 and Z-80, the first resident high-level language for a microprocessor, is now generally available on both a single copy and OEM basis. The BASIC became the subject of an extended legal dispute which resulted in the termination of an exclusive license to MITS, Inc.

The BASIC, best known in the field as Altair[tm] BASIC has been in use for 1½ years and has a user base of over 5000. Several software firms offer applications software written in Microsoft's BASIC. Current OEM users of the BASIC include General Electric, National Cash Register, Applied Digital Data Systems, Radio Shack and Data Terminals and Communications.

FIX FOR ELDERLY EDITOR/ASSEMBLER

Dear Dr. Dobbs, *Dated: 77 Sep 6*

Here is more data on that outstanding piece of junk, the M-T free Editor-Assembler. My version is the one distributed to clubs by Processor Technology about two years ago. I have found PCHL assembles as DF, not E9. The fix for that is easy; change the code in the table. In my listing it is location F978.

I have also found that the pseudo-op DB is counted as three bytes in the first pass; this makes all values in the symbol table be off by 2 x n (where n is the number of DB statements preceding the symbol). The fix for that is more subtle. In locations F78D (DATA1), change JMP OPZ to JMP patch (a four-byte area). At patch, put XRA A (AF), JMP OPZ (C3 6E FA).

Jim Kaufman
2890 15th St.
Boulder, CO 80302

```
369F F0 07              BEQ DISP2       yes
36A1 20 03 36           JSR OTVEC
36A4 E6 1B              INC CAL1
36A6 D0 F0              BNE DISP1       unconditional
36A8 20 03 36   DISP2   JSR OUTVEC
36AB 4C 07 36           JMP SCICAL
36AE A0 00      FUNCTN  LDY#00          function found
36B0 A2 00              LDX#00
36B2 20 E4 36           JSR LOOK        match 1st letter
36B5 20 E4 36           JSR LOOK        match 2nd letter
36B8 20 E4 36           JSR LOOK        match 3rd letter
36BB B9 86 37           LDA TAB2-1,Y    Add Hi byte
36BE 85 1C              STA CAL2
36C0 B9 85 37           LDA TAB2-2,Y    Add Lo byte
36C3 85 1B              STA CAL1
36C5 A0 06              LDY#06
36C7 20 0C 37           JSR LOAD        load arg
36CA B0 07              BCS FUN1
36CC 20 BB 31           JSR RECALL
36CF C9 FF              CMP#FF
36D1 F0 26              BEQ WHAT        number not in mem
36D3 20 FC FC   FUN1    JSR MVYX
36D6 20 E1 36           JSR FUN         perform function
36D9 20 0C FD           JSR MVZX
36DC A2 08              LDX#08          round off to 8 digits
36DE 4C 6E 36           JMP OUT         display result
36E1 6C 1B 00   FUN     JMP(CAL1)
36E4 B9 4B 37   LOOK    LDA TAB1,Y      compare letter to table
36E7 D5 42              CMP LR+2,X
36E9 F0 0B              BEQ FOUND
36EB C9 FF              CMP#FF          end of table
36ED F0 05              BEQ NTFND
36EF C8                 INY             next position
36F0 C8                 INY
36F1 C8                 INY
36F2 D0 F0              BNE LOOK
36F4 F0 03      NTFND   BEQ WHAT        function not there
36F6 E8         FOUND   INX
36F7 C8                 INY
36F8 60                 RTS
36F9 A2 05      WHAT    LDX#05          output "WHAT"
36FB BD 06 37   WHAT1   LDA WHAT2,X
36FE 95 40              STA LR,X
3700 CA                 DEX
3701 10 F8              BPL WHAT1
3703 4C 94 36           JMP DISP
3706 57 48 41           BYTE 57 48 41   "WHAT cr lf"
3709 54 0D 0A           BYTE 54 0D 0A
370C B1 08      LOAD    LDA(ARGYL),Y    load varible into Ry
370E 20 1B 37           JSR LTRTST
3711 90 07              BCC LOAD1
3713 20 89 35           JSR PACADD      adjust address
3716 20 00 30           JSR PACKER      load number
3719 38                 SEC
371A 60         LOAD1   RTS
371B C9 41      LTRTST  CMP#41          test for letter
371D 90 04              BCC BAD
371F C9 5B              CMP#5B
3721 90 01              BCC OUTL
3723 38         BAD     SEC             C=1,non letter
3724 60         OUTL    RTS
3725 C9 5E      OPERT   CMP# 5E         raise to power
3727 D0 03              BNE OP1
3729 4C 2E 35           JMP XRY

372C C9 2A      OP1     CMP#2A          *
372E D0 03              BNE OP2
3730 4C 0B F9           JMP MUL
3733 C9 2F      OP2     CMP#2F          /
3735 D0 03              BNE OP3
3737 4C 16 FA           JMP DIVIDE
373A C9 2B      OP3     CMP#2B          +
373C D0 03              BNE OP4
373E 4C 08 F8           JMP ADD
3741 C9 2D      OP4     CMP#2D          -
3743 D0 03              BNE OP5
3745 4C 00 F8           JMP SUB
3748 4C F0 FC   OP5     JMP MVXZ
```

Tables:

```
374B 41 42 53   ABS             TAB1    function code names
374E 41 43 53   ACS
3751 41 4C 47   ALG
3754 41 53 4E   ASN
3757 41 54 4E   ATN
375A 43 4F 53   COS
375D 44 45 47   DEG
3760 46 4E 41   FNA
3763 46 4E 42   FNB
3766 46 4E 43   FNC
3769 49 4E 56   INV
376C 4C 4F 47   LOG
376F 52 41 44   RAD
3772 53 49 4E   SIN
3775 53 51 52   SQR
3778 54 41 4E   TAN
377B FF FF FF
377E FF FF FF
3781 FF FF FF
3784 FF FF FF
3787 FF 13 35                   TAB2    function addresses
378A FF 3E 34
378D FF F9 32
3790 FF 54 34
3793 FF 6D 34
3796 FF 99 33
3799 FF 1E 35
379C FF F0 FC
379F FF F0 FC
37A2 FF F0 FC
37A5 FF 4A 35
37A8 FF 10 32
37AB FF 26 35
37AE FF 54 33
37B1 FF 72 32
37B4 FF 70 33
37B7 FF FF FF
37BA FF FF FF
37BD FF FF FF
37C0 40 17 45 32 92 51 99 40 F2   Pi/180       00
37C9 40 31 62 27 76 60 16 70 F1   1/SQR(10)    09
37D2 00 15 70 79 63 26 79 50 F0   Pi/2         12
37DB 00 18 F2                     180           1B
37DE 00 90 F1                      90           1E
37E1 00 31 41 59 26 53 59 F0       Pi           21
```

GPM for the M6800

Frits van der Wateren
Physical Laboratory
Wilhemina Hospital
Amsterdam 1056 - E.G
The Netherlands

Received: 77 Nov 28

This GPM version is written after an article in *The Computer Journal*, Vol. 8, page 225, by C. Strachey. I will give a short explanation about GPM here. For further details see the article mentioned above.

GPM is a general purpose macro generator, i.e. every character from the input device is directly echoed on the output device until the start of a macro call. This call is then replaced by its body, with all evaluation of parameters etc. And the final result appears then on the output device.

A macro call is initiated by the $ sign and looks like:

```
$MACRONAME, PARAMETER 1, PARAMETER 2;
```

We see three special characters: $, ; . The action of $ is already mentioned above. The , is used to separate the different items. And the ; closes the macro call. These characters are called "warning characters" in GPM. MACRONAME is now evaluated with the parameters: PARAMETER 1 and PARAMETER 2. Beyond the standard macros which are implemented in the system, you must define your own macros to be used. This is done by the standard macro DEF. We want, for instance, to define a macro SEQUENCE which gives as a result 0123456789. In GPM this looks like:

```
$DEF,SEQUENCE,0123456789;
```

If we now call the macro SEQUENCE by:

```
$SEQUENCE;
```

we get:

```
0123456789
```

as a result on the output device.

There are some other warning characters, namely: < > and #. The characters < and > are used as string quotes and have the effect of preventing evaluation of any macro calls inside.

In place of an evaluation, however, one "layer" of string quotes is removed. The warning character #, which is always followed by one single character, has, during the evaluation, as action that it is replaced by the parameter supplied in the call. The character which follows # may be any number from 0 to 9 and any letter from A to Z. A stands for 10, B for 11, etc. Now #1 stands for parameter 1 and so on, while #0 stands for the macro name itself.

Now a real example of a macro, which adds 2 to the contents of a memory location. We define ADD2 as:

```
$DEF,ADD2,<     LDAB        #1
    ADDB        <#>2
    STAB        #1
>;
```

We see that # in line 2 must be quoted to prevent evaluation, otherwise it would be replaced by parameter 2, which gives nothing as a result in this case.

A call to ADD2 with DATA as a parameter, thus:

```
$ADD2,DATA;
```

Will give as a result:

```
LDAB        DATA
ADDB        #2
STAB        DATA
```

Macro calls may be nested to any level. See, however, for a more detailed description of these techniques the article by Strachey.

Standard (or machine) macros in the M6800 implementation:

$DEF, macroname, macrobody;
Define the macro "macroname" with body: "macrobody". This macro body, in turn, may contain both macro calls and definitions.

$VAL, macroname;
This call has as a result the body of macro "macroname" without any evaluation. Applying this macro to a machine macro or a non existing macro, has nothing as a result.

$UPDATE, macroname, newbody;
This macro has the same sort of effect as DEF except that "macroname" is not newly defined but already exists, and is supplied with a new body, which is given as second argument in UPDATE. The new body may be of equal length or shorter. A longer body will be truncated, but will give no error message.

$CLEAR;
Reinitializes GPM, i.e all user-defined macros are deleted and all current evaluations are aborted. The warning characters are set to their default values.

$DUPLEX,n;
The only effect of DUPLEX is setting the input mode of the TTY to half or full duplex. When n=1 the input characters are echoed; when n=0 the input characters are not echoed. This feature is very useful when using GPM interactively.

$USER,xxxx,string;
Goto user routine at address xxxx (hex) with "string" as parameter. "string", however, may be omitted, thus: $USER,xxxx; when the address is shorter than 4 digits, the MSB digits are supplied with zeros. Non-hex characters are ignored. One must be familiar with the internal structure of GPM, when using this macro call.

$CW,<abcdef>;
Change warning characters in the following sequence: $,;#><So "$" becomes "a" and "," becomes "b", etc. This is of course an absurd example. $CW,<[.] \ / >; would be a more sensible definition. Now all warning characters are lower case characters on most terminals. When using this feature it is best to use this macro immediately after starting GPM.

As we see this GPM version has no error messages at all so everything is allowed. This is the greatest difference between this implementation and that of Strachey; and in my opinion an improvement. When you use in a definition, for instance, #4 as a parameter call. And it seems that in the actual macro call parameter 4 is not supplied, the #4 is replaced by the null string i.e. nothing, rather than an error message. This may be very handy. The only problem we can meet sometimes is a lack of memory space. GPM then types: ** MEMORY FULL ** and the current evaluation is aborted. All macros defined so far are retained.

```
                        M6800 MICRO ASSEMBLER



                *****************   GPM   ******************
                *
                * A GENERAL PURPOSE MACROGENERATOR.
                * AN IMPLEMENTATION FOR THE M6800,
                * AFTER AN ARTICLE IN THE COMPUTER JOURNAL
                * VOL. 8 PAGE 225 BY C.STRACHEY.
                * WRITTEN BY FRITS VAN DER WATEREN.
                * PHYSICAL LABORATORY
                * DEPT. E.N.T. WILHELMINA HOSPITAL
                * AMSTERDAM    THE NETHERLANDS.
                *
     0000               ORG 0
                        OPT 1
                *
                * OPT 1 HAS THE EFFECT OF SETTING BIT-7
                * WHEN EVALUATING ASCII CHAR. IN 'FCC' OR #'X
                * OPT 0 CLEARS THIS BIT.
                * THIS FEATURE IS VERY ESSENTIAL FOR THIS
                * IMPLEMENTATION OF GPM. SINCE ALL CHARACTERS
                * (8-BITS) HAVE THIS BIT SET.
                * ALL POINTERS (16-BITS) HAVE THIS BIT CLEARED.
                * SO GPM CAN ONLY RUN IN THE FIRST 32K!
                *
                *
                ******  BASE PAGE LOCATIONS  ******
                *
                *
0000 00         C0      FCB     0,0,0,0,0,0     6 WARNING CHAR.
0006 00         F       FDB     0       START OF NOT ENTERED MACRO
0008 00         P       FDB     0       START OF ENTERED MACRO
000A 00         H       FDB     0       POINTS TO 'LENGTH CELL'
000C 00         C       FDB     0       POINTS TO SOURCE CHAR.
000E 00         E       FDB     0       POINTS TO LAST MACRO IN E-CHAIN
0010 00         Q       FCB     0       QUOTE COUNTER
0011 00         S       FDB     0       POINTS TO FIRST FREE CELL
0013 00         V       FDB     0       V W K L ARE TEMPORIES.
0015 00         W       FDB     0
0017 00         K       FDB     0
0019 00         L       FDB     0
001B 00         BEG     FDB     0
001D 00         END     FDB     0
001F 00         SAVEX   FDB     0
0021 00         A16     FDB     0
0023 00         B16     FDB     0
0025 00         BOTTOM  FDB     0
0027 00         DUPLX   FCB     0       0=NO ECHO; $FF=ECHO
     0100               ORG     $100
                *
                * INITIAL START OF GPM IS HERE!
                *
0100 8E 00FF    GPM     LDS     #$FF    STACK IS IN ZERO PAGE
0103 FE 010D            LDX     ACIA
0106 86 01              LDA A   #%1     7-BIT+EVEN PARITY
0108 A7 00              STA A   0,X     CLOCK /16
010A 7E 04E8            JMP     CLEAR
                *
                * PATCH POINTS:
                *
010D FC         ACIA    FDB     $FCF4
010F 06         FWAM    FDB     FWS     FIRST WORD AVAILABLE MEMORY
0111 1F         LWAM    FDB     $1FFF   LAST WORD OF AVAILABLE MEMORY
                *
                * NEXTCH  LOADS NEXT CHARACTER IN A
                *
0113 DF 1F      NEXTCH  STX     SAVEX
0115 DE 0C              LDX     C       C AT BOTTOM OF STACK?
0117 27 07              BEQ     TH      YES,READ CHAR.
0119 A6 00              LDA A   0,X     NO,GET CHAR OF STACK
011B 08                 INX
011C DF 0C              STX     C
011E 20 1C              BRA     R1
0120 DE 0A      TH      LDX     H
0122 26 04              BNE     READS
0124 DE 11              LDX     S       WE ARE ON THE LOWEST LEVEL NOW
0126 DF 25              STX     BOTTOM   SAVE GPM STACK POINTER.
0128 FE 010D    READS   LDX     ACIA

012B A6 00      INCH    LDA  A  0,X
012D 47                 ASRA            WAIT FOR TTY
012E 24 FB              BCC     INCH
0130 A6 01              LDA A   1,X     READ CHAR.
0132 48                 ASLA
0133 0D                 SEC
0134 46                 RORA            SET BIT 7
0135 7D 0027            TST     DUPLX   ECHO?
0138 27 02              BEQ     R1      NO
013A 8D 14              BSR     OUTCH
013C DE 1F      R1      LDX     SAVEX
013E 4D                 TST A
013F 39                 RTS
                *
                * LOAD OR OUTPUT A CHAR. FROM A
                *
0140 DF 1F      LOAD    STX     SAVEX
0142 DE 0A              LDX     H       H AT BOTTOM OF STACK?
0144 27 05              BEQ     WRITES  YES,WRITE CHAR.
0146 BD 022A            JSR     PUSH    NO,PUSH ON STACK
0149 20 02              BRA     R2
014B 8D 03      WRITES  BSR     OUTCH
014D DE 1F      R2      LDX     SAVEX
014F 39                 RTS

0150 36         OUTCH   PSHA
0151 FE 010D            LDX     ACIA
0154 A6 00      OUTC1   LDA A   0,X
0156 47                 ASRA
0157 47                 ASRA
0158 24 FA              BCC     OUTC1   WAIT FOT TTY DONE!
015A 32                 PULA
015B A7 01              STA A   1,X     PRINT CHAR
015D 39                 RTS
                * ROUTINE 'FIND' SEARCHES FOR A MACRONAME
                * ( X POINTS TO THIS NAME ) IN THE E-CHAIN
                * ON EXIT X AND W POINT TO
                * THE FIRST WORD OF THE MACROBODY.
                * WHEN MACRO IS NOT IN THE E-CHAIN,
                * FIND RETURNS THE NULL STRING

015E DF 15      FIND    STX     W
0160 DE 0E              LDX     E
0162 DF 17              STX     K
0164 DE 15      SETUP   LDX     W
0166 DF 13              STX     V
0168 EE 00              LDX     0,X     GET LENGTH OF MACRO NAME.
016A DF 21              STX     A16
016C DE 17              LDX     K
016E DF 19              STX     L
0170 DE 13      SEARCH  LDX     V
0172 A6 00              LDAA    0,X
0174 08                 INX
0175 DF 13              STX     V
0177 DE 19              LDX     L
```

```
0179 A1 02            CMP A   2,X      COMPARE CHAR. BY CHAR.
017B 26 18            BNE     NEXT     SEARCH FAIL,SO PREVIOUS MACRO
017D 08               INX                IN E-CHAIN.
017E DF 19            STX     L
0180 DE 21            LDX     A16
0182 09               DEX              ALL CHARACTERS MATCH?
0183 DF 21            STX     A16
0185 26 E9            BNE     SEARCH   NOT YET!
0187 DE 17    FOUND   LDX     K
0189 DF 21            STX     A16
018B EE 02            LDX     2,X
018D BD 01A2          JSR     ADD      SKIP MACRONAME.
0190 08               INX
0191 08               INX              THIS IS THE FIRST WORD
0192 DF 15            STX     W          OF THE MACRO BODY.
0194 39               RTS
0195 DE 17    NEXT    LDX     K
0197 8C 05CE          CPX     #NIL.    E-CHAIN EXAUSTED?
019A 27 EB            BEQ     FOUND    YES,RETURN A NIL-BODY.
019C EE 00            LDX     0,X      NO,NEXT MEMBER OF E-CHAIN
019E DF 17            STX     K
01A0 20 C2            BRA     SETUP
                      *
                      * SUBROUTINE ADD   X=A16=A16+X
                      *                  B16= 'OLD' X
                      *
01A2 DF 23    ADD     STX     B16
01A4 36               PSH A
01A5 96 24            LDA A   B16+1
01A7 9B 22            ADD A   A16+1    ADD LSB
01A9 97 22            STA A   A16+1
01AB 96 23            LDA A   B16
01AD 99 21            ADC A   A16      ADD MSB
01AF 20 0D            BRA     EX
                      *
                      * SUBROUTINE SUB  X=A16=A16-X
                      *
01B1 DF 23    SUB     STX     B16
01B3 36               PSH A
01B4 96 22            LDA A   A16+1
01B6 90 24            SUB A   B16+1    SUBSTRACT LSB
01B8 97 22            STA A   A16+1
01BA 96 21            LDA A   A16
01BC 92 23            SBC A   B16      SUBSTRACT MSB
01BE 97 21    EX      STA A   A16
01C0 32               PUL A
01C1 DE 21            LDX     A16
01C3 39               RTS
                      *
                      *  REVERSE SUBSTRACT  X=B16=X-A16
                      *
01C4 DF 23    RSUB    STX     B16
01C6 36               PSH A
01C7 96 24            LDA A   B16+1
01C9 90 22            SUB A   A16+1    SUBSTRACT LSB
01CB 97 24            STA A   B16+1
01CD 96 23            LDA A   B16
01CF 92 21            SBC A   A16      SUBSTRACT MSB
01D1 97 23            STA A   B16
01D3 32               PUL A
01D4 DE 23            LDX     B16
01D6 39               RTS
                      *
                      * SUBROUTINE ADDB X=X+A
                      *
01D7 DF 23    ADDB    STX     B16
01D9 9B 24            ADDA    B16+1
01DB 97 24            STAA    B16+1
01DD 96 23            LDAA    B16
01DF 89 00            ADCA    #0
01E1 97 23            STAA    B16
01E3 DE 23            LDX     B16
01E5 39               RTS
                      *
                      * COMPUTE CHARACTER EQUIVALENT NUMBER IN A
                      *
                      *  0 - 9 (ASCII) GIVES 0 - 9 (DEC NUMBER)
                      *  A - Z (ASCII) GIVES 10 - 35 (DEC NUMBER)
                      *  ALL OTHER CHARACTERS GIVE A ZERO RESULT.
                      *
01E6 80 B0    NUMBER  SUB A   #@260
01E8 2D 0E            BLT     ZERO     NO LEGAL CHAR.
01EA 81 09            CMP A   #9       0 - 9 ?
01EC 2F 0B            BLE     ZERO+1   YES
01EE 80 07            SUB A   #7       NO,
01F0 81 09            CMP A   #9
01F2 2D 04            BLT     ZERO     A - Z ?
01F4 81 23            CMP A   #35
01F6 2E 01            BGT     ZERO+1   YES
01F8 4F       ZERO    CLR A
01F9 39               RTS
                      *
                      * MEMORY FULL!
                      * PRINT THIS MESSAGE AND
                      * DELETE F-CHAIN AND P-CHAIN, SAVE E-CHAIN.
                      *
01FA CE 0214  FULL    LDX     #MEMFUL
01FD A6 00    FULLL   LDAA    0,X
01FF 81 04            CMPA    #4
0201 27 0A            BEQ     RSTART
0203 08               INX
0204 DF 1F            STX     SAVEX
0206 BD 0150          JSR     OUTCH    PRINT STRING
0209 DE 1F            LDX     SAVEX
020B 20 F0            BRA     FULLL
020D DE 25    RSTART  LDX     BOTTOM
020F DF 11            STX     S        BOTTOM OF GPM STACK
0211 7E 0505          JMP     INIT

0214 0D       MEMFUL  FCB     $D,$A
0216 AA               FCC     /** MEMORY FULL **/
0227 0D               FCB     $D,$A,4
                      *
                      * PUSH A ON TOP OF GPM STACK
                      *
022A DE 11    PUSH    LDX     S
022C BC 0111          CPX     LWAM     MEMORY FULL?
022F 27 C9            BEQ     FULL     YES
0231 A7 00            STA A   0,X      NO,LOAD CHAR. AT TOP OF STACK
0233 08               INX              ADVANCE STACKPOINTER
0234 DF 11            STX     S
0236 39               RTS
                      *
                      * GET A CHARACTER AND SEARCH FOR WARNING CHARACTER
                      *
0237 BD 0113  START   JSR     NEXTCH
023A 2A 42            BPL     ENDFN.   MARKER!
023C 91 00            CMPA    C0       $
023E 27 41            BEQ     FN       CREATE NEW F-MEMBER
0240 91 01            CMPA    C0+1     ,
0242 27 5E            BEQ     NXTITM   NEXT ITEM
0244 91 02            CMPA    C0+2     ;
0246 27 30            BEQ     APPLY.   ADD F-MEMBER TO P-CHAIN
0248 91 03            CMPA    C0+3     #
024A 27 2F            BEQ     LDARG.   LOAD ARGUMENT
024C 91 04            CMPA    C0+4     >
024E 27 27            BEQ     EXIT
0250 91 05            CMPA    C0+5     <
```

```
0252 27 13          BEQ     Q2      Q=Q+1
0254 BD 0140 COPY   JSR     LOAD    NO WARNING CHAR. SO COPY
0257 96 10   SCAN   LDA A   Q       BETWEEN <> ?
0259 27 DC          BEQ     START   NO
025B BD 0113 Q1     JSR     NEXTCH  YES,COPY CHAR.
025E 91 05          CMPA    C0+5    < ?
0260 26 0A          BNE     Q3
0262 7C 0010        INC     Q       YES, Q=Q+1
0265 20 ED          BRA     COPY

0267 7C 0010 Q2     INC     Q
026A 20 EF          BRA     Q1

026C 91 04   Q3     CMPA    C0+4    > ?
026E 26 E4          BNE     COPY    NO,COPY CHAR.
0270 7A 0010        DEC     Q       YES,Q=Q-1
0273 27 C2          BEQ     START   OUT OF <> !
0275 20 DD          BRA     COPY

0277 3F      EXIT   SWI     SINGLE > IS USED TO LEAVE GPM

0278 7E 02C8 APPLY. JMP     APPLY
027B 7E 0366 LDARG. JMP     LDARG
027E 7E 03A5 ENDFN. JMP     ENDFN
             *  WARNINGCHARACTER $
             * CREATE NEW MEMBER AT TOP OF F-CHAIN.

0281 96 0A   FN     LDA A   H       PUSH H ON STACK
0283 8D A5          BSR     PUSH
0285 96 0B          LDA A   H+1
0287 8D A1          BSR     PUSH
0289 96 06          LDA A   F       PUSH F ON STACK
028B 8D 9D          BSR     PUSH
028D 96 07          LDA A   F+1
028F 8D 99          BSR     PUSH
0291 09             DEX
0292 09             DEX
0293 DF 06          STX     F       F NOW POINTS TO NEW MEMBER
0295 4F             CLR A
0296 8D 92          BSR     PUSH    RESERVE WORD FOR C-POINTER
0298 8D 90          BSR     PUSH
029A DF 0A   NEWH   STX     H       H POINTS TO NEW LENGTH CELL
029C 8D 8C          BSR     PUSH    AND CLEAR THIS CELL
029E 8D 8A          BSR     PUSH
02A0 20 95          BRA     START


             *  WARNINGCHARACTER ,
             * CLOSE CURRENT ITEM AND SETUP NEW ONE.

02A2 DE 0A   NXTITM LDX     H       ANY STRING ON STACK?
02A4 27 AE          BEQ     COPY    NO,COPY ,
02A6 DE 11          LDX     S       YES,COMPUTE LENGTH OF CURRENT IT
02A8 DF 21          STX     A16
02AA DE 0A          LDX     H
02AC BD 01B1        JSR     SUB     S-H
02AF DE 0A          LDX     H
02B1 EE 00          LDX     0,X
02B3 BD 01B1        JSR     SUB     S-H-(H)
02B6 DE 0A          LDX     H
02B8 96 21          LDA A   A16
02BA A7 00          STA A   0,X     AND LOAD CURRENT LENGTH CELL
02BC 96 22          LDA A   A16+1
02BE A7 01          STA A   1,X
02C0 DE 11          LDX     S
02C2 4F             CLR A
02C3 20 D5          BRA     NEWH

             *  WARNING CHARACTER ;
             * ADD F MEMBER TO TOP OF P-CHAIN
             * TOP F-MEMBER IS THEN DELETED.

02C5 7E 0254 COPY.  JMP     COPY

02C8 DE 08   APPLY  LDX     P
02CA DF 15          STX     W
02CC CE 0006        LDX     #F
02CF BD 0456        JSR     CMP     P>F ?
02D2 24 F1          BCC     COPY.   YES,UNMATCHED ;
02D4 DE 0A          LDX     H       NO,ANY STRING ON STACK?
02D6 27 ED          BEQ     COPY.   NO,COPY ;
02D8 DE 11          LDX     S
02DA DF 21          STX     A16
02DC DE 0A          LDX     H
02DE BD 01B1        JSR     SUB     S-H
02E1 DE 0A          LDX     H
02E3 96 21          LDA A   A16     ADJUST LENGTH CELL
02E5 A7 00          STA A   0,X     OF CURRENT ITEM
02E7 96 22          LDA A   A16+1
02E9 A7 01          STA A   1,X
02EB 4F             CLR A           YES,APPEND MARKER
02EC BD 022A        JSR     PUSH     (= TWO ZERO BYTES)
02EF BD 022A        JSR     PUSH
02F2 DF 21          STX     A16
02F4 DE 06          LDX     F
02F6 09             DEX
02F7 09             DEX             POINTS TO LENGTH CELL
02F8 DF 17          STX     K
02FA DE 06          LDX     F
02FC EE 00          LDX     0,X
02FE DF 06          STX     F       F NOW POINTS TO PREVIOUS F-MEMBR
0300 DE 17          LDX     K
0302 BD 01B1        JSR     SUB     S-F+2 (F-MEMBER LENGTH)
0305 DE 17          LDX     K
0307 EE 00          LDX     0,X     H NOW POINTS TO
0309 DF 0A          STX     H        THE MOST PREVIOUS LENGTH CELL
030B DE 17          LDX     K
030D 96 21          LDA A   A16     INSERT TRUE LENGTH
030F A7 00          STA A   0,X     OF F-MEMBER
0311 96 22          LDA A   A16+1
0313 A7 01          STA A   1,X
0315 96 08          LDA A   P
0317 A7 02          STA A   2,X     CHANGE F-MEMBER INTO P-MEMBER
0319 96 09          LDA A   P+1
031B A7 03          STA A   3,X
031D 96 0C          LDA A   C
031F A7 04          STA A   4,X
0321 96 0D          LDA A   C+1
0323 A7 05          STA A   5,X
0325 08             INX
0326 08             INX
0327 DF 08          STX     P       P NOW POINTS TO NEW MEMBER
0329 DE 0A          LDX     H       WAS THERE ANY PREVIOUS STRING.
032B 27 17          BEQ     FMACRO  NO
032D DE 17          LDX     K
032F EE 00          LDX     0,X     YES,GET LENGTH OF NEW P-MEMBER
0331 DF 21          STX     A16
0333 DE 0A          LDX     H
0335 EE 00          LDX     0,X     AND LENGTH OF PREVIOUS ITEM
0337 BD 21A2        JSR     ADD     AND CORRECT IT
033A DE 0A          LDX     H
033C 96 21          LDA A   A16
033E A7 00          STA A   0,X
0340 96 22          LDA A   A16+1
0342 A7 01          STA A   1,X
```

```
0344 DE 08      FMACRO  LDX     P
0346 08                 INX
0347 08                 INX
0348 08                 INX
0349 08                 INX     POINTS TO LENGTH CELL OF MACRO TO BE CAD
034A BD 015E            JSR     FIND    SEARCH FOR MACRO IN E-CHAIN
034D EE 00              LDX     0,X     IS IT A MACHINE MACRO?
034F 2A 0C              BPL     MBODY   NO
0351 DF 15              STX     W       YES (BIT 15 IS SET)
0353 78 0015            ASL     W       NOW CLEAR THIS BIT
0356 74 0015            LSR     W
0359 DE 15              LDX     W
035B 6E 00              JMP     X       AND EXECUTE MACINE MACRO
035D DE 15      MBODY   LDX     W
035F 08                 INX
0360 08                 INX
0361 DF 0C              STX     C       POINTS TO FIRST CHARACTER OF
0363 7E 0237            JMP     START   CALLED MACRO BODY
                * WARNING CHARACTER #
                * REPLACES ARGUMENTNUMBER BY ARGUMENT.

                * WHEN THERE IS NO ARGUMENT FOR THIS
                * NUMBER,NOTHING IS REPLACED.
0366 DE 08      LDARG   LDX     P       ANY MACRO CALL ENTERED?
0368 27 38              BEQ     COPY..  NO,COPY #
036A 08                 INX
036B 08                 INX
036C 08                 INX
036D 08                 INX
036E DF 21              STX     A16     POINTS TO LENGTH CELL.
0370 BD 0113            JSR     NEXTCH  GET NEXT SINGLE CHARACTER
0373 BD 01E6            JSR     NUMBER  A=NUMBER REPRESENTATION
0376 EE 00      GETARG  LDX     0,X     GET LENGTH CELL
0378 8C 0000            CPX     #0      IF IT IS A MARKER,
037B 27 22              BEQ     START.  ARGUMENT LIST IS EXAUSTED.
037D 4A                 DEC A           PARAMETER NUMBER REACHED?
037E 2B 07              BMI     REACHD  YES!
0380 BD 01A2            JSR     ADD     NO,COMPUTE NEXT ARGUMENT ADRES.
0383 DF 21              STX     A16
0385 20 EF              BRA     GETARG

0387 DE 21      REACHD  LDX     A16
0389 DF 1B              STX     BEG
038B EE 00              LDX     0,X     GET LENGTH OF THIS ARGUMENT
038D BD 01A2            JSR     ADD     END OF STRING+1 IN A16
0390 DE 1B              LDX     BEG     GET BEGIN OF STRING
0392 08                 INX
0393 08         MOVARG  INX
0394 9C 21              CPX     A16     EMPTY?
0396 27 07              BEQ     START.
0398 A6 00              LDAA    0,X     TRANSFER CHAR. WISE
039A BD 0142            JSR     LOAD
039D 20 F4              BRA     MOVARG

039F 7E 0237    START.  JMP     START
03A2 7E 0254    COPY..  JMP     COPY
                * WARNING CHARACTER 'MARKER'
                * DELETES THE TOP P-MEMBER,
                * WITH ALL LOCAL DEF'S IN IT.

03A5 DE 11      ENDFN   LDX     S
03A7 DF 17              STX     K
03A9 96 0E              LDA A   E       PUSH E ON TOP OF STACK
03AB BD 022A            JSR     PUSH
03AE 96 0F              LDA A   E+1
03B0 BD 022A            JSR     PUSH
03B3 DE 08              LDX     P       PREPAIRE FOR E-CHAIN
03B5 09                 DEX              POINTER CORRECTIONS.
03B6 09                 DEX
03B7 DF 1B              STX     BEG     X POINTS TO BEGIN OF P-MEMBER
03B9 DF 21              STX     A16
03BB EE 00              LDX     0,X
03BD DF 15              STX     W       W=LENGTH OF ARG.
03BF BD 01A2            JSR     ADD
03C2 DF 1D              STX     END     END POINTS TO END+1 OF P-MEMBER
03C4 DE 15              LDX     W
03C6 DF 21              STX     A16
03C8 DE 17              LDX     K
03CA EE 00      NEXTE   LDX     0,X     POINTS TO PREVIOUS E-MEMBER
03CC DF 15              STX     W
03CE CE 001D            LDX     #END
03D1 BD 0456            JSR     CMP     BELOW END OF P-MEMBER.?
03D4 25 13              BCS     LOCAL   YES,PERHAPS A LOCAL DEF.
03D6 BD 01C4            JSR     RSUB    NO,CORRECT E-POINTER
03D9 DE 17              LDX     K       I.E. E=E-W
03DB 96 23              LDA A   B16
03DD A7 00              STA A   0,X
03DF 96 24              LDA A   B16+1
03E1 A7 01              STA A   1,X
03E3 DE 15              LDX     W
03E5 DF 17              STX     K
03E7 20 E1              BRA     NEXTE

03E9 DF 15      LOCAL   STX     W
03EB CE 001B            LDX     #BEG
03EE 8D 66              BSR     CMP     E-MEMBER BEFORE P-MEMBER.?
03F0 25 04              BCS     NOLOC   YES,NO LOCAL DEFS LEFT!
03F2 EE 00              LDX     0,X     NO,GET PREVIOUS E-MEMBER
03F4 20 F3              BRA     LOCAL
03F6 DF 15      NOLOC   STX     W
03F8 DE 17              LDX     K       E-CHAIN OF LOCAL DEFS
03FA 96 15              LDA A   W        IS NOW DISCONNECTED.
03FC A7 00              STA A   0,X
03FE 96 16              LDA A   W+1
0400 A7 01              STA A   1,X
0402 DE 11              LDX     S
0404 09                 DEX
0405 09                 DEX
0406 DF 11              STX     S
0408 EE 00              LDX     0,X     PULL NEW E FROM STACK
040A DF 0E              STX     E
040C DE 0A              LDX     H       DEFS ONLY?
040E 27 1F              BEQ     MOVDWN  YES,GO DELETE ARG.
0410 DF 15              STX     W
0412 CE 0008            LDX     #P
0415 8D 3F              BSR     CMP     H-P
0417 25 07              BCS     HLP     H<P!
0419 BD 01C4            JSR     RSUB    H>=P!
041C DF 0A              STX     H       H=H-PHI
041E 20 0F              BRA     MOVDWN   ( PHI = ARG. LENGTH )
0420 EE 00      HLP     LDX     0,X
0422 BD 01C4            JSR     RSUB    (H)=(H)-PHI
0425 DE 0A              LDX     H
0427 96 23              LDA A   B16
0429 A7 00              STA A   0,X
042B 96 24              LDA A   B16+1
042D A7 01              STA A   1,X
042F DE 08      MOVDWN  LDX     P
0431 EE 02              LDX     2,X
0433 DF 0C              STX     C       C NOW POINTS TO PREVIOUS P-MEMBR
0435 DE 08              LDX     P
0437 EE 00              LDX     0,X
0439 DF 08              STX     P       SO DO P
043B DE 11              LDX     S
043D BD 01C4            JSR     RSUB    S=S-PHI
0440 DF 11              STX     S
0442 DE 1D      MOVN1   LDX     END
```

```
0444 A6 00          LDAA    0,X     NOW MOVE DOWN,
0446 08             INX              AND OVERWRITE TOP P-MEMBER.
0447 DF 1D          STX     END
0449 DE 1B          LDX     BEG
044B 9C 11          CPX     S       MOVE DONE?
044D 27 54          BEQ     STRT
044F A7 00          STA A   0,X
0451 08             INX
0452 DF 1B          STX     BEG
0454 20 EC          BRA     MOVN1

                * CMP COMPARES THE 16-BIT VALUE IN W
                * WITH THE 16-BIT VALUE WHERE X POINTS TO.
                * [(X)-W]  (UNSIGNED)
                *  C=0 W >= (X)
                *  C=1 W <  (X)

0456 D6 16      CMP     LDA B   W+1
0458 E0 01              SUB B   1,X
045A D6 15              LDA B   W
045C E2 00              SBC B   0,X
045E DE 15              LDX     W
0460 39                 RTS
                * MACHINE MACRO 'DEF'
                *
                * $DEF,MACRONAME,ARGUMENT;
                *
                * SET MACRO 'MACRONAME' WITH ITS ARGUMENT
                * ON TOP OF THE E-CHAIN.
                *
0461 DE 08      DEF     LDX     P
0463 09                 DEX
0464 09                 DEX
0465 DF 1B              STX     BEG     POINTS TO LENGTH CELL
0467 EE 00              LDX     0,X
0469 DF 21              STX     A16
046B DE 0A              LDX     H       ANY STRINGS LEFT?
046D 27 0F              BEQ     APPEND  NO,THATS EASY
046F EE 00              LDX     0,X     YES,CORRECT LENGTH CELL
0471 BD 01C4            JSR     RSUB
0474 DE 0A              LDX     H
0476 96 23              LDA A   B16
0478 A7 00              STA A   0,X     (H)=(H)-PHI
047A 96 24              LDA A   B16+1
047C A7 01              STA A   1,X
047E DE 08      APPEND  LDX     P
0480 EE 00              LDX     0,X
0482 DF 08              STX     P       REMOVE TOP MEMBER
0484 DE 1B              LDX     BEG      OF P-CHAIN.
0486 96 0E              LDA A   E
0488 A7 00              STA A   0,X
048A 96 0F              LDA A   E+1     AND APPEND IT TO E-CHAIN.
048C A7 01              STA A   1,X
048E DF 0E              STX     E
0490 DE 1B              LDX     BEG     REMOVE MACRONAME
0492 A6 0B      DELDEF  LDA A   11,X     'DEF' FROM LIST!
0494 A7 02              STA A   2,X
0496 08                 INX
0497 9C 11              CPX     S       DONE?
0499 26 F7              BNE     DELDEF  NO!
049B C6 09              LDA B   #9      YES,STACKPOINTER 9 BACK
049D 09         SMIN    DEX
049E 5A                 DEC B
049F 26 FC              BNE     SMIN
04A1 DF 11              STX     S
04A3 7E 0237    STRT    JMP     START
```

```
                * MACHINEMACRO 'CW' CHANGE WARNINGCHARACTERS
                *
                * $CW,<ABCDEF>;
                *
                * ABCDEF STANDS FOR THE DEFAULT
                * WARNING CHARACTERS: $,;#>< IN THIS ORDER.
                *
04A6 DE 08      CW      LDX     P
04A8 DF 15              STX     W
04AA E6 09              LDA B   9,X     GET LENGTH (LSB) OF BODY
04AC C0 02              SUBB    #2
04AE D7 13              STAB    BEG     NUMBER OF CHAR. IN BODY
04B0 5F                 CLRB
04B1 CE 0020            LDX     #C0
04B4 DF 17      CW1     STX     K
04B6 DE 15              LDX     W
04B8 08                 INX
04B9 C1 06              CMPB    #6
04BB 27 10              BEQ     CW2     MORE THAN 6 WARNINGCHAR.!
04BD D1 1B              CMPB    BEG
04BF 27 0C              BEQ     CW2     MACRO BODY EXAUSTED!
04C1 5C                 INCB
04C2 A6 09              LDAA    9,X
04C4 DF 15              STX     W
04C6 DE 17              LDX     K
04C8 A7 02              STAA    0,X     CHANGE WARNINGCHARACTER.
04CA 08                 INX
04CB 20 E7              BRA     CW1
04CD 7E 03A5    CW2     JMP     ENDFN
                *
                * MACHINE MACRO 'VAL'
                *
                * $VAL,XXX;
                *
                * RETURNS THE BODY OF MACRO XXX
                * WITHOUT EVALUATING IT
                *
04D0 DE 08      VAL     LDX     P
04D2 86 09              LDAA    #9
04D4 BD 01D7            JSR     ADDB
04D7 BD 015E            JSR     FIND    FIND THIS MACRO
04DA A6 00              LDAA    0,X
04DC 2B EF              BMI     CW2     IGNORE MACHINE MACROS
04DE 08         VAL1    INX             POINTS TO FIRST CHAR.
04DF A6 01              LDAA    1,X      OF BODY.
04E1 27 EA              BEQ     CW2     AND LOAD ALL CHAR. OF BODY
04E3 BD 0140            JSR     LOAD     UNTIL A MARKER.
04E6 20 F6              BRA     VAL1
                *
                * MACHINE MACRO 'CLEAR'
                *
                * $CLEAR;
                *
                * CLEAR FLUSHES THE GPM STACK
                * AND SETS THE WARNING CHARACTERS
                * TO THEIR DEFAULT VALUE
                *
04E8 86 FF      CLEAR   LDAA    #$FF
04EA 97 27              STAA    DUPLX
04EC CE 0614            LDX     #DUPL.
04EF DF 0E              STX     E
04F1 FE 010F            LDX     FWAM
04F4 DF 11              STX     S
04F6 FE 0516            LDX     SWC     SET WARNING CHARACTERS.
04F9 DF 00              STX     C0
04FB FE 0518            LDX     SWC+2
04FE DF 02              STX     C0+2
0500 FE 051A            LDX     SWC+4
0503 DF 04              STX     C0+4
```

```
0505 CE 2000    INIT    LDX     #0
0508 DF 06              STX     F       SET GPM POINTERS TO
050A DF 08              STX     P       THERE INITIAL VALUES
050C DF 0C              STX     C
050E DF 0A              STX     H
0510 4F                 CLRA
0511 97 10              STAA    Q
0513 7E 0237            JMP     START
0516 A4         SWC     FCC     /$,;#><
                *
                * MACHINE MACRO 'DUPLEX'
                *
                * $DUPLEX,0;
                * CHARACTERS ARE NOT ECHOED!
                *
                * $DUPLEX,1;
                * CHARACTERS ARE ECHOED!
                *
051C DE 08      DUPLEX  LDX     P
051E A6 0D              LDA A   13,X    GET LENHTH OF BODY
0520 81 02              CMPA    #2      EMPTY BODY?
0522 2F 0E              BLE     NOP     YES,NO ACTION.
0524 A6 0E              LDA A   14,X    NO,GET FIRST CHAR.
0526 81 B0              CMPA    #$B0    0=SIMPLEX
0528 27 0B              BEQ     SIMPLX
052A 81 B1              CMP A   #$B1    1=DUPLEX
052C 26 04              BNE     NOP     ALL OTHERS=NO ACTION.
052E 86 FF              LDA A   #$FF
0530 97 27              STA A   DUPLX
0532 7E 03A5    NOP     JMP     ENDFN
0535 7F 0027    SIMPLX  CLR     DUPLX
0538 20 F8              BRA     NOP
                *
                * MACHINE MACRO 'UPDATE'
                *
                * $UPDATE,MACRONAME,ARGUMENT;
                *
                * UPDATE MACRO 'MACRONAME' IN THE E-CHAIN.
                * WITH A NEW 'ARGUMENT'.
                * WHEN THE NEW ARGUMENT DOES NOT FIT,
                * IT IS TRUNCTATED.
                * WHEN THE MACRO IS NOT IN THE E-CHAIN,
                * 'UPDATE' HAS NO RESULT.
                *
053A DE 08      UPDATE  LDX     P
053C 86 2C              LDA A   #12     SKIP OVER MACRONAME 'UPDATE'
053E BD 01D7            JSR     ADDB
0541 BD 015E            JSR     FIND    FIND MACRO TO BE UPDATED.
0544 EE 00              LDX     0,X     MAX LENGTH OF MACRO BODY TO BE
0546 2B 30              BMI     UPDONE  IGNORE MACHINE MACROS.
0548 DF 13              STX     V
054A DE 08              LDX     P
054C EE 0C              LDX     12,X
054E DF 21              STX     A16
0550 DE 08              LDX     P
0552 BD 01A2            JSR     ADD
0555 DE 13              LDX     V
0557 09                 DEX
0558 09         UPDMOV  DEX             DOES IT STILL FIT?
0559 27 16              BEQ     UPDEND  NO,TRUNCTATE
055B DF 13              STX     V
055D DE 21              LDX     A16     YES,CHANGE ARGUMENT!
055F A6 0E              LDA A   14,X    MARKER?
0561 2A 0E              BPL     UPDEND  YES,TRANSFER DONE!
0563 08                 INX
0564 DF 21              STX     A16
0566 DE 15              LDX     W
0568 A7 02              STA A   2,X
056A 08                 INX
056B DF 15              STX     W
056D DE 13              LDX     V
056F 20 E7              BRA     UPDMOV

0571 4F         UPDEND  CLR A
0572 DE 15              LDX     W
0574 A7 02              STA A   2,X     APPEND MARKER
0576 A7 03              STA A   3,X
0578 7E 03A5    UPDONE  JMP     ENDFN
                * MACHINE MACRO 'USER'
                *
                * $USER,XXXX,ANY ARGUMENT;
                *
                * CALL USER ROUTINE AT ADDRESS XXXX (HEX)
                * WITH ANY ARGUMENT STRUCTURE TO BE USED
                * IN THE USER ROUTINE.
                * ON ENTRY 'W' POINTS TO THE FIRST WORD
                * OF THIS ARGUMENT ON THE GPM STACK
                * THE USER MUST RETURN WITH 'JMP ENDFN'
                *
057B DE 08      USER    LDX     P
057D DF 15              STX     W
057F EE 0A              LDX     10,X    GET LENGTH OF ADDRESS ARGUMENT
0581 09                 DEX
0582 DF 13              STX     V
0584 7F 001B            CLR     BEG
0587 7F 001C            CLR     BEG+1
058A DE 13      NXTHEX  LDX     V
058C 09                 DEX             ALL DIGITS DONE?
058D 27 32              BEQ     GO.MAC  YES,GO
058F DF 13              STX     V
0591 DE 15              LDX     W
0593 A6 0C              LDA A   12,X    NO,GET CHAR.
0595 08                 INX
0596 DF 15              STX     W       AND TRANSFORM TO HEX.
0598 80 B0              SUB A   #$B0    WHEN NOT HEX,SKIP THIS CHAR.
059A 2D EE              BLT     NXTHEX
059C 81 09              CMP A   #9
059E 2F 0A              BLE     INHEX
05A0 80 07              SUB A   #7
05A2 81 0A              CMP A   #10
05A4 2D E4              BLT     NXTHEX
05A6 81 10              CMP A   #16
05A8 2C E0              BGE     NXTHEX
05AA D6 1C      INHEX   LDA B   BEG+1
05AC 58                 ASL B           SHIFT DIGIT INTO 16-BIT ADDRESS
05AD 79 001B            ROL     BEG
05B0 58                 ASL B
05B1 79 001B            ROL     BEG
05B4 58                 ASL B
05B5 79 001B            ROL     BEG
05B8 58                 ASL B
05B9 79 001B            ROL     BEG
05BC 1B                 ABA
05BD 97 1C              STA A   BEG+1
05BF 20 C9              BRA     NXTHEX
05C1 DE 15      GO.MAC  LDX     W
05C3 86 0C              LDA A   #12     CALCULATE USER ARGUMENT ADRRESS
05C5 BD 01D7            JSR     ADDB
05C8 DF 15              STX     W
05CA DE 1B              LDX     BEG
05CC 6E 00              JMP     0,X     HERE WE GO!?
                *
                * THE ENVIRONMENT CHAIN (E-CHAIN)
                *
05CE 00         NIL.    FDB     0,5
05D2 CE                 FCC     /NIL/
05D5 00                 FDB     2,0     NULL STRING
```

```
A16         0021 ACIA        010D ADD         01A2
ADDB        21D7 APPEND      047E APPLY       02C8
APPLY.      0278 B16         0023 BEG         021B
BOTTOM      0025 C           000C C0          0000
CLEAR       04E8 CLEAR.      0609 CMP         0456
COPY        0254 COPY.       02C5 COPY..      03A2
CW          04A6 CW.         05E2 CW1         04B4
CW2         04CD DEF         0461 DEF.        05D9
DELDEF      0492 DUPL.       0614 DUPLEX      051C
DUPLX       0027 E           000E END         001D
ENDFN       03A5 ENDFN.      027E EX          01BE
EXIT        0277 F           0006 FIND        015E
FMACRO      0344 FN          0281 FOUND       0187
FULL        01FA FULLL       01FD FWAM        010F
FWS         0620 GETARG      0376 GO.MAC      05C1
GPM         0100 H           000A HLP         0420
INCH        012B INHEX       05AA INIT        0505
K           0017 L           0219 LDARG       0366
LDARG.      027B LOAD        0140 LOCAL       03E9
LWAM        2111 MBODY       035D MEMFUL      0214
MOVARG      0393 MOVDWN      042F MOVN1       0442
NEWH        029A NEXT        0195 NEXTCH      0113
NEXTE       03CA NIL.        05CE NOLOC       03F6
NOP         0532 NUMBER      01E6 NXTHEX      058A
NXTITM      02A2 OUTC1       0154 OUTCH       0150
P           0008 PUSH        022A Q           0010
Q1          025B Q2          0267 Q3          026C
R1          013C R2          014D REACHD      0387
READS       0128 RSTART      020D RSUB        01C4
S           0011 SAVEX       001F SCAN        0257
SEARCH      0170 SETUP       0164 SIMPLX      2535
SMIN        049D START       0237 START.      039F
STRT        04A3 SUB         01B1 SWC         0516
TH          0122 UPD.        05FD UPDATE      053A
UPDEND      0571 UPDMOV      0558 UPDONE      0578
USER        057B USER.       05EA V           0013
VAL         04D0 VAL.        05F4 VAL1        04DE
W           0015 WRITES      014B ZERO        01F8
```

```
MICRO V2E
>
                                                
```

SWTPC EDITOR FORMAT UPDATE

Dear People, *Received 77 Oct 21*

Attached is a program which converts a tape in the format required by the original SWTPC Editor (MP-E) to a format usable by the new co-resident Editor-Assembler (MP-EA). The program assumes that the input tape has no line numbers and creates an output tape with line numbers starting at 1 and incrementing by 1.

For anyone not wishing to type in the program it is available in source form on an AC-30 compatible (Kansas City) co-resident Editor-Assembler format cassette for $5.00 postpaid from Specialized Systems Consultants, PO Box 2847, Olympia, WA 98507.

Also, is anyone interested in MUMPS on a microprocessor?

Phil Hughes

PO Box 2847
Olympia, WA 98507

```
00010                          NAM    SRF
00020                  * S.S.C. POB 2847  Olympia, WA 98507  10-1977
00030                  * REFORMAT SWTPC TAPES FROM EDITOR TO CORES
00040                          OPT    NOG
00050                          OPT    O
00060                          OPT    P
00070                          OPT    NOS
00080                  *
00090       E1AC       INEEE  EQU    $E1AC
00100       E1D1       OUTEEE EQU    $E1D1
00110       E0C8       OUT4HS EQU    $E0C8
00120       8004       CPIA   EQU    $8004
00130                  *
00140 0100                     ORG    $100
00150 0100 7E 0300             JMP    START
00160 0103 7E 0300             JMP    START
00170                  *
00180 0200                     ORG    $200
00190 0200 0002        CDPTR   RMB    2
00200 0202 0002        SAVX    RMB    2
00210 0204 0002        CSAVX   RMB    2
00220 0206 0080        CASBUF  RMB    128       CASSETTE BUFFER
00230 0286 0002        LINE    RMB    2         LINE NUMBER
00240 0300                     ORG    $300
00250       0300       START  EQU    *
00260 0300 BD 03DD             JSR    CINIT
00270 0303 7F 0286             CLR    LINE
00280 0306 7F 0287             CLR    LINE+1
00290 0309 CE 03E2             LDX    #BUF
00300 030C 86 02               LDA A  #2
00310 030E A7 00       LOOP    STA A  0,X
00320 0310 08                  INX
00330 0311 86 20               LDA A  #'
00340       0313       LUP2   EQU    *
00350 0313 A7 00               STA A  0,X
00360 0315 08                  INX
00370 0316 BD 03BA             JSR    CGET
00380 0319 81 1A               CMP A  #$1A      EOF?
00390 031B 27 08               BEQ    EOFR
00400 031D 81 0D               CMP A  #$0D      CR?
00410 031F 26 F2               BNE    LUP2
00420 0321 86 04               LDA A  #4
00430 0323 20 E9               BRA    LOOP
00440 0325 86 03       EOFR    LDA A  #3
00450 0327 A7 00               STA A  0,X
00460 0329 CE 03E2             LDX    #BUF
00470 032C 86 12               LDA A  #$12
00480 032E BD E1D1             JSR    OUTEEE
00490 0331 C6 80               LDA B  #$80
00500 0333 4F          DELAY   CLR A
00510 0334 BD E1D1             JSR    OUTEEE
00520 0337 5A                  DEC B
00530 0338 2A F9               BPL    DELAY
00540 033A 08                  INX
00550 033B 86 02               LDA A  #02       STX
00560 033D BD E1D1             JSR    OUTEEE
00570 0340 20 10               BRA    LNUM      PUT OUT LINE NUMBER
00580 0342 A6 00       OUTLUP  LDA A  0,X
00590 0344 08                  INX
00600 0345 81 03               CMP A  #3
00610 0347 27 2A               BEQ    DIE
00620 0349 36                  PSH A
00630 034A BD E1D1             JSR    OUTEEE
00640 034D 32                  PUL A
00650 034E 81 04               CMP A  #04
00660 0350 26 F0               BNE    OUTLUP
00670       0352       LNUM   EQU    *         INCREMENT AND OUTPUT LINE NUM
00680 0352 B6 0287             LDA A  LINE+1
00690 0355 8B 01               ADD A  #1
00700 0357 19                  DAA
00710 0358 B7 0287             STA A  LINE+1
00720 035B B6 0286             LDA A  LINE
00730 035E 89 00               ADC A  #0
00740 0360 19                  DAA
00750 0361 B7 0286             STA A  LINE
00760 0364 FF 0202             STX    SAVX
00770 0367 CE 0286             LDX    #LINE
00780 036A BD E0C8             JSR    OUT4HS
00790 036D FE 0202             LDX    SAVX
00800 0370 08                  INX
00810 0371 20 CF               BRA    OUTLUP
00820       0373       DIE    EQU    *
00830 0373 86 04               LDA A  #04
00840 0375 BD E1D1             JSR    OUTEEE    OUTPUT EOT
00850 0378 86 03               LDA A  #03
00860 037A BD E1D1             JSR    OUTEEE    OUTPUT ETX
00870 037D 86 14               LDA A  #$14
00880 037F BD E1D1             JSR    OUTEEE
00890 0382 3F                  SWI
00900                  * CASSSETTE INPUT ROUTINES
00910                  *
00920                  * RESET CASSETTE DATA POINTER
00930       0383       CREAD  EQU    *
00940 0383 86 11               LDA A  #$11
00950 0385 BD E1D1             JSR    OUTEEE
00960 0388 86 3C               LDA A  #$3C
00970 038A B7 8007             STA A  CPIA+3
00980                  * LOOK FOR S9
00990 038D BD E1AC RDADR       JSR    INEEE
01000 0390 81 53               CMP A  #'S
01010 0392 26 F9               BNE    RDADR
01020 0394 BD E1AC             JSR    INEEE
01030 0397 81 39               CMP A  #'9
01040 0399 26 F2               BNE    RDADR
01050                  * GET DATA
01060 039B 37                  PSH B
01070 039C 5F                  CLR B
01080 039D BD E1AC RDCHR       JSR    INEEE
01090 03A0 A7 00               STA A  0,X
01100 03A2 08                  INX
01110 03A3 5C                  INC B
01120 03A4 C1 80               CMP B  #$80
01130 03A6 26 F5               BNE    RDCHR     GET NEXT CHAR
01140                  * IGNORE CHECKSUM
01150 03A8 BD E1AC             JSR    INEEE
01160 03AB BD E1AC             JSR    INEEE
01170                  * STOP READER
01180 03AE 86 13               LDA A  #$13
01190 03B0 BD E1D1             JSR    OUTEEE
01200 03B3 86 34               LDA A  #$34
01210 03B5 B7 8007             STA A  CPIA+3
01220 03B8 33                  PUL B
01230 03B9 39                  RTS
01240                  *
01250                  * GET CHARACTER FROM CASSETTE
01260       03BA       CGET   EQU    *
01270 03BA FF 0204             STX    CSAVX
01280 03BD FE 0200             LDX    CDPTR
01290 03C0 A6 00               LDA A  0,X
01300 03C2 08                  INX
01310 03C3 8C 0286             CPX    #CASBUF+128
01320 03C6 27 07               BEQ    CBMPTY
01330 03C8 FF 0200             STX    CDPTR
01340 03CB FE 0204             LDX    CSAVX
01350 03CE 39                  RTS
01360 03CF CE 0206 CBMPTY      LDX    #CASBUF
01370 03D2 36                  PSH A
01380 03D3 FF 0200             STX    CDPTR
01390 03D6 8D AB               BSR    CREAD     FILL BUFFER
01400 03D8 32                  PUL A
01410 03D9 FE 0204             LDX    CSAVX
20 03DC 39                 RTS
01430                  *
01440                  * INITIALIZ CASSETTE DATA POINTER
01450       03DD       CINIT  EQU    *
01460 03DD FF 0204             STX    CSAVX
01470 03E0 20 ED               BRA    CBMPTY
01480       03E2       BUF    EQU    *
01490                  *
01500 A048                     ORG    $A048
01510 A048 0100                FDB    $0100
01520                          END

TOTAL ERRORS 00000
```

COLLEGE NEEDS INFO ON DESKTOP SYSTEMS

Received: 77 Dec 15

I am currently surveying the market for desk top type systems that could be used in a Business School programme, for teaching computer languages, running interactive simulation and possibly C.A.L. and/or C.A.I. Any direction on the above could be most gratefully received.

B.J. Trohon
Master, Business Division

Algonquin College
315 Pembroke St. East
Pembroke, Ontario K8A 3K2
Canada

KAPIAR

AN 8080 MACROPROCESSOR

by Steve Newberry
24225 Summerhill Ave.
Los Altos, CA 94022

KAPIAR is a general purpose macro processor for the 8080. With the exception only of a few added subroutines (ISIS I/O routines, display, signon, recovery, etc.) it is a literal translation into PLM of the RATFOR program MACRO which appears in Chapter 8 of "Software Tools" by Kernighan and Plauger. K. and P. credit Dennis Ritchie for the design, hence the name KAPIAR. The programming was carried out on the Intel MDS computer running under ISIS II at the Stanford University Microcomputer Laboratory under the supervision of Dennis Allison.

Instructions for Use

KAPIAR has six predefined macros: DEFINE, which defines other macros; IFELSE, which enables such definitions to be conditioned upon input; INCR and DECR, which increment/decrement the (numerical) argument by one; SUBSTR, which returns a substring of a given string; and DELETE, which cancels a previous definition (and reclaims the space occupied by that definition).

In addition to these macros, there are also the special controls: Control-A, which APPENDS the result of a macro to the immediately preceding string; Control- C, which CEASES the operation of KAPIAR and returns to ISIS; Control-D, which DISPLAYS the contents of the macro table; and the ampersand '&' which, when immediately followed by carriage return, is the line continuation character.

Macro definitions/calls may be nested to depths of not more than 255 and may be recursive. Up to 256 short (ie., name+defn =< 14 characters) may be entered at the macro table at one time.

The format for macro DEFINITION is:

DEFINE(<name>=[<definition>])

NB: No spaces!!!

Sometimes the square brackets may be omitted, or the equals-sign replaced by a comma, but the above format *always* works (subject only to requirements for additional pairs of square brackets as detailed below).

Up to nine (9) parameters may be specified by placing the dummy parameters $1, . . . $9 in the definition string. $0 may also be used and will always be replaced by the macro name.

The format for macro calls is either just the macro name itself (when no parameters are passed) or:

<name> (<parmlist>)

where <parmlist> is a (possibly empty!) sequence of up to nine real parameters separated by commas. The parameters will be passed in the order of their appearance, eg., the first will replace all occurrences of $1, the second of $2, etc.

Although KAPIAR will routinely pass through unchanged any textual material not (containing strings) defined to be macros, it is good practice to enclose all lines of non-macro-active text in square brackets.

Example:

Suppose you have already defined a macro a=[whammo!]. Then whenever KAPIAR sees an occurrence of 'a' surrounded by non-alphanumeric characters (other than '[' and ']') it will substitute 'whammo!' for that occurrence–even if you had intended otherwise. '[Hello, isn't it a nice day?]' will pass through unchanged. If you omit the brackets the output will be: Hello, isn't it whammo! nice day? Indeed, the question of when and where to use square bracket pairs, and how many pairs to use, constitutes the principal difficulty in programming with KAPIAR.

Example:

If you wish to re-define a macro then the occurrence of the name (in the new definition) must be enclosed in brackets. Thus, the sequence

'DEFINE(A=[B]) DEFINE(A=[C]) A'

will define A to be B, will define B (*not* A!) to be C, and will return C for the call A by first replacing A by B and then replacing B by C.

Example:

If you wish to delete a macro previously defined, then the occurrence of the name (in the DELETE command) must be enclosed in brackets. (Otherwise the DELETE command will be applied to the macro expansion rather than to the macro name.) Thus, the sequence

'DEFINE(A=[B]) DEFINE(B=[C]) DELETE(A)'

will *not* delete A=[B], but will delete B=[C] instead!

Conditionals, Substrings and Recursion

The format for the conditional is:

IFELSE(str1,str2,str3,str4)

If str1 = str2 then str3 will be pushed back on the buffer-stack for rescan. Otherwise str4 will be pushed back.

Example:

The sequence 'DEFINE(A=[IFELSE($1,$2, same, different)]) A(grommet, grommet) A(grommet, blivet)' will return 'same different'. The use of the conditional together with INCR and DECR enables (among other things) recursive definitions by McCarthy's device:

f(n) = IF n=0 THEN f(0) ELSE g(f(n-1), h(n))

which carries over to IFELSE as

DEFINE(f=[IFELSE($1,0,a,[g(f(n-1), h(n))])])

where a = f(0).

Example:

The classic recursive definition of sum in terms of successor 's' and predecessor 'p' is:

```
sum(m,n) = IF m=0 THEN n ELSE sum(p(m), s(n))
```

which carries over as

```
DEFINE(sum=[IFELSE($1,0,$2,[sum(DECR($1),
  INCR($2))])])
```

One may, in similar fashion, define product and exponent, but the execution is hilariously slow, eg., exponent (4,2) takes fifteen seconds to come up with 16.

The SUBSTR function takes three arguments: string, i, j and returns the substring of string which begins with the i-th character and contains j characters. If j is not specified then the substring extends to the end of the string.

Example:

SUBSTR (abcdefg,2,3) returns 'bcd'. SUBSTR (abcdefg,2) returns 'bcdefg'.

Example:

We define a function LENGTH, of one string argument, which returns the number of characters in the string. If the string is null (contains no characters) then return 0, else return 1 plus the length of SUBSTR ($1,2).

```
DEFINE (LENGTH=[IFELSE($1,,0,[INCR(LENGTH
  (SUBSTR($1,2)))])])
```

NB: The function is defined for the null case.

Example:

We define a function REVERSE, of one string argument, which returns the characters of the string in reverse order. If the string is of length one then return the string, else return the first character appended to the string minus the first character. [In what follows, the "at-sign" @ will be used to indicate the placement of the Control-A which is the APPEND operator.]

```
DEFINE(REVERSE=[IFELSE(LENGTH($1),1,$1,
  [REVERSE(SUBSTR($1,2))]]& [[@SUBSTR
  ($1,1,1)])])
```

This exemplifies the usage of the continuation character '&' which will be explained below. NB: REVERSE is undefined for the null case.

Continuation Character

The continuation character '&' may be used at any point on a line and as many times as desired. The *only* requirement is that all "open" (ie., unbalanced) left-brackets be "closed" (i.e., balanced) prior to the occurrence of the '&', that it be *immediately* followed by <crlf>, and that the unbalanced left-brackets be re-opened at the *beginning* of the next line. The reason for this is that KAPIAR will not SEE the ampersand-carriage-return-linefeed if it occurs within square brackets. And, presumably, you want the square brackets to enclose at least some part of the next following line. Further examples follow.

Example:

We define a function DEBLANK, of one string argument, which returns the non-blank characters of the string in the order of appearance. DEBLANK will be used to demonstrate some of the oddities of the APPEND command. If the first character is a blank then return DEBLANK of the rest of the string, else return the first character followed by Control-A and then DEBLANK the rest of the string. [This won't work, but it will be instructive to see why it fails.]

```
DEFINE(DEBLANK=[IFELSE(SUBSTR($1,1,1), ,
  [DEBLANK(SUBSTR($1,2))],]&[[SUBSTR($1,1,1)
  @DEBLANK(SUBSTR($1,2))])])
```

This definition will return the de-blanked version of the argument string and then go into an infinite loop. Suppose the string had been 'a b': Since 'a' is not a blank the recursion step is invoked, and we get 'a@b@DEBLANK(b)'. Since ' ' is a blank, the "true-part" is invoked, and we get 'a@DEBLANK(b)'. Since 'b' is not a blank, the recursion step in invoked, and we get 'a@b@DEBLANK( )'. Since the null string '' is not a blank, the recursion step is invoked and we get 'a@b@@DEBLANK( ), etc., and that is the infinite loop. When this happens to *YOU*, and you may be quite sure that it will, don't be in a hurry to hit the interrupt switch, because this is a typical infinite loop which will blow up the evaluation stack (or the input bufferstack) and KAPIAR can recover from that eventuality. Go take a short walk, or otherwise dispose of about a minute or so and KAPIAR will give you an error message and a prompt.

OK, so we know how to fix this problem. We DELETE the faulty definition and write:

```
DEFINE(DEBLANK=[IFELSE($1,,,[IFELSE(SUBSTR
  ($1,1,1), ,[DEBLANK(SUBSTR)]]&[[[($1,2))],
  [SUBSTR($1,1,1)@DEBLANK(SUBSTR($1,2))])])])
```

Now if we apply this version of DEBLANK to the test string 'a b' we will get 'a@b@', and since Control-A is a non-printing character, it will look to *us* like 'ab'. And the output of REVERSE, for the same input, will look to us like 'b a' even though it really is 'b@ @a', because KAPIAR can *see* Control-A's. So if you wish to define a function:

```
DEFINE(PALINDROME=[IFELSE(DEBLANK($1),
  DEBLANK(REVERSE($1)),yes,no)])
```

you will have to further modify DEBLANK to eliminate occurrences of Control-A as well as blanks. Even *then* if you write REVERSE(DEBLANK($1)) instead of DEBLANK (REVERSE($1)) it won't work. (Try it!!) Anyhow, it takes PALINDROME three whole minutes to verify that "A MAN A PLAN A CANAL PANAMA" is a palindrome, so that should tell you something about recursive definitions.

To recapitulate:

- DEFINE wants the second parameter in brackets, and if a redefinition, then *both* parameters in brackets.
- DELETE wants the macro-name in brackets.
- IFELSE wants the fourth parameter in brackets if it involves any macro-active text (definitions or calls).
- &<crlf> wants balanced square brackets.
- Control-A can be a trap. When working out a definition involving it, use @ instead and then go over to Control-A.
- Recursion can be a trap. Watch out for null-case versus unit-case terminations.

Programming with KAPIAR

The usual application of KAPIAR is the manipulation/transformation of text: You (1) build up a collection of defined macros (which constitute the *program*), (2) create a file in which the program is followed by the text (the *data*),

and (3) submit this file (the *runfile*) to KAPIAR for processing in batch mode.

When KAPIAR signs on, it tells you how to select batch mode (enter the name of the runfile and hit <cr>), or interactive mode (enter *two* <cr>'s). In either mode, KAPIAR copies its output to the file :F1:KAPIAR.SAV.

Normal procedure is as follows: (1) type KAPIAR <cr> and select interactive mode. It is a good idea to have your macros already written out. (2) Test each macro until you are satisfied that it does what you intend and then type a left hand square bracket, some comment to the effect that the following definition is okay, then the DEFINE statement as tested, then a right hand square bracket <cr>. KAPIAR will output this to the save file. Then go on to test the next macro, etc. (3) When all the macros have been tested and saved, then exit KAPIAR (Control-C<cr>) and EDIT the save file. KAPIAR.SAV will contain, in addition to the commented DEFINE statements, all the junk output from your testing. (4) Copy the edited file containing the tested DEFINE statements to :F1:<filename>.PGM, and write-protect it. (5) Somehow, get your data into :F1:<filename>.TXT. (6) Now COPY :F1:<filename>.PGM, :F1:<filename>.TXT to :F1:<filename>.RUN. (7) Now reload KAPIAR and select batch mode by responding to the prompt with :F1:<filename>.RUN. Your output will appear in :F1:KAPIAR.SAV.

Closing Remarks

In the development/testing phase you will often find it useful to display the macro-table (Control-D) to check that what you think you typed in is what KAPIAR really got. The tilde ' ~ ' is an end-of-string marker. The ascii EM, SUB, FS, GS, RS, US and tilde have special significance to KAPIAR so if you must use any of them in text then be sure to enclose them in square brackets. Of these, only tilde is a printing character. All other ascii printing characters may be freely used.

If you get hung in a loop, wait for a minute or so before interrupting out. KAPIAR will be able to recover if any of the stacks blow up, and most loops will blow up at least one stack. Remember that if your definition doesn't work (or loops) then you have probably omitted some square bracket pairs or parenthesis pairs of a null case terminator or have failed to take into account the presence of the APPEND operator as a string element. KAPIAR always acknowledges a new definition by giving an extra <crlf> (blank line) before the prompt. If you don't get that blank line then KAPIAR is waiting for more right hand parens or brackets.

Finally, it should be remarked that KAPIAR is most appropriately employed with simple non-recursive string replacement macros. Quoting Kernighan and Plauger, "Beware of becoming too clever with macros. In principle, [KAPIAR] is capable of performing any computing task, but it is all too easy to write unreadable macros that cause more trouble than they save work."

Theory of Operation

KAPIAR consists of two large stacks (BUF and EVALST), a large macro table (TABLE), two small stacks (ARGSTK and CALLST), a large procedure (EVAL), the mainline driver, and a supporting cast of service routines. (See Appendix.)

The driver corresponds to the subroutine MACRO of *Software Tools*, pp 270-71, and does the scanning. EVAL (p. 273) does the text-replacement and parameter-substitution.

Macro definitions are entered in TABLE by DODEF and INSTAL. The format of the representation in TABLE is:

'<name> ~<definition> ~', where tilde ' ~ ' is the end-of-string marker.

Driver calls GETTOK, which loads either alphanumeric strings or single non-alphanumeric characters into TOKEN. If TOKEN is alphanumeric then driver calls LOOKUP to see whether or not TOKEN is a macro name in TABLE. If not, then driver calls PUTTOK. If we are not in the middle of evaluating some other macro then TOKEN is irrelevant to the macro business and PUTTOK sends it on to the outside world. Otherwise PUTTOK pushes it onto EVALST for further processing. If TOKEN *was* found by LOOKUP in TABLE then the name and definition are pushed (by PUTTOK) onto EVALST in reverse order: <definition> ~ <name> ~ and CP is incremented to record the level of macro nesting.

Now driver calls GETTOK again, and looks for a left hand paren to signal the beginning of the parm-list. (If it doesn't find one, it pushes an empty paren-pair back onto the bufferstack BUF and tries again.) The contents of the parmlist (possibly empty) are pushed back onto EVALST until a balanced right hand paren is encountered. At this point, EVALST looks like this (assuming that we are not nested in some other macro):

<definition> ~<name> ~<parm 1>, . . . ,<parm k> ~

Driver then calls EVAL, which begins by looking at the bottom of the stack: if <definition> corresponds to any of the special symbols DEFTYPE, INCTYPE, etc., the EVAL knows that this is a predefined macro and calls the appropriate subroutine (DODEF, DOINCR, etc.,) otherwise EVAL starts scanning <definition> backwards, from right to left, looking for ARGFLAG '$'. Any character which is not preceded by '$' is pushed back onto the bufferstack BUF. Anything which *is* preceded by '$' is regarded by EVAL to be a decimal digit d, 0 < = d < = 9, and the d-th string above <definition> in EVALST is pushed back instead. If d is zero then <name> is pushed back, otherwise <parm d> is pushed back. EVAL does some clever pointer-juggelry with the help of the short stacks ARGSTK and CALLST to avoid doing something stupid if the parmlist contains fewer actual parms then anticipated by <definition>. EVAL does rely on the user to indicate by consecutive commas the absence of initial or intermediate elements of the parmlist.

When EVAL has pushed back all the replacement text and/or real parms invoked by the macro call it returns to driver, which then flushes that portion of EVALST by resetting the pointers AP, CP and EP. Driver then resumes the scanning of input from BUF by calling GETTOK and the next cycle begins.

There is an exception to the above-described process: When the scan turns up a left hand square bracket, driver throws it away and goes looking for the matching right hand bracket, passing everything in between to PUTTOK without further analysis. When driver finds the matching right hand bracket it throws that away too, and then resumes the normal scanning process.

Appendix

The cast of characters in order of appearance:

OPEN, READ, WRITE, CLOSE and EXIT are ISIS subroutines for file I/O. Consult ISIS PROGRAMMER'S MANUAL for details.

PUTC takes one parm, CHR, byte, and writes that single character out to both the console and to :F1:KAPIAR.SAV. Called by REMARK, PBSTR, INSTAL and EVAL.

CRLF outputs a carriage-return and linefeed. Called by REMARK, EIGHTY and DISPLAY.

REMARK takes one parm, STR$PTR, address, outputs a crlf and the ascii string pointed to by STR$PTR. Called by GETC, PUTBAK, GETTOK, INSTAL, SEEKNBR, PUTCHR, PUTTOK, PUSH and driver.

EIGHTY takes one parm, CHR, byte, outputs crlf and 80 copies of CHR. Called by DISPLAY, GETC.

DISPLAY takes two parms, INDEX, ARRAY$PTR, addresses, outputs crlf, eighty dashes, all ascii characters of the array in the closed interval [head, tail], eighty dashes and a crlf. Called by GETC.

SIGNON outputs the sign-on message and sets up the run for either interactive or batch operation.

GETC returns the next character from the standard input buffer, calls SIGNON at startup time and updates the state-vector deque by calling SAVESTATES whenever a new bufferload of input arrives from the console. Also handles special control characters Control-A, Control-C and Control-D. Called by GETTOK, affects BUF and BP.

SAVESTATES maintains a deque of state-vectors for recovery purposes.

PUTBAK takes one parm, CHR, byte, pushes it back on the input bufferstack. Called by EVAL, PBSTR and GETTOK, affects BUF and BP.

PBSTR takes one parm, IN$PTR, address, calls PUTBAK for each character in the string pointed at by IN$PTR. Called by EVAL and MACRO, alters BUF, BP through the action of PUTBAK.

INDEX takes two parms, STR$PTR and CHR, address and byte. Returns zero if CHR not in STR, else returns the index of CHR in STR. Called by EVAL.

TYPE takes one parm, CHR, byte. Returns DIGIT if CHR is a digit, returns LETTER if CHR is a letter, otherwise returns CHR. The code for TYPE differs substantially from that given in "Tools". Called by GETTOK.

GETTOK takes two parms, TOK$PTR and TOKSIZ, address and a byte constant (upper limit on length of TOKEN). As a side-effect either loads TOKEN with alphanumeric string and EOS or with a single character. As a function, returns a byte value signifying which (ie., alpha string or single non-alpha). Called by driver, affects TOKEN.

SCOPY takes four parms, FROMPTR, I, TOOPTR, J, addresses, and copies the string beginning at .FROM(I) and ending with EOS to the position beginning at .TOO(J). ['TO' is a PLM reserved word, hence the spelling 'TOO'.] Called by INSTAL and LOOKUP, alters contents of TABLE and DEFN.

INSTAL takes two parms, NAME$PTR and DEFPTR, addresses, calls SCOPY to enter the strings beginning at .NAME and .DEFN in TABLE. Called by driver and DODEF, alters TABLE, NAMDEXINTBL (a 1-dimensional address array holding the initial locations (in TABLE) of the name-definition string-pairs), PLAST (an address variable indexing string entries in TABLE).

EQUAL takes two parms, STR1$PTR and STR2$PTR, addresses, and returns either YES or NO according as STR1 = STR2. Called by DELETE.

DELETE takes one parm, NAM$PTR, address, and removes the last definition of NAME from TABLE. Called by EVAL, alters TABLE.

PUTCHR takes one parm, CHR, byte. If CP is zero then passes CHR to output by calling PUTC, else pushes CHR onto EVALST and bumps EP. Called by driver and PUTTOK, alters EVALST and EP.

PUTTOK takes one parm, STR$PTR, address, is an elaboration of PUTCHR to handle strings. See comments, "Tools" pp. 281ff. Called by driver, alters EVALST and EP.

PUSH takes three parms, EP, ARGSTK$PTR, AP, addresses, pushes current value of EP onto ARGSTK and bumps AP. Called from driver.

LOOKUP takes two parms, NAME$PTR and DEFN$PTR, addresses, looks to see if NAME is entered in TABLE. If so, copies <definition> to the temp DEFN by calling SCOPY. Called by driver, alters DEFN.

EVAL takes three parms, ARGSTKPTR, I, J, addresses. Contains eight of its own private procedures, DODEF, DOIF, CTOI, PBNUM, DOINCR, DODECR, MIN and DOSUBSTR. Consult THEORY OF OPERATION above for details. Called by driver, alters TABLE (by calling DODEF) or BUF and BP. The driver is analyzed in THEORY OF OPERATION above.

RECOVER is a non-procedure which technically belongs to driver, but is intuitively more like a subroutine. It is entered by a GOTO from any of GETTOK, INSTAL, PUTCHR, PUTTOK, PUSH and driver. It pops the tops of the various state-vector deques to their target pointers, moves everything (in each deque) up one notch, enters the program-entry initial values at the bottom of each deque, and jumps to RESTART in driver. The deques preserve the values of the critical pointers (AP, CP, EP, PLAST, TLAST and STACKPTR) for the previous 32 linebuffer inputs from the console. (See GETC above.) Whenever a stack blows up, RECOVER is jumped to, and the state of KAPIAR restored to that preceding the last bufferload of input from the console.

For Further Reading

Brown, P. J., *Macro Processors and Techniques for Portable Software*, Wiley, London, 1974.

Campbell-Kelly, M. *An Introduction to Macros*, American-Elsevier, New York, 1973.

Cole, A. J., *Macro Processors*, Cambridge Computer Science Texts 4, Cambridge University Press, Cambridge, 1976.

Kernighan, Brian W. and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading, MA, 1976.

Mooers, C. N. "TRAC, A Procedure-Describing Language for the Reactive Typewriter", *Communications of the ACM* 9:3, (March, 1966), pp 215-219.

Strachey, C. "A General Purpose Macrogenerator," *Computer Journal*, 8:3, (March 1965) pp. 225-41.

Wegner, P., *Programming Languages, Information Structures and Machine Organization*, McGraw-Hill, New York, 1968.

Waite, William M., *Implementing Software for Non-Numeric Applications*, Prentice-Hall, Englewood Cliffs, N. J., 1973.

```
                /*   I S I S    S Y S T E M    D E C L A R A T I O N S     */


2   1       OPEN: PROCEDURE(AFT, FILE, ACCESS, MODE, STATUS)EXTERNAL;
3   2             DECLARE (AFT, FILE, ACCESS, MODE, STATUS) ADDRESS;
4   2             END OPEN;


5   1       READ: PROCEDURE(AFT, BUFFER, COUNT, ACTUAL, STATUS) EXTERNAL;
6   2             DECLARE (AFT, BUFFER, COUNT, ACTUAL, STATUS) ADDRESS;
7   2             END READ;


8   1       WRITE: PROCEDURE(AFT, BUFFER, COUNT, STATUS) EXTERNAL;
9   2              DECLARE (AFT, BUFFER, COUNT, STATUS) ADDRESS;
10  2              END WRITE;


11  1       CLOSE: PROCEDURE(AFT, STATUS) EXTERNAL;
12  2              DECLARE (AFT, STATUS)ADDRESS;
13  2              END CLOSE;


14  1       EXIT: PROCEDURE EXTERNAL;
15  2             DECLARE STATUS ADDRESS;
16  2             END EXIT;


                /*    M A C R O S    L I T E R A L L Y    C O N S T A N T S     */


17  1        DECLARE IS LITERALLY 'LITERALLY', MAXPTR IS '256', BUFSIZE IS '4096',
                   ARGFLAG IS '24H', CR IS '0DH', LF IS '0AH',
                   EOS IS '0FEH', ALPHA IS '41H', EVALSIZE IS '4096',
                   LETTER IS '41H', DIGIT IS '0', DEFTYPE IS '01FH',
                   MAXLINE IS '80', EOFF IS '0FFH', MAXTBL IS '4096',
                   MAXTOK IS '122', MAXDEF IS '1024', EQUALS IS '3DH',
                   LPAREN IS '28H', RPAREN IS '29H', STAR IS '2AH',
                   LBRACK IS '5BH', RBRACK IS '5DH', DASH IS '2DH',
                   COMMA IS '2CH', YES IS '1', NO IS '0',IFTYPE IS '01EH',
                   CALLSIZE IS '256', ARGSIZE IS '1024', DELETYPE IS '01DH',
                   INCRTYPE IS '01CH', DECRTYPE IS '01AH',
                   SUBSTRTYPE IS '019H', BLANK IS '20H';


                /*     G L O B A L    D E C L A R A T I O N S      */


18  1        DECLARE (BUFACT, INSTAT, OUTACT, OUTSTAT, INPAFT, SAVAFT, STATUS,
                   AP, BP, CP, EP, HEAD, TAIL, PLAST, STKSAVO, TLAST) ADDRESS,
                   TABLE(MAXTBL) BYTE, BUF(BUFSIZE) BYTE, INTERACTIVE BYTE,
                   EVALST(EVALSIZE) BYTE, NAMDEXINTBL(256) ADDRESS,
                   DEFN(MAXDEF) BYTE, ARGSTK(ARGSIZE) ADDRESS,
                   (APSAV, CPSAV, EPSAV, PLASTSAV, TLASTSAV, STKPTRSAV) (32)
                   ADDRESS, START BYTE, TOKEN(MAXTOK) BYTE, INFILNAM(15) BYTE,
                   CALLST(CALLSIZE) ADDRESS, PLEV(CALLSIZE) BYTE,
                   BALP(3) BYTE DATA ('(',')', EOS), DEFNAM(7) BYTE DATA
                   ('D', 'E', 'F', 'I', 'N', 'E', EOS), DEFTYP(2) BYTE DATA
                   (DEFTYPE, EOS), CONDNAM(7) BYTE DATA ('I','F','E','L','S','E',EOS),
                   CONDTYP(2) BYTE DATA (IFTYPE,EOS), (T, NLB) BYTE,
                   DELTNAM(7) BYTE DATA ('D','E','L','E','T','E',EOS),
                   DELTYP(2) BYTE DATA (DELETYPE,EOS), INCRNAM(5) BYTE DATA
                   ('I','N','C','R',EOS), DECRNAM(5) BYTE DATA
                   ('D','E','C','R',EOS), SUBSTRNAM(7) BYTE DATA
                   ('S','U','B','S','T','R',EOS), INCRTYP(2) BYTE DATA
                   (INCRTYPE,EOS), DECRTYP(2) BYTE DATA (DECRTYPE,EOS),
                   SUBSTRTYP(2) BYTE DATA (SUBSTRTYPE,EOS);


                /*         P R O C E D U R E    D E C L A R A T I O N S          */


19  1        PUTC: PROCEDURE (CHR);
                                                ; STATEMENT # 19
                    , PROC  PUTC
          0678  212F43          LXI     H,CHR
          067B  71              MOV     M,C
20  2             DECLARE CHR BYTE, (OUTACT, OUTSTAT) ADDRESS;

21  2             OUTACT = 1;
                                                ; STATEMENT # 21
          067C  210100          LXI     H,1H
          067F  223043          SHLD    OUTACT
22  2             CALL WRITE(SAVAFT, .CHR, OUTACT, .OUTSTAT);
                                                ; STATEMENT # 22
          0682  2A0C00          LHLD    SAVAFT
          0685  E5              PUSH    H       ; 1
          0686  012F43          LXI     B,CHR
          0689  C5              PUSH    B       ; 2
          068A  2A3043          LHLD    OUTACT
          068D  44              MOV     B,H
          068E  4D              MOV     C,L
          068F  113243          LXI     D,OUTSTAT
          0692  CD0000          CALL    WRITE
23  2             IF INTERACTIVE THEN
                                                ; STATEMENT # 23
          0695  3A2220          LDA     INTERACTIVE
          0698  1F              RAR
          0699  D2AF06          JNC     @1
24  2                 CALL WRITE(0, .CHR, .OUTACT, .OUTSTAT);
                                                ; STATEMENT # 24
          069C  010000          LXI     B,0H
          069F  C5              PUSH    B       ; 1
          06A0  012F43          LXI     B,CHR
          06A3  C5              PUSH    B       ; 2
          06A4  2A3043          LHLD    OUTACT
          06A7  44              MOV     B,H
          06A8  4D              MOV     C,L
          06A9  113243          LXI     D,OUTSTAT
          06AC  CD0000          CALL    WRITE
                    @1
25  2             RETURN;
                                                ; STATEMENT # 25
          06AF  C9              RET
26  2             END PUTC;
                                                ; STATEMENT # 26


27  1        LENGTH: PROCEDURE(STR$PTR) ADDRESS; /* COMPUTE LENGTH OF STRING */
                                                ; STATEMENT # 27
                    , PROC  LENGTH
          06B0  213543          LXI     H,STRPTR+1H
          06B3  70              MOV     M,B
          06B4  2B              DCX     H
          06B5  71              MOV     M,C
28  2             DECLARE STR$PTR ADDRESS, STR BASED STR$PTR (1) BYTE, LNGTH ADDRESS;

29  2             LNGTH = 0;
                                                ; STATEMENT # 29
          06B8  210000          LXI     H,0H
          06BB  223643          SHLD    LNGTH
30  2             DO WHILE STR(LNGTH) <> EOS;
                                                ; STATEMENT # 30
                    @100:
          06BE  2A3643          LHLD    LNGTH
          06BF  EB              XCHG
          06C0  2A3443          LHLD    STRPTR
          06C3  19              DAD     D
          06C4  7E              MOV     A,M
          06C5  FEFE            CPI     0FEH
          06C7  CAD406          JZ      @101
31  3               LNGTH = LNGTH + 1;
                                                ; STATEMENT # 31
          06CA  2A3643          LHLD    LNGTH
          06CD  23              INX     H
          06CE  223643          SHLD    LNGTH
32  3             END /* DO-WHILE */;
                                                ; STATEMENT # 32
          06D1  C3BE06          JMP     @100
                    @101:
33  2             RETURN LNGTH;
                                                ; STATEMENT # 33
          06D4  2A3643          LHLD    LNGTH
          06D7  C9              RET
34  2             END LENGTH;
                                                ; STATEMENT # 34


35  1        CRLF: PROCEDURE;
                                                ; STATEMENT # 35
                    , PROC  CRLF
36  2             CALL PUTC(CR);
                                                ; STATEMENT # 36
          06D8  0E0D            MVI     C,0DH
          06DA  CD7806          CALL    PUTC
37  2             CALL PUTC(LF);
                                                ; STATEMENT # 37
          06DD  0E0A            MVI     C,0AH
          06DF  CD7806          CALL    PUTC
38  2             END CRLF;
                                                ; STATEMENT # 38
          06E2  C9              RET
```

```
39  1       REMARK: PROCEDURE(STR$PTR); /*  GENERAL-PURPOSE MESSAGE PRINTER  */
                                        ; STATEMENT # 39
                  ; PROC  REMARK
        06E8  213943        LXI     H,STRPTR+1H
        06E5  70            MOV     M,B
        06E7  2B            DCX     H
        06E8  71            MOV     M,C
40  2               DECLARE (STR$PTR, STAT) ADDRESS, STRING BASED STR$PTR (1) BYTE,
                        L ADDRESS;

41  2               L = LENGTH( STRING);
                                        ; STATEMENT # 41
        06E9  2A3843        LHLD    STRPTR
        06EC  44            MOV     B,H
        06ED  4D            MOV     C,L
        06EE  CDB006        CALL    LENGTH
        06F1  223C43        SHLD    L
42  2               CALL CRLF;
                                        ; STATEMENT # 42
        06F4  CDD806        CALL    CRLF
43  2               CALL WRITE(SAVAFT, STR$PTR, L, STAT);
                                        ; STATEMENT # 43
        06F7  2A0C00        LHLD    SAVAFT
        06FA  E5            PUSH    H       ; 1
        06FB  2A3843        LHLD    STRPTR
        06FE  E5            PUSH    H       ; 2
        06FF  2A3C43        LHLD    L
        0702  44            MOV     B,H
        0703  4D            MOV     C,L
        0704  113A43        LXI     D,STAT
        0707  CD0000        CALL    WRITE
44  2               CALL WRITE(0, STR$PTR, L, STAT);
                                        ; STATEMENT # 44
        070A  010000        LXI     B,0H
        070D  C5            PUSH    B       ; 1
        070E  2A3843        LHLD    STRPTR
        0711  E5            PUSH    H       ; 2
        0712  2A3C43        LHLD    L
        0715  44            MOV     B,H
        0716  4D            MOV     C,L
        0717  113A43        LXI     D,STAT
        071A  CD0000        CALL    WRITE
45  2               RETURN;
                                        ; STATEMENT # 45
        071D  C9            RET
46  2               END REMARK;
                                        ; STATEMENT # 46


47  1       EIGHTY: PROCEDURE(CHR);   /*  PUT OUT A LINE OF 80 CHRS  */
                                        ; STATEMENT # 47
                  ; PROC  EIGHTY
        071E  213E43        LXI     H,CHR
        0721  71            MOV     M,C
48  2               DECLARE (CHR, I) BYTE;

49  2               CALL CRLF;
                                        ; STATEMENT # 49
        0722  CDD806        CALL    CRLF
50  2               DO I = 1 TO 80;
                                        ; STATEMENT # 50
        0725  213F43        LXI     H,I
        0728  3601          MVI     M,1H
                        @102:
        072A  3E50          MVI     A,50H
        072C  213F43        LXI     H,I
        072F  BE            CMP     M
        0730  DA4107        JC      @103
51  3             CALL PUTC(CHR);
                                        ; STATEMENT # 51
        0733  2A3E43        LHLD    CHR
        0736  4D            MOV     C,L
        0737  CD7806        CALL    PUTC
52  3               END;
                                        ; STATEMENT # 52
        073A  213F43        LXI     H,I
        073D  34            INR     M
        073E  C22A07        JNZ     @102
                        @103:
53  2               RETURN;
                                        ; STATEMENT # 53
        0741  C9            RET
54  2               END EIGHTY;
                                        ; STATEMENT # 54
```

```
55  1       DISPLAY: PROCEDURE (INDEX, ARRAY$PTR);    /*  SHOW CONTENTS OF ARRAY  */
                                        ; STATEMENT # 55
                  PROC  DISPLAY
        0742  214343        LXI     H,ARRAYPTR+1H
        0745  72            MOV     M,D
        0746  2B            DCX     H
        0747  73            MOV     M,E
        0748  2B            DCX     H
        0749  70            MOV     M,B
        074A  2B            DCX     H
        074B  71            MOV     M,C
56  2               DECLARE (ARRAY$PTR, I, INDEX) ADDRESS,
                        ARRAY BASED ARRAY$PTR (1) BYTE,
                        (CHR, LINE) BYTE;

57  2               CALL EIGHTY(DASH);
                                        ; STATEMENT # 57
        074C  0E2D          MVI     C,2DH
        074E  CD1E07        CALL    EIGHTY
58  2               CALL CRLF;
                                        ; STATEMENT # 58
        0751  CDD806        CALL    CRLF
59  2               HEAD, LINE = 0;
                                        ; STATEMENT # 59
        0754  210000        LXI     H,0H
        0757  221800        SHLD    HEAD
        075A  7D            MOV     A,L
        075B  324743        STA     LINE
60  2               TAIL = INDEX;
                                        ; STATEMENT # 60
        075E  2A4043        LHLD    INDEX
        0761  221A00        SHLD    TAIL
61  2               DO I = HEAD TO TAIL;
                                        ; STATEMENT # 61
        0764  2A1800        LHLD    HEAD
        0767  224443        SHLD    I
                        @104:
        076A  111A00        LXI     D,TAIL
        076D  014443        LXI     B,I
        0770  CD0000        CALL    @P0098
        0773  DABA07        JC      @105
62  3             CHR = ARRAY(I);
                                        ; STATEMENT # 62
        0776  2A4443        LHLD    I
        0779  EB            XCHG
        077A  2A4243        LHLD    ARRAYPTR
        077D  19            DAD     D
        077E  7E            MOV     A,M
        077F  324643        STA     CHR
63  3             CALL PUTC(CHR);
                                        ; STATEMENT # 63
        0782  2A4643        LHLD    CHR
        0785  4D            MOV     C,L
        0786  CD7806        CALL    PUTC
64  3             IF (CHR = CR) THEN
                                        ; STATEMENT # 64
        0789  3A4643        LDA     CHR
        078C  FE0D          CPI     0DH
        078E  C29607        JNZ     @2
65  3               LINE = 0;
                                        ; STATEMENT # 65
        0791  214743        LXI     H,LINE
        0794  3600          MVI     M,0H
                        @2:
66  3             IF LINE < 79 THEN
                                        ; STATEMENT # 66
        0796  3A4743        LDA     LINE
        0799  FE4F          CPI     4FH
        079B  D2A507        JNC     @3
67  3               LINE = LINE + 1;
                                        ; STATEMENT # 67
        079E  214743        LXI     H,LINE
        07A1  34            INR     M
        07A2  C3AD07        JMP     @4
                        @3:
                ELSE  /*  LINE IS FULL  */
68  3               DO;
69  4               CALL CRLF;
                                        ; STATEMENT # 69
        07A5  CDD806        CALL    CRLF
70  4               LINE = 0;
                                        ; STATEMENT # 70
        07A8  214743        LXI     H,LINE
        07AB  3600          MVI     M,0H
71  4               END;
                                        ; STATEMENT # 71
```

```
                        @4:
72  3                   END /*  DO I = HEAD ...  */;
                                                    ; STATEMENT # 72
        07AD  110100            LXI     D,1H
        07B0  2A4443            LHLD    I
        07B3  19                DAD     D
        07B4  224443            SHLD    I
        07B7  D26A07            JNC     @104
                        @105:
73  2                   CALL EIGHTY(DASH);
                                                    ; STATEMENT # 73
        07BA  0E2D              MVI     C,2DH
        07BC  CD1E07            CALL    EIGHTY
74  2                   CALL CRLF;
                                                    ; STATEMENT # 74
        07BF  CDD806            CALL    CRLF
75  2                   RETURN;
                                                    ; STATEMENT # 75
        07C2  C9                RET
76  2                   END DISPLAY;
                                                    ; STATEMENT # 76


77  1       SIGNON: PROCEDURE;    /*  SET/CLEAR INTERACTIVE FLAG  */
                                                    ; STATEMENT # 77
                    ; PROC  SIGNON
78  2                   DECLARE I BYTE;

79  2                   DO I = 0 TO 14;
                                                    ; STATEMENT # 79
        07C3  214843            LXI     H,I
        07C6  3600              MVI     M,0H
                        @106:
        07C8  3E0E              MVI     A,0EH
        07CA  214843            LXI     H,I
        07CD  BE                CMP     M
        07CE  DAE307            JC      @107
80  3                     INFILNAM(I) = 20H;    /*  BLANK IT OUT  */
                                                    ; STATEMENT # 80
        07D1  2A4843            LHLD    I
        07D4  2600              MVI     H,0
        07D6  011E40            LXI     B,INFILNAM
        07D9  09                DAD     B
        07DA  3620              MVI     M,20H
81  3                   END;
                                                    ; STATEMENT # 81
        07DC  214843            LXI     H,I
        07DF  34                INR     M
        07E0  C20807            JNZ     @106
                        @107:
82  2                   INTERACTIVE = 0FFH;
                                                    ; STATEMENT # 82
        07E3  212220            LXI     H,INTERACTIVE
        07E6  36FF              MVI     M,0FFH
83  2                   CALL READ(1, .BUF, 128, .BUFACT, .INSTAT);    /*  FLUSH BUFFER  */
                                                    ; STATEMENT # 83
        07E8  010100            LXI     B,1H
        07EB  C5                PUSH    B       ; 1
        07EC  012210            LXI     B,BUF
        07EF  C5                PUSH    B       ; 2
        07F0  018000            LXI     B,80H
        07F3  C5                PUSH    B       ; 3
        07F4  110400            LXI     D,INSTAT
        07F7  010200            LXI     B,BUFACT
        07FA  CD0000            CALL    READ
84  2                   CALL CRLF;
                                                    ; STATEMENT # 84
        07FD  CDD806            CALL    CRLF
85  2                   CALL REMARK(
                                                    ; STATEMENT # 85
        0800  014000            LXI     B,$-7C0H
        0803  CDE306            CALL    REMARK
            ('FOR BATCH-MODE OPERATION, RESPOND BY ENTERING NAME OF FILE TO BE PROCESSED', EOS));
86  2                   CALL REMARK(
                                                    ; STATEMENT # 86
        0806  018B00            LXI     B,$-77BH
        0809  CDE306            CALL    REMARK
          ('(AND HIT RETURN), OTHERWISE HIT RETURN   T W I C E', EOS));
87  2                   CALL CRLF;
                                                    ; STATEMENT # 87
        080C  CDD806            CALL    CRLF
88  2                   CALL READ(1, .INFILNAM, 15, .BUFACT, .INSTAT);
                                                    ; STATEMENT # 88
        080F  010100            LXI     B,1H
        0812  C5                PUSH    B       ; 1
        0813  011E40            LXI     B,INFILNAM
        0816  C5                PUSH    B       ; 2
        0817  010F00            LXI     B,0FH
        081A  C5                PUSH    B       ; 3
        081B  110400            LXI     D,INSTAT
        081E  010200            LXI     B,BUFACT
        0821  CD0000            CALL    READ
89  2                   CALL READ(1, .BUF, 113, .BUFACT, .INSTAT);
                                                    ; STATEMENT # 89
        0824  010100            LXI     B,1H
        0827  C5                PUSH    B       ; 1
        0828  012210            LXI     B,BUF
        082B  C5                PUSH    B       ; 2
        082C  017100            LXI     B,71H
        082F  C5                PUSH    B       ; 3
        0830  110400            LXI     D,INSTAT
        0833  010200            LXI     B,BUFACT
        0836  CD0000            CALL    READ
90  2                   IF (INFILNAM(0) AND 7FH) <> 0DH THEN
                                                    ; STATEMENT # 90
        0839  3A1E40            LDA     INFILNAM
        083C  E67F              ANI     7FH
        083E  FE0D              CPI     0DH
        0840  CA5D08            JZ      @5
91  2                     DO;
92  3                       CALL OPEN(.INPAFT, .INFILNAM, 1, 0, .STATUS);
                                                    ; STATEMENT # 92
        0843  010A00            LXI     B,INPAFT
        0846  C5                PUSH    B       ; 1
        0847  011E40            LXI     B,INFILNAM
        084A  C5                PUSH    B       ; 2
        084B  010100            LXI     B,1H
        084E  C5                PUSH    B       ; 3
        084F  110E00            LXI     D,STATUS
        0852  010000            LXI     B,0H
        0855  CD0000            CALL    OPEN
93  3                       INTERACTIVE = 0;
                                                    ; STATEMENT # 93
        0858  212220            LXI     H,INTERACTIVE
        085B  3600              MVI     M,0H
94  3                     END; /*  IF-DO  */
                                                    ; STATEMENT # 94
                        @5:
95  2                   RETURN;
                                                    ; STATEMENT # 95
        085D  C9                RET
96  2                   END SIGNON;
                                                    ; STATEMENT # 96

97  1       GETC: PROCEDURE BYTE;    /* RETURNS NEXT CHAR FROM STD INPUT BUFFER  */
                                                    ; STATEMENT # 97
                    ; PROC  GETC
98  2                 DECLARE (CHR, I) BYTE;

99  2                 SAVESTATES: PROCEDURE;
                                                    ; STATEMENT # 99
                      ; PROC  SAVESTATES
100 3                   DECLARE I BYTE;

101 3                   DO I = 0 TO 30;   /*  PUSH ALL VALUES DOWN ONE  */
                                                    ; STATEMENT # 101
        0943  214B43            LXI     H,I
        0946  3600              MVI     M,0H
                        @108:
        0948  3E1E              MVI     A,1EH
        094A  214B43            LXI     H,I
        094D  BE                CMP     M
        094E  DA060A            JC      @109
102 4                     APSAV(I) = APSAV(I+1);
                                                    ; STATEMENT # 102
        0951  2A4B43            LHLD    I
        0954  2600              MVI     H,0
        0956  01253E            LXI     B,APSAV+2H
        0959  29                DAD     H
        095A  09                DAD     B
        095B  E5                PUSH    H       ; 1
        095C  2A4B43            LHLD    I
        095F  2600              MVI     H,0
        0961  01233E            LXI     B,APSAV
        0964  29                DAD     H
        0965  09                DAD     B
        0966  E3                XTHL            ; 1
        0967  4E                MOV     C,M
        0968  23                INX     H
        0969  46                MOV     B,M
        096A  E1                POP     H       ; 1
        096B  71                MOV     M,C
        096C  23                INX     H
        096D  70                MOV     M,B
```

```
103  4                          CPSAV(I) = CPSAV(I+1);
                                        ; STATEMENT # 103
        096E  2A4B43        LHLD    I
        0971  2600          MVI     H,0
        0973  01653E        LXI     B,CPSAV+2H
        0976  29            DAD     H
        0977  09            DAD     B
        0978  E5            PUSH    H       ; 1
        0979  2A4B43        LHLD    I
        097C  2600          MVI     H,0
        097E  01633E        LXI     B,CPSAV
        0981  29            DAD     H
        0982  09            DAD     B
        0983  E3            XTHL            ; 1
        0984  4E            MOV     C,M
        0985  23            INX     H
        0986  46            MOV     B,M
        0987  E1            POP     H       ; 1
        0988  71            MOV     M,C
        0989  23            INX     H
        098A  70            MOV     M,B
104  4                          EPSAV(I) = EPSAV(I+1);
                                        ; STATEMENT # 104
        098B  2A4B43        LHLD    I
        098E  2600          MVI     H,0
        0990  01A53E        LXI     B,EPSAV+2H
        0993  29            DAD     H
        0994  09            DAD     B
        0995  E5            PUSH    H       ; 1
        0996  2A4B43        LHLD    I
        0999  2600          MVI     H,0
        099B  01A33E        LXI     B,EPSAV
        099E  29            DAD     H
        099F  09            DAD     B
        09A0  E3            XTHL            ; 1
        09A1  4E            MOV     C,M
        09A2  23            INX     H
        09A3  46            MOV     B,M
        09A4  E1            POP     H       ; 1
        09A5  71            MOV     M,C
        09A6  23            INX     H
        09A7  70            MOV     M,B
105  4                          PLASTSAV(I) = PLASTSAV(I+1);
                                        ; STATEMENT # 105
        09A8  2A4B43        LHLD    I
        09AB  2600          MVI     H,0
        09AD  01E53E        LXI     B,PLASTSAV+2H
        09B0  29            DAD     H
        09B1  09            DAD     B
        09B2  E5            PUSH    H       ; 1
        09B3  2A4B43        LHLD    I
        09B6  2600          MVI     H,0
        09B8  01E33E        LXI     B,PLASTSAV
        09BB  29            DAD     H
        09BC  09            DAD     B
        09BD  E3            XTHL            ; 1
        09BE  4E            MOV     C,M
        09BF  23            INX     H
        09C0  46            MOV     B,M
        09C1  E1            POP     H       ; 1
        09C2  71            MOV     M,C
        09C3  23            INX     H
        09C4  70            MOV     M,B
106  4                          TLASTSAV(I) = TLASTSAV(I+1);
                                        ; STATEMENT # 106
        09C5  2A4B43        LHLD    I
        09C8  2600          MVI     H,0
        09CA  01253F        LXI     B,TLASTSAV+2H
        09CD  29            DAD     H
        09CE  09            DAD     B
        09CF  E5            PUSH    H       ; 1
        09D0  2A4B43        LHLD    I
        09D3  2600          MVI     H,0
        09D5  01233F        LXI     B,TLASTSAV
        09D8  29            DAD     H
        09D9  09            DAD     B
        09DA  E3            XTHL            ; 1
        09DB  4E            MOV     C,M
        09DC  23            INX     H
        09DD  46            MOV     B,M
        09DE  E1            POP     H       ; 1
        09DF  71            MOV     M,C
        09E0  23            INX     H
        09E1  70            MOV     M,B
```

```
107  4                          STKPTRSAV(I) = STKPTRSAV(I+1);
                                        ; STATEMENT # 107
        09E2  2A4B43        LHLD    I
        09E5  2600          MVI     H,0
        09E7  01E53F        LXI     B,STKPTRSAV+2H
        09EA  29            DAD     H
        09EB  09            DAD     B
        09EC  E5            PUSH    H       ; 1
        09ED  2A4B43        LHLD    I
        09F0  2600          MVI     H,0
        09F2  01E33F        LXI     B,STKPTRSAV
        09F5  29            DAD     H
        09F6  09            DAD     B
        09F7  E3            XTHL            ; 1
        09F8  4E            MOV     C,M
        09F9  23            INX     H
        09FA  46            MOV     B,M
        09FB  E1            POP     H       ; 1
        09FC  71            MOV     M,C
        09FD  23            INX     H
        09FE  70            MOV     M,B
108  4                          END; /*  DO-LOOP,  I = 31  */
                                        ; STATEMENT # 108
        09FF  214B43        LXI     H,I
        0A02  34            INR     M
        0A03  C24809        JNZ     @108
                    @109
109  3                          APSAV(I) = AP;    /*  TOP OF STACKS GET CURRENT VALUES  */
                                        ; STATEMENT # 109
        0A06  2A4B43        LHLD    I
        0A09  2600          MVI     H,0
        0A0B  01233E        LXI     B,APSAV
        0A0E  29            DAD     H
        0A0F  09            DAD     B
        0A10  E5            PUSH    H       ; 1
        0A11  2A1000        LHLD    AP
        0A14  EB            XCHG
        0A15  E1            POP     H       ; 1
        0A16  73            MOV     M,E
        0A17  23            INX     H
        0A18  72            MOV     M,D
110  3                          CPSAV(I) = CP;
                                        ; STATEMENT # 110
        0A19  2A4B43        LHLD    I
        0A1C  2600          MVI     H,0
        0A1E  01633E        LXI     B,CPSAV
        0A21  29            DAD     H
        0A22  09            DAD     B
        0A23  E5            PUSH    H       ; 1
        0A24  2A1400        LHLD    CP
        0A27  EB            XCHG
        0A28  E1            POP     H       ; 1
        0A29  73            MOV     M,E
        0A2A  23            INX     H
        0A2B  72            MOV     M,D
111  3                          EPSAV(I) = EP;
                                        ; STATEMENT # 111
        0A2C  2A4B43        LHLD    I
        0A2F  2600          MVI     H,0
        0A31  01A33E        LXI     B,EPSAV
        0A34  29            DAD     H
        0A35  09            DAD     B
        0A36  E5            PUSH    H       ; 1
        0A37  2A1600        LHLD    EP
        0A3A  EB            XCHG
        0A3B  E1            POP     H       ; 1
        0A3C  73            MOV     M,E
        0A3D  23            INX     H
        0A3E  72            MOV     M,D
112  3                          PLASTSAV(I) = PLAST;
                                        ; STATEMENT # 112
        0A3F  2A4B43        LHLD    I
        0A42  2600          MVI     H,0
        0A44  01E33E        LXI     B,PLASTSAV
        0A47  29            DAD     H
        0A48  09            DAD     B
        0A49  E5            PUSH    H       ; 1
        0A4A  2A1C00        LHLD    PLAST
        0A4D  EB            XCHG
        0A4E  E1            POP     H       ; 1
        0A4F  73            MOV     M,E
        0A50  23            INX     H
        0A51  72            MOV     M,D
```

```
113  3                      TLASTSAV(I) = TLAST;
                                        ; STATEMENT # 113
     0A52  2A4B43       LHLD    I
     0A55  2600         MVI     H,0
     0A57  01233F       LXI     B,TLASTSAV
     0A5A  29           DAD     H
     0A5B  09           DAD     B
     0A5C  E5           PUSH    H       ; 1
     0A5D  2A2000       LHLD    TLAST
     0A60  EB           XCHG
     0A61  E1           POP     H       ; 1
     0A62  73           MOV     M,E
     0A63  23           INX     H
     0A64  72           MOV     M,D
114  3                      STKPTRSAV(I) = STACKPTR;   /* PLM PREDECLARED */
                                        ; STATEMENT # 114
     0A65  2A4B43       LHLD    I
     0A68  2600         MVI     H,0
     0A6A  01633F       LXI     B,STKPTRSAV
     0A6D  29           DAD     H
     0A6E  09           DAD     B
     0A6F  E5           PUSH    H       ; 1
     0A70  210000       LXI     H,0     ; 1
     0A73  39           DAD     SP
     0A74  EB           XCHG
     0A75  E1           POP     H       ; 1
     0A76  73           MOV     M,E
     0A77  23           INX     H
     0A78  72           MOV     M,D
115  3                      RETURN;
                                        ; STATEMENT # 115
     0A79  C9           RET
116  3                  END SAVESTATES;
                                        ; STATEMENT # 116


117  2          RIGHT$JUSTIFY: PROCEDURE (COUNT); /* MOVE INPUT TO HIGH END OF BUF */
                                        ; STATEMENT # 117
                ; PROC  RIGHTJUSTIFY
     0A7A  214D43       LXI     H,COUNT+1H
     0A7D  70           MOV     M,B
     0A7E  2B           DCX     H
     0A7F  71           MOV     M,C
118  3                      DECLARE (COUNT) ADDRESS;

119  3                      BUF(COUNT) = EOS;
                                        ; STATEMENT # 119
     0A80  2A4C43       LHLD    COUNT
     0A83  012210       LXI     B,BUF
     0A86  09           DAD     B
     0A87  36FE         MVI     M,0FEH
120  3                      DO I = 0 TO COUNT;
                                        ; STATEMENT # 120
     0A89  214A43       LXI     H,I
     0A8C  3600         MVI     M,0H
                @110:
     0A8E  114C43       LXI     D,COUNT
     0A91  3A4A43       LDA     I
     0A94  CD0000       CALL    @P0101
     0A97  DABF0A       JC      @111
121  4                      BUF(BUFSIZE-1 - I) = BUF(COUNT - I);
                                        ; STATEMENT # 121
     0A9A  3A4A43       LDA     I
     0A9D  114C43       LXI     D,COUNT
     0AA0  CD0000       CALL    @P0101
     0AA3  012210       LXI     B,BUF
     0AA6  09           DAD     B
     0AA7  3A4A43       LDA     I
     0AAA  11FF0F       LXI     D,0FFFH
     0AAD  E5           PUSH    H       ; 1
     0AAE  CD0000       CALL    @P0096
     0AB1  012210       LXI     B,BUF
     0AB4  09           DAD     B
     0AB5  C1           POP     B       ; 1
     0AB6  0A           LDAX    B
     0AB7  77           MOV     M,A
122  4                      END;
                                        ; STATEMENT # 122
     0AB8  214A43       LXI     H,I
     0ABB  34           INR     M
     0ABC  C28E0A       JNZ     @110
                @111:
```

```
123  3                      BP = BUFSIZE - BUFACT - 1; /* -1 TO
                                                ACCOUNT FOR THE EOS */
                                        ; STATEMENT # 123
     0ABF  2A0200       LHLD    BUFACT
     0AC2  110010       LXI     D,1000H
     0AC5  CD0000       CALL    @P0095
     0AC8  2B           DCX     H
     0AC9  221200       SHLD    BP
124  3                      RETURN;
                                        ; STATEMENT # 124
     0ACC  C9           RET
125  3                  END RIGHT$JUSTIFY;
                                        ; STATEMENT # 125


                /* G E T C   E X E C U T I O N   B E G I N S   H E R E */


126  2              IF BUF(BP) = EOS THEN
                                        ; STATEMENT # 126
     085E  2A1200       LHLD    BP
     0861  012210       LXI     B,BUF
     0864  09           DAD     B
     0865  7E           MOV     A,M
     0866  FEFE         CPI     0FEH
     0868  C2F008       JNZ     @6
127  2                  DO;
128  3                    IF START THEN
                                        ; STATEMENT # 128
     086B  3AA33F       LDA     START
     086E  1F           RAR
     086F  D27808       JNC     @7
129  3                      CALL SAVESTATES;
                                        ; STATEMENT # 129
     0872  CD4309       CALL    SAVESTATES
     0875  C37D08       JMP     @8
                @7:
                     ELSE
130  3                      START = 1;
                                        ; STATEMENT # 130
     0878  21A33F       LXI     H,START
     087B  3601         MVI     M,1H
                @8:
131  3                    IF INTERACTIVE THEN
                                        ; STATEMENT # 131
     087D  3A2220       LDA     INTERACTIVE
     0880  1F           RAR
     0881  D2BD08       JNC     @9
132  3                      DO;
133  4                        OUTACT = 2;
                                        ; STATEMENT # 133
     0884  210200       LXI     H,2H
     0887  220600       SHLD    OUTACT
134  4                      CALL WRITE(0, .(' '), OUTACT, .OUTSTAT);
                                        ; STATEMENT # 134
     088A  010000       LXI     B,0H
     088D  C5           PUSH    B       ; 1
     088E  01BE00       LXI     B,$-7D0H
     0891  C5           PUSH    B       ; 2
     0892  2A0600       LHLD    OUTACT
     0895  44           MOV     B,H
     0896  4D           MOV     C,L
     0897  110800       LXI     D,OUTSTAT
     089A  CD0000       CALL    WRITE
135  4                      CALL READ(1, .BUF, 128, .BUFACT, .INSTAT);
                                        ; STATEMENT # 135
     089D  010100       LXI     B,1H
     08A0  C5           PUSH    B       ; 1
     08A1  012210       LXI     B,BUF
     08A4  C5           PUSH    B       ; 2
     08A5  018000       LXI     B,80H
     08A8  C5           PUSH    B       ; 3
     08A9  110400       LXI     D,INSTAT
     08AC  010200       LXI     B,BUFACT
     08AF  CD0000       CALL    READ
136  4                      CALL RIGHT$JUSTIFY(.BUFACT);
                                        ; STATEMENT # 136
     08B2  2A0200       LHLD    BUFACT
     08B5  44           MOV     B,H
     08B6  4D           MOV     C,L
     08B7  CD7A0A       CALL    RIGHTJUSTIFY
137  4                      END; /* IF-DO */
                                        ; STATEMENT # 137
     08BA  C3F008       JMP     @10
                @9:
                     ELSE
138  3                      DO;
```

```
139  4                    CALL READ(INPAFT, .BUF, 128, .BUFACT, .INSTAT);
                                        ; STATEMENT # 139
        08BD 2A0A00          LHLD    INPAFT
        08C0 E5              PUSH    H       ; 1
        08C1 012210          LXI     B,BUF
        08C4 C5              PUSH    B       ; 2
        08C5 018000          LXI     B,80H
        08C8 C5              PUSH    B       ; 3
        08C9 110400          LXI     D,INSTAT
        08CC 010200          LXI     B,BUFACT
        08CF CD0000          CALL    READ
140  4                    IF BUFACT <> 0 THEN
                                        ; STATEMENT # 140
        08D2 3E00            MVI     A,0H
        08D4 110200          LXI     D,BUFACT
        08D7 CD0000          CALL    @P0101
        08DA B5              ORA     L
        08DB CAE908          JZ      @11
141  4                      CALL RIGHT$JUSTIFY(BUFACT);
                                        ; STATEMENT # 141
        08DE 2A0200          LHLD    BUFACT
        08E1 44              MOV     B,H
        08E2 4D              MOV     C,L
        08E3 CD7A0A          CALL    RIGHTJUSTIFY
        08E6 C3F008          JMP     @12
                     @11:
                         ELSE
142  4                      DO;
143  5                        STACKPTR = STKSAV0;
                                        ; STATEMENT # 143
        08E9 2A1E00          LHLD    STKSAV0
        08EC F9              SPHL
.44  5                        GOTO CALLSIGNON;
                                        ; STATEMENT # 144
        08ED C33402          JMP     CALLSIGNON
145  5                      END; /* ELSE-DO */
                                        ; STATEMENT # 145
                     @12:
146  4                    END; /* ELSE-DO */
                                        ; STATEMENT # 146
                     @10:
147  3                  END;          /* IF BUF(BP) = EOS    */
                                        ; STATEMENT # 147
                     @:
148  2                  CHR = BUF(BP) AND 7FH    /* MASK OFF PARITY BIT  */
                                        ; STATEMENT # 148
        08F0 2A1200          LHLD    BP
        08F3 012210          LXI     B,BUF
        08F6 09              DAD     B
        08F7 3E7F            MVI     A,7FH
        08F9 A6              ANA     M
        08FA 324943          STA     CHR
149  2                      BP = BP + 1;
                                        ; STATEMENT # 149
        08FD 2A1200          LHLD    BP
        0900 23              INX     H
        0901 221200          SHLD    BP
150  2                      IF CHR = 1 THEN       /* CONTROL-A   A P P E N D S   THE RESULT OF
                                        ; STATEMENT # 150
        0904 FE01            CPI     1H
        0906 C20909          JNZ     @13
                                        ; STATEMENT # 151
                     @13:
                                        ONE STRING-MACRO TO THE PRECEEDING STRING  */
152  2                      IF CHR = 3 THEN CHR = 0FFH; /*  CONTROL-C FORCES   C E A S E   (CONTROL
                                        ; STATEMENT # 152
        0909 3A4943          LDA     CHR
        090C FE03            CPI     3H
        090E C21609          JNZ     @14
                                        ; STATEMENT # 153
        0911 214943          LXI     H,CHR
        0914 36FF            MVI     M,0FFH
                     @14:
                                                RETURNS TO ISIS)  */
154  2                      IF CHR = 4 THEN       /*  CONTROL-D (CRLF)   D I S P L A Y S
                                                                    MACRO-TABLE  */
                                        ; STATEMENT # 154
        0916 3A4943          LDA     CHR
        0919 FE04            CPI     4H
        091B C23F09          JNZ     @15
155  2                        DO;
156  3                        CALL EIGHTY(STAR); CALL CRLF;
                                        ; STATEMENT # 156
        091E 0E2A            MVI     C,2AH
        0920 CD1E07          CALL    EIGHTY
                                        ; STATEMENT # 157
        0923 CDD806          CALL    CRLF
158  3                        CALL REMARK(.('MACRO-TABLE IS:', EOS));
                                        ; STATEMENT # 158
        0926 01C000          LXI     B,$-866H
        0929 CDE306          CALL    REMARK
159  3                        CALL DISPLAY(TLAST, .TABLE);
                                        ; STATEMENT # 159
        092C 2A2000          LHLD    TLAST
        092F 44              MOV     B,H
        0930 4D              MOV     C,L
        0931 112200          LXI     D,TABLE
        0934 CD4207          CALL    DISPLAY
160  3                        CALL EIGHTY(STAR); CALL CRLF;
                                        ; STATEMENT # 160
        0937 0E2A            MVI     C,2AH
        0939 CD1E07          CALL    EIGHTY
                                        ; STATEMENT # 161
        093C CDD806          CALL    CRLF
162  3                      END;
                                        ; STATEMENT # 162
                     @15:
163  2                    RETURN CHR;
                                        ; STATEMENT # 163
        093F 3A4943          LDA     CHR
        0942 C9              RET
164  2                  END GETC;
                                        ; STATEMENT # 164


165  1          PUTBAK: PROCEDURE (CHR); /*  PUSH CHARACTER BACK ONTO INPUT  */
                                        ; STATEMENT # 165
                         ; PROC  PUTBAK
        0ACD 214E43          LXI     H,CHR
        0AD0 71              MOV     M,C
166  2                    DECLARE CHR BYTE;
167  2                    BP = BP - 1;
                                        ; STATEMENT # 167
        0AD1 2A1200          LHLD    BP
        0AD4 2B              DCX     H
        0AD5 221200          SHLD    BP
168  2                    IF BP = 0FFFFH THEN
                                        ; STATEMENT # 168
        0AD8 11FFFF          LXI     D,0FFFFH
        0ADB CD0000          CALL    @P0095
        0ADE B5              ORA     L
        0ADF C2EE0A          JNZ     @16
169  2                      DO;
170  3                        CALL REMARK(.('*** ERROR: TOO MANY CHARACTERS
                                                        PUSHED BACK',EOS));
                                        ; STATEMENT # 170
        0AE2 01D000          LXI     B,$-0A12H
        0AE5 CDE306          CALL    REMARK
171  3                        GOTO RECOVER;
                                        ; STATEMENT # 171
        0AE8 C34305          JMP     RECOVER
172  3                      END;
                                        ; STATEMENT # 172
        0AEB C3F90A          JMP     @17
                     @16:
173  2                    ELSE BUF(BP) = CHR;
                                        ; STATEMENT # 173
        0AEE 2A1200          LHLD    BP
        0AF1 012210          LXI     B,BUF
        0AF4 09              DAD     B
        0AF5 3A4E43          LDA     CHR
        0AF8 77              MOV     M,A
                     @17:
174  2                    RETURN;
                                        ; STATEMENT # 174
        0AF9 C9              RET
175  2                  END PUTBAK;
                                        ; STATEMENT # 175


176  1          PBSTR: PROCEDURE (IN$PTR);  /*  PUSH STRING BACK ONTO INPUT  */
                                        ; STATEMENT # 176
                         ; PROC  PBSTR
        0AFA 215043          LXI     H,INPTR+1H
        0AFD 70              MOV     M,B
        0AFE 2B              DCX     H
        0AFF 71              MOV     M,C
177  2                    DECLARE (I, IN$PTR) ADDRESS, IN BASED IN$PTR (1) BYTE;
```

```
178  2            I = LENGTH( IN) - 1;   /*  FORTRAN TO PLM ARRAY CONVERSION  */
                                          ; STATEMENT # 178
     0B00  2A4F43          LHLD    INPTR
     0B03  44              MOV     B,H
     0B04  4D              MOV     C,L
     0B05  CDB006          CALL    LENGTH
     0B08  2B              DCX     H
     0B09  225143          SHLD    I
179  2            DO WHILE I <> OFFFFH;
                                          ; STATEMENT # 179
                  @112:
     0B0C  01FFFF          LXI     B,0FFFFH
     0B0F  115143          LXI     D,I
     0B12  CD0000          CALL    @P0100
     0B15  B5              ORA     L
     0B16  CA2F0B          JZ      @113
180  3            CALL PUTBAK(IN(I));
                                          ; STATEMENT # 180
     0B19  2A5143          LHLD    I
     0B1C  EB              XCHG
     0B1D  2A4F43          LHLD    INPTR
     0B20  19              DAD     D
     0B21  4E              MOV     C,M
     0B22  CDCD0A          CALL    PUTBAK
181  3            I = I - 1;
                                          ; STATEMENT # 181
     0B25  2A5143          LHLD    I
     0B28  2B              DCX     H
     0B29  225143          SHLD    I
182  3            END /*  DO-WHILE  */;
                                          ; STATEMENT # 182
     0B2C  C30C0B          JMP     @112

                  @113:
183  2            RETURN;
                                          ; STATEMENT # 183
     0B2F  C9              RET
184  2            END PBSTR;
                                          ; STATEMENT # 184


185  1       INDEX:  PROCEDURE(STR$PTR, CHR) BYTE;  /*  FIND CHARACTER  CHR
                                          ; STATEMENT # 185
                  ; PROC  INDEX
     0B30  215543          LXI     H,CHR
     0B33  73              MOV     M,E
     0B34  2B              DCX     H
     0B35  70              MOV     M,B
     0B36  2B              DCX     H
     0B37  71              MOV     M,C
                                                  IN STRING STR  */
186  2            DECLARE STR$PTR ADDRESS, STR BASED STR$PTR (1) BYTE, (CHR, INDEX) BYTE;

187  2            INDEX = 1;
                                          ; STATEMENT # 187
     0B38  215643          LXI     H,INDEX
     0B3B  3601            MVI     M,1H
188  2            DO WHILE STR(INDEX) <> EOS;
                                          ; STATEMENT # 188
                  @114:
     0B3D  2A5643          LHLD    INDEX
     0B40  2600            MVI     H,0
     0B42  EB              XCHG
     0B43  2A5343          LHLD    STRPTR
     0B46  19              DAD     D
     0B47  7E              MOV     A,M
     0B48  FEFE            CPI     0FEH
     0B4A  CA690B          JZ      @115
189  3            IF STR(INDEX) = CHR THEN
                                          ; STATEMENT # 189
     0B4D  2A5643          LHLD    INDEX
     0B50  2600            MVI     H,0
     0B52  EB              XCHG
     0B53  2A5343          LHLD    STRPTR
     0B56  19              DAD     D
     0B57  3A5543          LDA     CHR
     0B5A  BE              CMP     M
     0B5B  C2620B          JNZ     @18
190  3              RETURN INDEX;
                                          ; STATEMENT # 190
     0B5E  3A5643          LDA     INDEX
     0B61  C9              RET
                  @18:
191  3            INDEX = INDEX + 1;
                                          ; STATEMENT # 191
     0B62  215643          LXI     H,INDEX
     0B65  34              INR     M
```

```
192  3            END /*  DO-WHILE  */;
                                          ; STATEMENT # 192
     0B66  C33D0B          JMP     @114
                  @115:
193  2            RETURN 0;
                                          ; STATEMENT # 193
     0B69  3E00            MVI     A,0H
     0B6B  C9              RET
194  2            END INDEX;
                                          ; STATEMENT # 194


195  1       TYPE  PROCEDURE(CHR) BYTE;  /*  DETERMINE TYPE OF CHARACTER  */
                                          ; STATEMENT # 195
                  ; PROC  TYPE
     0B6C  215743          LXI     H,CHR
     0B6F  71              MOV     M,C
196  2            DECLARE CHR BYTE;

197  2            IF (30H <= CHR) AND (CHR <= 39H) THEN
                                          ; STATEMENT # 197
     0B70  3A5743          LDA     CHR
     0B73  D630            SUI     30H
     0B75  9F              SBB     A
     0B76  2F              CMA
     0B77  F5              PUSH    PSW     ; 1
     0B78  3E39            MVI     A,39H
     0B7A  215743          LXI     H,CHR
     0B7D  96              SUB     M
     0B7E  9F              SBB     A
     0B7F  2F              CMA
     0B80  C1              POP     B       ; 1
     0B81  48              MOV     C,B
     0B82  A1              ANA     C
     0B83  1F              RAR
     0B84  D28A0B          JNC     @19
198  2              RETURN DIGIT;
                                          ; STATEMENT # 198
     0B87  3E00            MVI     A,0H
     0B89  C9              RET
                  @19:
                ELSE
199  2              IF ((61H <= CHR) AND (CHR <= 7AH) OR
                                          ; STATEMENT # 199
     0B8A  3A5743          LDA     CHR
     0B8D  D661            SUI     61H
     0B8F  9F              SBB     A
     0B90  2F              CMA
     0B91  F5              PUSH    PSW     ; 1
     0B92  3E7A            MVI     A,7AH
     0B94  215743          LXI     H,CHR
     0B97  96              SUB     M
     0B98  9F              SBB     A
     0B99  2F              CMA
     0B9A  C1              POP     B       ; 1
     0B9B  48              MOV     C,B
     0B9C  A1              ANA     C
     0B9D  F5              PUSH    PSW     ; 1
     0B9E  7E              MOV     A,M
     0B9F  D641            SUI     41H
     0BA1  9F              SBB     A
     0BA2  2F              CMA
     0BA3  F5              PUSH    PSW     ; 2
     0BA4  3E5A            MVI     A,5AH
     0BA6  96              SUB     M
     0BA7  9F              SBB     A
     0BA8  2F              CMA
     0BA9  C1              POP     B       ; 2
     0BAA  48              MOV     C,B
     0BAB  A1              ANA     C
     0BAC  C1              POP     B       ; 1
     0BAD  48              MOV     C,B
     0BAE  B1              ORA     C
     0BAF  1F              RAR
     0BB0  D2B60B          JNC     @21
                  (41H <= CHR) AND (CHR <= 5AH)) THEN
200  2                RETURN LETTER;
                                          ; STATEMENT # 200
     0BB3  3E41            MVI     A,41H
     0BB5  C9              RET
                  @21:
                ELSE
201  2                RETURN CHR;
                                          ; STATEMENT # 201
     0BB6  3A5743          LDA     CHR
     0BB9  C9              RET
                  @22:
                  @20:
```

```
202  2            END TYPE;
                                        ; STATEMENT # 202
      0BBA C9               RET

203  1       GETTOK: PROCEDURE(TOK$PTR, TOKSIZ) BYTE;  /*  GET ALPHANUMERIC STRING OR
                                        ; STATEMENT # 203
                     ; PROC  GETTOK
      0BBB 215A43           LXI     H,TOKSIZ
      0BBE 73               MOV     M,E
      0BBF 2B               DCX     H
      0BC0 70               MOV     M,B
      0BC1 2B               DCX     H
      0BC2 71               MOV     M,C
                                                        SINGLE NON-ALPHA  */
204  2              DECLARE TOK$PTR ADDRESS, TOKEN BASED TOK$PTR (1) BYTE,
                           (CHR, GETTOK, I, TOKSIZ) BYTE;

205  2              DO I = 0 TO TOKSIZ-1;  /*  FORTRAN TO PLM ARRAY CONVERSION  */
                                        ; STATEMENT # 205
      0BC3 215D43           LXI     H,I
      0BC6 3600             MVI     M,0H
                   @116:
      0BC8 3A5A43           LDA     TOKSIZ
      0BCB 3D               DCR     A
      0BCC 215D43           LXI     H,I
      0BCF BE               CMP     M
      0BD0 DA100C           JC      @117
206  3                CHR, TOKEN(I) = GETC;
                                        ; STATEMENT # 206
      0BD3 CD5E08           CALL    GETC
      0BD6 325B43           STA     CHR
      0BD9 2A5D43           LHLD    I
      0BDC 2600             MVI     H,0
      0BDE EB               XCHG
      0BDF 2A5843           LHLD    TOKPTR
      0BE2 19               DAD     D
      0BE3 77               MOV     M,A
207  3                GETTOK = TYPE(CHR);
                                        ; STATEMENT # 207
      0BE4 2A5B43           LHLD    CHR
      0BE7 4D               MOV     C,L
      0BE8 CD6C0B           CALL    TYPE
      0BEB 325C43           STA     GETTOK
208  3                IF (GETTOK <> LETTER) AND (GETTOK <> DIGIT) THEN
                                        ; STATEMENT # 208
      0BEE 3A5C43           LDA     GETTOK
      0BF1 D641             SUI     41H
      0BF3 C6FF             ADI     255
      0BF5 9F               SBB     A
      0BF6 F5               PUSH    PSW     ; 1
      0BF7 3A5C43           LDA     GETTOK
      0BFA D600             SUI     0H
      0BFC C6FF             ADI     255
      0BFE 9F               SBB     A
      0BFF C1               POP     B       ; 1
      0C00 48               MOV     C,B
      0C01 A1               ANA     C
      0C02 1F               RAR
      0C03 D2090C           JNC     @23
209  3                  GOTO EXIT;
                                        ; STATEMENT # 209
      0C06 C3100C           JMP     EXIT
                   @23:
210  3                END /*  DO I = 0 ...  */;
                                        ; STATEMENT # 210
      0C09 215D43           LXI     H,I
      0C0C 34               INR     M
      0C0D C2C80B           JNZ     @116
                   @117:
211  2           EXIT: IF I >= TOKSIZ THEN
                                        ; STATEMENT # 211
                   EXIT:
      0C10 215A43           LXI     H,TOKSIZ
      0C13 3A5D43           LDA     I
      0C16 BE               CMP     M
      0C17 DA230C           JC      @24
212  2                DO;
213  3                  CALL REMARK(.('*** ERROR: TOKEN TOO LONG '),EOS));
                                        ; STATEMENT # 213
      0C1A 01FC00           LXI     B,$-0B1EH
      0C1D CDE306           CALL    REMARK
214  3                  GOTO RECOVER;
                                        ; STATEMENT # 214
      0C20 C34305           JMP     RECOVER

215  3                END;
                                        ; STATEMENT # 215
                   @24:
216  2                IF I > 0 THEN    /*  SOME ALPHA WAS SEEN  */
                                        ; STATEMENT # 216
      0C23 3E00             MVI     A,0H
      0C25 215D43           LXI     H,I
      0C28 BE               CMP     M
      0C29 D2410C           JNC     @25
217  2                DO;
218  3                  CALL PUTBAK(TOKEN(I));
                                        ; STATEMENT # 218
      0C2C 2A5D43           LHLD    I
      0C2F 2600             MVI     H,0
      0C31 EB               XCHG
      0C32 2A5843           LHLD    TOKPTR
      0C35 19               DAD     D
      0C36 4E               MOV     C,M
      0C37 CDCD0A           CALL    PUTBAK
219  3                  I = I - 1;
                                        ; STATEMENT # 219
      0C3A 215D43           LXI     H,I
      0C3D 35               DCR     M
220  3                  GETTOK = ALPHA;
                                        ; STATEMENT # 220
      0C3E 2B               DCX     H
      0C3F 3641             MVI     M,41H
221  3                END /*  IF-DO  */;
                                        ; STATEMENT # 221
                   @25:
                   /*  ELSE SINGLE CHARACTER TOKEN  */
222  2                TOKEN(I+1) = EOS;
                                        ; STATEMENT # 222
      0C41 2A5D43           LHLD    I
      0C44 2600             MVI     H,0
      0C46 010100           LXI     B,TOKEN+1H
      0C49 09               DAD     B
      0C4A EB               XCHG
      0C4B 2A5843           LHLD    TOKPTR
      0C4E 19               DAD     D
      0C4F 36FE             MVI     M,0FEH
223  2              RETURN GETTOK;
                                        ; STATEMENT # 223
      0C51 3A5C43           LDA     GETTOK
      0C54 C9               RET
224  2              END GETTOK;
                                        ; STATEMENT # 224

225  1     SCOPY: PROCEDURE (FROM$PTR, I, TOO$PTR, J); /*  COPY STRING AT
                                                  FROM(I)  TO  TOO(J)  */
                                        ; STATEMENT # 225
                     ; PROC  SCOPY
      0C55 216543           LXI     H,J+1H
      0C58 72               MOV     M,D
      0C59 2B               DCX     H
      0C5A 73               MOV     M,E
      0C5B 2B               DCX     H
      0C5C 70               MOV     M,B
      0C5D 2B               DCX     H
      0C5E 71               MOV     M,C
      0C5F 2B               DCX     H
      0C60 D1               POP     D
      0C61 C1               POP     B
      0C62 70               MOV     M,B
      0C63 2B               DCX     H
      0C64 71               MOV     M,C
      0C65 2B               DCX     H
      0C66 C1               POP     B
      0C67 70               MOV     M,B
      0C68 2B               DCX     H
      0C69 71               MOV     M,C
      0C6A D5               PUSH    D
226  2              DECLARE (FROM$PTR, TOO$PTR, I, J, K1, K2) ADDRESS,
                           (FROM BASED FROM$PTR, TOO BASED TOO$PTR) (1) BYTE;

227  2              K2 = J;
                                        ; STATEMENT # 227
      0C6B 2A6443           LHLD    J
      0C6E 226843           SHLD    K2
228  2              K1 = I;
                                        ; STATEMENT # 228
      0C71 2A6043           LHLD    I
      0C74 226643           SHLD    K1
```

```
229  2            DO WHILE FROM(K1) <> EOS;
                                              ; STATEMENT # 229
                  @118
     0C77 2A6643          LHLD    K1
     0C7A EB              XCHG
     0C7B 2A5E43          LHLD    FROMPTR
     0C7E 19              DAD     D
     0C7F 7E              MOV     A,M
     0C80 FEFE            CPI     0FEH
     0C82 CAAA0C          JZ      @119
230  3            TOO(K2) = FROM(K1);
                                              ; STATEMENT # 230
     0C85 2A6643          LHLD    K1
     0C88 EB              XCHG
     0C89 2A5E43          LHLD    FROMPTR
     0C8C 19              DAD     D
     0C8D E5              PUSH    H         ; 1
     0C8E 2A6843          LHLD    K2
     0C91 EB              XCHG
     0C92 2A6243          LHLD    TOOPTR
     0C95 19              DAD     D
     0C96 C1              POP     B         ; 1
     0C97 0A              LDAX    B
     0C98 77              MOV     M,A
231  3            K2 = K2 + 1;
                                              ; STATEMENT # 231
     0C99 2A6843          LHLD    K2
     0C9C 23              INX     H
     0C9D 226843          SHLD    K2
232  3            K1 = K1 + 1;
                                              ; STATEMENT # 232
     0CA0 2A6643          LHLD    K1
     0CA3 23              INX     H
     0CA4 226643          SHLD    K1
233  3            END /*  DO-WHILE  */;
                                              ; STATEMENT # 233
     0CA7 C3770C          JMP     @118
                  @119:
234  2            TOO(K2) = EOS;
                                              ; STATEMENT # 234
     0CAA 2A6843          LHLD    K2
     0CAD EB              XCHG
     0CAE 2A6243          LHLD    TOOPTR
     0CB1 19              DAD     D
     0CB2 36FE            MVI     M,0FEH
235  2            RETURN;
                                              ; STATEMENT # 235
     0CB4 C9              RET
236  2            END SCOPY;
                                              ; STATEMENT # 236


237  1       INSTAL: PROCEDURE (NAME$PTR,DEFPTR);  /*  ADD NAME AND DEFINITION TO TABLE  */
                                              ; STATEMENT # 237
                  ; PROC  INSTAL
     0CB5 216D43          LXI     H,DEFPTR+1H
     0CB8 72              MOV     M,D
     0CB9 2B              DCX     H
     0CBA 73              MOV     M,E
     0CBB 2B              DCX     H
     0CBC 70              MOV     M,B
     0CBD 2B              DCX     H
     0CBE 71              MOV     M,C
238  2            DECLARE (NAME$PTR, DEFPTR) ADDRESS,
                         DEFN BASED DEFPTR (1) BYTE,
                         NAME BASED NAME$PTR (1) BYTE,
                         (DLEN, NLEN) ADDRESS;


239  2            NLEN = LENGTH( NAME) + 1;
                                              ; STATEMENT # 239
     0CBF 2A6A43          LHLD    NAMEPTR
     0CC2 44              MOV     B,H
     0CC3 4D              MOV     C,L
     0CC4 CDB006          CALL    LENGTH
     0CC7 23              INX     H
     0CC8 227043          SHLD    NLEN
240  2            DLEN = LENGTH( DEFN) + 1;
                                              ; STATEMENT # 240
     0CCB 2A6C43          LHLD    DEFPTR
     0CCE 44              MOV     B,H
     0CCF 4D              MOV     C,L
     0CD0 CDB006          CALL    LENGTH
     0CD3 23              INX     H
     0CD4 226E43          SHLD    DLEN

241  2            IF PLAST <> 0FFFFH THEN   /*      THEN PLAST, TLAST := 0  */
                                              ; STATEMENT # 241
     0CD7 01FFFF          LXI     B,0FFFFH
     0CDA 111C00          LXI     D,PLAST
     0CDD CD0000          CALL    @P0100
     0CE0 B5              ORA     L
     0CE1 CA150D          JZ      @26
242  2                 IF (TLAST + NLEN + DLEN > MAXTBL) OR (PLAST >= MAXPTR) THEN
                                              ; STATEMENT # 242
     0CE4 2A7043          LHLD    NLEN
     0CE7 EB              XCHG
     0CE8 2A2000          LHLD    TLAST
     0CEB 19              DAD     D
     0CEC E5              PUSH    H         ; 1
     0CED 2A6E43          LHLD    DLEN
     0CF0 C1              POP     B         ; 1
     0CF1 09              DAD     B
     0CF2 110010          LXI     D,1000H
     0CF5 CD0000          CALL    @P0095
     0CF8 9F              SBB     A
     0CF9 010001          LXI     B,100H
     0CFC 111C00          LXI     D,PLAST
     0CFF F5              PUSH    PSW       ; 1
     0D00 CD0000          CALL    @P0100
     0D03 9F              SBB     A
     0D04 2F              CMA
     0D05 C1              POP     B         ; 1
     0D06 48              MOV     C,B
     0D07 B1              ORA     C
     0D08 1F              RAR
     0D09 D2150D          JNC     @27
243  2                 DO;
244  3                   CALL REMARK( ( *** ERROR  TOO MANY DEFINITIONS ':EOS:));
                                              ; STATEMENT # 244
     0D0C 011701          LXI     B,$-0BF5H
     0D0F CDE306          CALL    REMARK
245  3                   GOTO RECOVER
                                              ; STATEMENT # 245
     0D12 C34305          JMP     RECOVER
246  3                 END;
                                              ; STATEMENT # 246
                  @27:
                  @26:
247  2            PLAST = PLAST + 1;
                                              ; STATEMENT # 247
     0D15 2A1C00          LHLD    PLAST
     0D18 23              INX     H
     0D19 221C00          SHLD    PLAST
248  2            NAMEX(INTBL(PLAST)) = TLAST + 1;
                                              ; STATEMENT # 248
     0D1C 2A2000          LHLD    TLAST
     0D1F 23              INX     H
     0D20 E5              PUSH    H         ; 1
     0D21 2A1C00          LHLD    PLAST
     0D24 012330          LXI     B,NAMEX(INTBL
     0D27 29              DAD     H
     0D28 09              DAD     B
     0D29 C1              POP     B         ; 1
     0D2A 71              MOV     M,C
     0D2B 23              INX     H
     0D2C 70              MOV     M,B
249  2            CALL SCOPY( NAME, 0, .TABLE, TLAST + 1);
                                              ; STATEMENT # 249
     0D2D 2A6A43          LHLD    NAMEPTR
     0D30 E5              PUSH    H         ; 1
     0D31 010000          LXI     B,0H
     0D34 C5              PUSH    B         ; 2
     0D35 2A2000          LHLD    TLAST
     0D38 23              INX     H
     0D39 EB              XCHG
     0D3A 012200          LXI     B,.TABLE
     0D3D CD550C          CALL    SCOPY
250  2            CALL SCOPY( DEFN, 0, .TABLE, TLAST + NLEN + 1);
                                              ; STATEMENT # 250
     0D40 2A6C43          LHLD    DEFPTR
     0D43 E5              PUSH    H         ; 1
     0D44 010000          LXI     B,0H
     0D47 C5              PUSH    B         ; 2
     0D48 2A7043          LHLD    NLEN
     0D4B EB              XCHG
     0D4C 2A2000          LHLD    TLAST
     0D4F 19              DAD     D
     0D50 23              INX     H
     0D51 EB              XCHG
     0D52 012200          LXI     B,.TABLE
     0D55 CD550C          CALL    SCOPY
```

```
251  2                TLAST = TLAST + NLEN + DLEN;
                                                ; STATEMENT # 251
          0D58  2A7043          LHLD    NLEN
          0D5B  EB              XCHG
          0D5C  2A2000          LHLD    TLAST
          0D5F  19              DAD     D
          0D60  E5              PUSH    H       ; 1
          0D61  2A6E43          LHLD    DLEN
          0D64  C1              POP     B       ; 1
          0D65  09              DAD     B
          0D66  222000          SHLD    TLAST
252  2              RETURN;
                                                ; STATEMENT # 252
          0D69  C9              RET
253  2              END INSTAL;
                                                ; STATEMENT # 253


254  1      EQUAL: PROCEDURE(STR1$PTR, STR2$PTR) BYTE;    /*  COMPARES
                                                        TWO STRINGS  */
                                                ; STATEMENT # 254
                 ; PROC  EQUAL
          0D6A  217543          LXI     H,STR2PTR+1H
          0D6D  72              MOV     M,D
          0D6E  2B              DCX     H
          0D6F  73              MOV     M,E
          0D70  2B              DCX     H
          0D71  70              MOV     M,B
          0D72  2B              DCX     H
          0D73  71              MOV     M,C
255  2            DECLARE (STR1$PTR, STR2$PTR) ADDRESS, I BYTE,
                        (STR1 BASED STR1$PTR, STR2 BASED STR2$PTR) (1) BYTE;
256  2            I = 0;
                                                ; STATEMENT # 256
          0D74  217643          LXI     H,I
          0D77  3600            MVI     M,0H
257  2            DO WHILE STR1(I) = STR2(I);
                                                ; STATEMENT # 257
               @120:
          0D79  2A7643          LHLD    I
          0D7C  2600            MVI     H,0
          0D7E  EB              XCHG
          0D7F  2A7243          LHLD    STR1PTR
          0D82  19              DAD     D
          0D83  E5              PUSH    H       ; 1
          0D84  2A7643          LHLD    I
          0D87  2600            MVI     H,0
          0D89  EB              XCHG
          0D8A  2A7443          LHLD    STR2PTR
          0D8D  19              DAD     D
          0D8E  C1              POP     B       ; 1
          0D8F  0A              LDAX    B
          0D90  BE              CMP     M
          0D91  C2AE0D          JNZ     @121
258  3              IF STR1(I) = EOS THEN
                                                ; STATEMENT # 258
          0D94  2A7643          LHLD    I
          0D97  2600            MVI     H,0
          0D99  EB              XCHG
          0D9A  2A7243          LHLD    STR1PTR
          0D9D  19              DAD     D
          0D9E  7E              MOV     A,M
          0D9F  FEFE            CPI     0FEH
          0DA1  C2A70D          JNZ     @28
259  3                RETURN YES;
                                                ; STATEMENT # 259
          0DA4  3E01            MVI     A,1H
          0DA6  C9              RET
               @28:
260  3              I = I + 1;
                                                ; STATEMENT # 260
          0DA7  217643          LXI     H,I
          0DAA  34              INR     M
261  3            END /*  DO-WHILE  */;
                                                ; STATEMENT # 261
          0DAB  C3790D          JMP     @120
               @121:
262  2            RETURN NO;
                                                ; STATEMENT # 262
          0DAE  3E00            MVI     A,0H
          0DB0  C9              RET
263  2            END EQUAL;
                                                ; STATEMENT # 263


264  1      SEEKNBR: PROCEDURE (TABDEX) ADDRESS;
                                                ; STATEMENT # 264
                 PROC  SEEKNBR
          0DB1  217843          LXI     H,TABDEX+1H
          0DB4  70              MOV     M,B
          0DB5  2B              DCX     H
          0DB6  71              MOV     M,C
265  2            DECLARE (I, TABDEX) ADDRESS;

266  2            DO I = 0 TO PLAST;
                                                ; STATEMENT # 266
          0DB7  210000          LXI     H,0H
          0DBA  227943          SHLD    I
               @122:
          0DBD  111C00          LXI     D,PLAST
          0DC0  017943          LXI     B,I
          0DC3  CD0000          CALL    @P0098
          0DC6  DAECOD          JC      @123
267  3              IF NAMDEXINTBL(I) = TABDEX THEN
                                                ; STATEMENT # 267
          0DC9  2A7943          LHLD    I
          0DCC  012330          LXI     B,NAMDEXINTBL
          0DCF  29              DAD     H
          0DD0  09              DAD     B
          0DD1  117743          LXI     D,TABDEX
          0DD4  CD0000          CALL    @P0099
          0DD7  B5              ORA     L
          0DD8  C2DF0D          JNZ     @29
268  3                RETURN I;
                                                ; STATEMENT # 268
          0DDB  2A7943          LHLD    I
          0DDE  C9              RET
               @29:
269  3            END /* DO-LOOP */;
                                                ; STATEMENT # 269
          0DDF  110100          LXI     D,1H
          0DE2  2A7943          LHLD    I
          0DE5  19              DAD     D
          0DE6  227943          SHLD    I
          0DE9  D2BD0D          JNC     @122
               @123:
270  2            CALL REMARK( ('*** ERROR  TABDEX NOT FOUND IN NAMDEXINTBL',EOS));
                                                ; STATEMENT # 270
          0DEC  013801          LXI     B,$-0CB4H
          0DEF  CDE306          CALL    REMARK
271  2            END SEEKNBR;
                                                ; STATEMENT # 271
          0DF2  C9              RET


272  1      SEEKSTR: PROCEDURE(STR$PTR) ADDRESS;   /*  FIND STRING IN TABLE, RETURN INDEX  */
                                                ; STATEMENT # 272
                 ; PROC  SEEKSTR
          0DF3  217C43          LXI     H,STRPTR+1H
          0DF6  70              MOV     M,B
          0DF7  2B              DCX     H
          0DF8  71              MOV     M,C
273  2            DECLARE (I, J, K, STR$PTR) ADDRESS,
                        STRING BASED STR$PTR (1) BYTE;

274  2            I = PLAST;
                                                ; STATEMENT # 274
          0DF9  2A1C00          LHLD    PLAST
          0DFC  227D43          SHLD    I
275  2            DO WHILE I <> 0FFFFH;
                                                ; STATEMENT # 275
               @124:
          0DFF  01FFFF          LXI     B,0FFFFH
          0E02  117D43          LXI     D,I
          0E05  CD0000          CALL    @P0100
          0E08  B5              ORA     L
          0E09  CA830E          JZ      @125
276  3              J = NAMDEXINTBL(I);
                                                ; STATEMENT # 276
          0E0C  2A7D43          LHLD    I
          0E0F  012330          LXI     B,NAMDEXINTBL
          0E12  29              DAD     H
          0E13  09              DAD     B
          0E14  5E              MOV     E,M
          0E15  23              INX     H
          0E16  56              MOV     D,M
          0E17  EB              XCHG
          0E18  227F43          SHLD    J
277  3              K = 0;
                                                ; STATEMENT # 277
          0E1B  210000          LXI     H,0H
          0E1E  228143          SHLD    K
```

```
278  3                 DO WHILE (STRING(K) = TABLE(J)) AND (STRING(K) <> EOS);
                                                    ; STATEMENT # 278
                   @126:
     0E21  2A8143          LHLD    K
     0E24  EB              XCHG
     0E25  2A7B43          LHLD    STRPTR
     0E28  19              DAD     D
     0E29  E5              PUSH    H       ; 1
     0E2A  2A7F43          LHLD    J
     0E2D  012200          LXI     B,TABLE
     0E30  09              DAD     B
     0E31  D1              POP     D       ; 1
     0E32  1A              LDAX    D
     0E33  96              SUB     M
     0E34  D601            SUI     1
     0E36  9F              SBB     A
     0E37  F5              PUSH    PSW     ; 1
     0E38  1A              LDAX    D
     0E39  D6FE            SUI     0FEH
     0E3B  C6FF            ADI     255
     0E3D  9F              SBB     A
     0E3E  C1              POP     B       ; 1
     0E3F  48              MOV     C,B
     0E40  A1              ANA     C
     0E41  1F              RAR
     0E42  D2560E          JNC     @127
279  4                  J = J + 1;
                                                    ; STATEMENT # 279
     0E45  2A7F43          LHLD    J
     0E48  23              INX     H
     0E49  227F43          SHLD    J
280  4                  K = K + 1;
                                                    ; STATEMENT # 280
     0E4C  2A8143          LHLD    K
     0E4F  23              INX     H
     0E50  228143          SHLD    K
281  4                END /*  DO-WHILE  */;
                                                    ; STATEMENT # 281
     0E53  C3210E          JMP     @126
                   @127:
282  3                IF STRING(K) = TABLE(J) THEN    /*  FOUND IT  */
                                                    ; STATEMENT # 282
     0E56  2A8143          LHLD    K
     0E59  EB              XCHG
     0E5A  2A7B43          LHLD    STRPTR
     0E5D  19              DAD     D
     0E5E  E5              PUSH    H       ; 1
     0E5F  2A7F43          LHLD    J
     0E62  012200          LXI     B,TABLE
     0E65  09              DAD     B
     0E66  D1              POP     D       ; 1
     0E67  1A              LDAX    D
     0E68  BE              CMP     M
     0E69  C2790E          JNZ     @30
283  3                  RETURN NAMDEXINTBL(I);
                                                    ; STATEMENT # 283
     0E6C  2A7D43          LHLD    I
     0E6F  012330          LXI     B,NAMDEXINTBL
     0E72  29              DAD     H
     0E73  09              DAD     B
     0E74  5E              MOV     E,M
     0E75  23              INX     H
     0E76  56              MOV     D,M
     0E77  EB              XCHG
     0E78  C9              RET
                   @30:
284  3                  I = I - 1;
                                                    ; STATEMENT # 284
     0E79  2A7D43          LHLD    I
     0E7C  2B              DCX     H
     0E7D  227D43          SHLD    I
285  3                END /*  DO-WHILE  */;
                                                    ; STATEMENT # 285
     0E80  C3FF0D          JMP     @124
                   @125:
286  2                RETURN 0FFFFH;
                                                    ; STATEMENT # 286
     0E83  21FFFF          LXI     H,0FFFFH
     0E86  C9              RET
287  2                END SEEKSTR;
                                                    ; STATEMENT # 287

288  1           DELETE: PROCEDURE(NAM$PTR);    /*  REMOVES LAST DEF'N OF NAME FROM TABLE  */
                                                    ; STATEMENT # 288
                      ; PROC  DELETE
     0E87  218443          LXI     H,NAMPTR+1H
     0E8A  70              MOV     M,B
     0E8B  2B              DCX     H
     0E8C  71              MOV     M,C
289  2                DECLARE (DEXDEX, I, J, K) NAM$PTR, TABDEX) ADDRESS
                           NAME BASED NAM$PTR  '1' BYTE;

290  2                TABDEX = SEEKSTR( NAME);      /*  TABDEX POINTS INTO TABLE  */
                                                    ; STATEMENT # 290
     0E8D  2A8343          LHLD    NAMPTR
     0E90  44              MOV     B,H
     0E91  4D              MOV     C,L
     0E92  CDF30D          CALL    SEEKSTR
     0E95  228D43          SHLD    TABDEX
291  2                IF TABDEX <> 0FFFFH THEN
                                                    ; STATEMENT # 291
     0E98  01FFFF          LXI     B,0FFFFH
     0E9B  118D43          LXI     D,TABDEX
     0E9E  CD0000          CALL    @P0100
     0EA1  B5              ORA     L
     0EA2  CA310F          JZ      @31
292  2                  DO;
293  3                    DEXDEX = SEEKNBR(TABDEX);    /*  FIND TABDEX IN NAMDEXINTBL  */
                                                    ; STATEMENT # 293
     0EA5  2A8D43          LHLD    TABDEX
     0EA8  44              MOV     B,H
     0EA9  4D              MOV     C,L
     0EAA  CDB10D          CALL    SEEKNBR
     0EAD  228543          SHLD    DEXDEX
294  3                    IF DEXDEX < PLAST THEN   /*  CRUNCH TABLE TO OVERLAY THE NAMED MACRO
                                                    ; STATEMENT # 294
     0EB0  011C00          LXI     B,PLAST
     0EB3  118543          LXI     D,DEXDEX
     0EB6  CD0000          CALL    @P0098
     0EB9  D2230F          JNC     @32
295  3                      DO;
296  4                        J = TABDEX;    /*  NB. TABDEX IS NAMDEXINTBL(DEXDEX)  */
                                                    ; STATEMENT # 296
     0EBC  2A8D43          LHLD    TABDEX
     0EBF  228943          SHLD    J
297  4                        K = NAMDEXINTBL(DEXDEX+1);    /*  NEXT MACRO IN TABLE  */
                                                    ; STATEMENT # 297
     0EC2  2A8543          LHLD    DEXDEX
     0EC5  012530          LXI     B,NAMDEXINTBL+2H
     0EC8  29              DAD     H
     0EC9  09              DAD     B
     0ECA  5E              MOV     E,M
     0ECB  23              INX     H
     0ECC  56              MOV     D,M
     0ECD  EB              XCHG
     0ECE  228B43          SHLD    K
298  4                        DO I = 0 TO (TLAST - K);      /*  THE BLOCK MOVE  */
                                                    ; STATEMENT # 298
     0ED1  210000          LXI     H,0H
     0ED4  228743          SHLD    I
                   @128:
     0ED7  018B43          LXI     B,K
     0EDA  112000          LXI     D,TLAST
     0EDD  CD0000          CALL    @P0098
     0EE0  EB              XCHG
     0EE1  218743          LXI     H,I
     0EE4  CD0000          CALL    @P0104
     0EE7  DA0C0F          JC      @129
299  5                      TABLE(J+I) = TABLE(K+I);
                                                    ; STATEMENT # 299
     0EEA  2A8743          LHLD    I
     0EED  EB              XCHG
     0EEE  2A8B43          LHLD    K
     0EF1  19              DAD     D
     0EF2  012200          LXI     B,TABLE
     0EF5  09              DAD     B
     0EF6  E5              PUSH    H       ; 1
     0EF7  2A8943          LHLD    J
     0EFA  19              DAD     D
     0EFB  09              DAD     B
     0EFC  D1              POP     D       ; 1
     0EFD  1A              LDAX    D
     0EFE  77              MOV     M,A
```

```
300  5                    END;  /*  DO-LOOP  */
                                             ; STATEMENT # 300
        0EFF  110100          LXI     D,1H
        0F02  2A8743          LHLD    I
        0F05  19              DAD     D
        0F06  228743          SHLD    I
        0F09  D2D70E          JNC     @128
                    @129:
301  4                    TLAST = TLAST - (K - J);
                                             ; STATEMENT # 301
        0F0C  018943          LXI     B,J
        0F0F  118B43          LXI     D,K
        0F12  CD0000          CALL    @P0098
        0F15  112000          LXI     D,TLAST
        0F18  CD0000          CALL    @P0102
        0F1B  EB              XCHG
        0F1C  2B              DCX     H
        0F1D  73              MOV     M,E
        0F1E  23              INX     H
        0F1F  72              MOV     M,D
302  4                   END;   /* IF-DO  */
                                             ; STATEMENT # 302
        0F20  C32A0F          JMP     @33
                    @32:
                      ELSE             /*  NAME IS THE LAST MACRO IN TABLE  */
303  3                      TLAST = TABDEX - 1;
                                             ; STATEMENT # 303
        0F23  2A8D43          LHLD    TABDEX
        0F26  2B              DCX     H
        0F27  222000          SHLD    TLAST
                    @33:
304  3                     PLAST = PLAST - 1;
                                             ; STATEMENT # 304
        0F2A  2A1C00          LHLD    PLAST
        0F2D  2B              DCX     H
        0F2E  221C00          SHLD    PLAST
305  3                  END;  /*  IF-DO  */
                                             ; STATEMENT # 305
                    @31:
306  2                  RETURN;
                                             ; STATEMENT # 306
        0F31  C9              RET
307  2                  END DELETE;
                                             ; STATEMENT # 307


308  1          PUTCHR: PROCEDURE(CHR);   /*  PUT SINGLE CHAR ON OUTPUT OR INTO
                                                        EVALUATION STACK  */
                                             ; STATEMENT # 308
                    ; PROC  PUTCHR
        0F32  218F43          LXI     H,CHR
        0F35  71              MOV     M,C
309  2                  DECLARE CHR BYTE;

310  2                  IF CP = 0FFFFH THEN
                                             ; STATEMENT # 310
        0F36  01FFFF          LXI     B,0FFFFH
        0F39  111400          LXI     D,CP
        0F3C  CD0000          CALL    @P0100
        0F3F  B5              ORA     L
        0F40  C24D0F          JNZ     @34
311  2                   CALL PUTC(CHR);
                                             ; STATEMENT # 311
        0F43  2A8F43          LHLD    CHR
        0F46  4D              MOV     C,L
        0F47  CD7806          CALL    PUTC
        0F4A  C3740F          JMP     @35
                    @34:
                   ELSE
312  2                   DO;
313  3                     IF EP > EVALSIZE THEN
                                             ; STATEMENT # 313
        0F4D  110010          LXI     D,1000H
        0F50  211600          LXI     H,EP
        0F53  CD0000          CALL    @P0104
        0F56  D2620F          JNC     @36
314  3                    DO;
315  4                      CALL REMARK(.('*** ERROR: EVALUATION STACK OVERFLOW ',EOS));
                                             ; STATEMENT # 315
        0F59  016301          LXI     B,$-0DF6H
        0F5C  CDE306          CALL    REMARK
316  4                     GOTO RECOVER;
                                             ; STATEMENT # 316
        0F5F  C34305          JMP     RECOVER
317  4                    END;
                                             ; STATEMENT # 317
                    @36:

318  3                   EVALST(EP) = CHR;
                                             ; STATEMENT # 318
        0F62  2A1600          LHLD    EP
        0F65  012320          LXI     B,EVALST
        0F68  09              DAD     B
        0F69  3A8F43          LDA     CHR
        0F6C  77              MOV     M,A
319  3                    EP = EP + 1;
                                             ; STATEMENT # 319
        0F6D  2A1600          LHLD    EP
        0F70  23              INX     H
        0F71  221600          SHLD    EP
320  3                   END /*  ELSE-DO  */;
                                             ; STATEMENT # 320
                    @35:
321  2                   END PUTCHR;
                                             ; STATEMENT # 321
        0F74  C9              RET


322  1       PUTTOK: PROCEDURE (STR$PTR);  /*  PUT A TOKEN EITHER ON OUTPUT OR INTO
                                                                    EVALUATION STACK  */
                                             ; STATEMENT # 322
                    ; PROC  PUTTOK
        0F75  219143          LXI     H,STRPTR+1H
        0F78  70              MOV     M,B
        0F79  2B              DCX     H
        0F7A  71              MOV     M,C
323  2                  DECLARE STR$PTR ADDRESS, STR BASED STR$PTR (1) BYTE, I BYTE;

324  2                  I = 0;
                                             ; STATEMENT # 324
        0F7B  219243          LXI     H,I
        0F7E  3600            MVI     M,0H
325  2                  DO WHILE STR(I) <> EOS;
                                             ; STATEMENT # 325
                    @130:
        0F80  2A9243          LHLD    I
        0F83  2600            MVI     H,0
        0F85  EB              XCHG
        0F86  2A9043          LHLD    STRPTR
        0F89  19              DAD     D
        0F8A  7E              MOV     A,M
        0F8B  FEFE            CPI     0FEH
        0F8D  CAE60F          JZ      @131
326  3                   IF CP = 0FFFFH THEN
                                             ; STATEMENT # 326
        0F90  01FFFF          LXI     B,0FFFFH
        0F93  111400          LXI     D,CP
        0F96  CD0000          CALL    @P0100
        0F99  B5              ORA     L
        0F9A  C2AE0F          JNZ     @37
327  3                    CALL PUTC(STR(I));
                                             ; STATEMENT # 327
        0F9D  2A9243          LHLD    I
        0FA0  2600            MVI     H,0
        0FA2  EB              XCHG
        0FA3  2A9043          LHLD    STRPTR
        0FA6  19              DAD     D
        0FA7  4E              MOV     C,M
        0FA8  CD7806          CALL    PUTC
        0FAB  C3DF0F          JMP     @38
                    @37:
                    ELSE
328  3                    DO;
329  4                      IF EP > EVALSIZE THEN
                                             ; STATEMENT # 329
        0FAE  110010          LXI     D,1000H
        0FB1  211600          LXI     H,EP
        0FB4  CD0000          CALL    @P0104
        0FB7  D2C30F          JNC     @39
330  4                     DO;
331  5                       CALL REMARK(.('*** ERROR: EVALUATION STACK
                                                        OVERFLOW', EOS));
                                             ; STATEMENT # 331
        0FBA  018901          LXI     B,$-0E31H
        0FBD  CDE306          CALL    REMARK
332  5                      GOTO RECOVER;
                                             ; STATEMENT # 332
        0FC0  C34305          JMP     RECOVER
333  5                     END;
                                             ; STATEMENT # 333
                    @39:
```

```
334  4                    EVALST(EP) = STR(I);
                                                ; STATEMENT # 334
     0FC3  2A9243           LHLD    I
     0FC6  2600             MVI     H,0
     0FC8  EB               XCHG
     0FC9  2A9043           LHLD    STRPTR
     0FCC  19               DAD     D
     0FCD  E5               PUSH    H       ; 1
     0FCE  2A1600           LHLD    EP
     0FD1  012320           LXI     B,EVALST
     0FD4  09               DAD     B
     0FD5  D1               POP     D       ; 1
     0FD6  1A               LDAX    D
     0FD7  77               MOV     M,A
335  4                  EP = EP + 1;
                                                ; STATEMENT # 335
     0FD8  2A1600           LHLD    EP
     0FDB  23               INX     H
     0FDC  221600           SHLD    EP
336  4                END /*  ELSE-DO  */;
                                                ; STATEMENT # 336
                       @38:
337  3                 I = I + 1;
                                                ; STATEMENT # 337
     0FDF  219243           LXI     H,I
     0FE2  34               INR     M
338  3              END /*  DO-WHILE  */;
                                                ; STATEMENT # 338
     0FE3  C3800F           JMP     @130
                       @131:
339  2              RETURN;
                                                ; STATEMENT # 339
     0FE6  C9               RET
340  2              END PUTTOK;
                                                ; STATEMENT # 340


341  1       PUSH: PROCEDURE (EP, ARGSTK$PTR, AP) ADDRESS;  /*  PUSH EP ONTO ARGSTK,
                                                        RETURN NEW POINTER AP  */
                                                ; STATEMENT # 341
                        ; PROC  PUSH
     0FE7  219843           LXI     H,AP+1H
     0FEA  72               MOV     M,D
     0FEB  2B               DCX     H
     0FEC  73               MOV     M,E
     0FED  2B               DCX     H
     0FEE  70               MOV     M,B
     0FEF  2B               DCX     H
     0FF0  71               MOV     M,C
     0FF1  2B               DCX     H
     0FF2  D1               POP     D
     0FF3  C1               POP     B
     0FF4  70               MOV     M,B
     0FF5  2B               DCX     H
     0FF6  71               MOV     M,C
     0FF7  D5               PUSH    D
342  2              DECLARE (AP, EP, ARGSTK$PTR) ADDRESS, ARGSTK BASED
                    ARGSTK$PTR (1) ADDRESS;

343  2              IF AP > ARGSIZE THEN
                                                ; STATEMENT # 343
     0FF8  110004           LXI     D,400H
     0FFB  219743           LXI     H,AP
     0FFE  CD0000           CALL    @P0104
     1001  D20D10           JNC     @40
344  2                   DO;
345  3                     CALL REMARK( ( *** ERROR  ARG STACK OVERFLOW ',EOS));
                                                ; STATEMENT # 345
     1004  01AE01           LXI     B,$-0E56H
     1007  CDE306           CALL    REMARK
346  3                     GOTO RECOVER
                                                ; STATEMENT # 346
     100A  C34305           JMP     RECOVER
347  3                   END;
                                                ; STATEMENT # 347
                       @40:
348  2                ARGSTK(AP) = EP;
                                                ; STATEMENT # 348
     100D  2A9743           LHLD    AP
     1010  29               DAD     H
     1011  EB               XCHG
     1012  2A9543           LHLD    ARGSTK$PTR
     1015  19               DAD     D
     1016  E5               PUSH    H       ; 1
     1017  2A9343           LHLD    EP
     101A  EB               XCHG
     101B  E1               POP     H       ; 1
     101C  73               MOV     M,E
     101D  23               INX     H
     101E  72               MOV     M,D
349  2              RETURN AP+1;
                                                ; STATEMENT # 349
     101F  2A9743           LHLD    AP
     1022  23               INX     H
     1023  C9               RET
350  2              END PUSH;
                                                ; STATEMENT # 350


351  1       LOOKUP: PROCEDURE (NAME$PTR, DEFN$PTR) BYTE;  /*  LOCATE NAME, EXTRACT
                                                        DEFINITION FROM TABLE  */
                                                ; STATEMENT # 351
                        ; PROC  LOOKUP
     1024  219C43           LXI     H,DEFNPTR+1H
     1027  72               MOV     M,D
     1028  2B               DCX     H
     1029  73               MOV     M,E
     102A  2B               DCX     H
     102B  70               MOV     M,B
     102C  2B               DCX     H
     102D  71               MOV     M,C
352  2              DECLARE (I, J, K, NAME$PTR, DEFN$PTR) ADDRESS,
                           NAME BASED NAME$PTR (1) BYTE,
                           DEFN BASED DEFN$PTR (1) BYTE;

353  2              I = PLAST;
                                                ; STATEMENT # 353
     102E  2A1C00           LHLD    PLAST
     1031  229D43           SHLD    I
354  2              DO WHILE I <> 0FFFFH;
                                                ; STATEMENT # 354
                       @132:
     1034  01FFFF           LXI     B,0FFFFH
     1037  119D43           LXI     D,I
     103A  CD0000           CALL    @P0100
     103D  B5               ORA     L
     103E  CAC210           JZ      @133
355  3                   J = NAMDEXINTBL(I);
                                                ; STATEMENT # 355
     1041  2A9D43           LHLD    I
     1044  012330           LXI     B,NAMDEXINTBL
     1047  29               DAD     H
     1048  09               DAD     B
     1049  5E               MOV     E,M
     104A  23               INX     H
     104B  56               MOV     D,M
     104C  EB               XCHG
     104D  229F43           SHLD    J
356  3                   K = 0;
                                                ; STATEMENT # 356
     1050  210000           LXI     H,0H
     1053  22A143           SHLD    K
357  3                   DO WHILE (NAME(K) = TABLE(J)) AND (NAME(K) <> EOS);
                                                ; STATEMENT # 357
                       @134:
     1056  2AA143           LHLD    K
     1059  EB               XCHG
     105A  2A9943           LHLD    NAMEPTR
     105D  19               DAD     D
     105E  E5               PUSH    H       ; 1
     105F  2A9F43           LHLD    J
     1062  012200           LXI     B,TABLE
     1065  09               DAD     B
     1066  D1               POP     D       ; 1
     1067  1A               LDAX    D
     1068  96               SUB     M
     1069  D601             SUI     1
     106B  9F               SBB     A
     106C  F5               PUSH    PSW     ; 1
     106D  1A               LDAX    D
     106E  D6FE             SUI     0FEH
     1070  C6FF             ADI     255
     1072  9F               SBB     A
     1073  C1               POP     B       ; 1
     1074  48               MOV     C,B
     1075  A1               ANA     C
     1076  1F               RAR
     1077  D28B10           JNC     @135
358  4                        J = J + 1;
                                                ; STATEMENT # 358
     107A  2A9F43           LHLD    J
     107D  23               INX     H
     107E  229F43           SHLD    J
```

```
359  4                  K = K + 1;
                                                ; STATEMENT # 359
          1081  2AA143          LHLD    K
          1084  23              INX     H
          1085  22A143          SHLD    K
360  4               END /* DO-WHILE */;
                                                ; STATEMENT # 360
          1088  C35610          JMP     @134
                          @135:
361  3               IF NAME(K) = TABLE(J) THEN /* GOT ONE */
                                                ; STATEMENT # 361
          108B  2AA143          LHLD    K
          108E  EB              XCHG
          108F  2A9943          LHLD    NAMEPTR
          1092  19              DAD     D
          1093  E5              PUSH    H       ; 1
          1094  2A9F43          LHLD    J
          1097  012200          LXI     B,TABLE
          109A  09              DAD     B
          109B  D1              POP     D       ; 1
          109C  1A              LDAX    D
          109D  BE              CMP     M
          109E  C2B810          JNZ     @41
362  3                 DO;
363  4                   CALL SCOPY(.TABLE, J+1, .DEFN, 0);
                                                ; STATEMENT # 363
          10A1  012200          LXI     B,TABLE
          10A4  C5              PUSH    B       ; 1
          10A5  2A9F43          LHLD    J
          10A8  23              INX     H
          10A9  E5              PUSH    H       ; 2
          10AA  2A9B43          LHLD    DEFNPTR
          10AD  44              MOV     B,H
          10AE  4D              MOV     C,L
          10AF  110000          LXI     D,0H
          10B2  CD550C          CALL    SCOPY
364  4                  RETURN YES;
                                                ; STATEMENT # 364
          10B5  3E01            MVI     A,1H
          10B7  C9              RET
365  4                 END /* IF-DO */;
                                                ; STATEMENT # 365
                          @41:
366  3                  I = I - 1;
                                                ; STATEMENT # 366
          10B8  2A9D43          LHLD    I
          10BB  2B              DCX     H
          10BC  229D43          SHLD    I
367  3               END /* DO-WHILE */;
                                                ; STATEMENT # 367
          10BF  C33410          JMP     @132
                          @133:
368  2               RETURN NO;
                                                ; STATEMENT # 368
          10C2  3E00            MVI     A,0H
          10C4  C9              RET
369  2               END LOOKUP;
                                                ; STATEMENT # 369



370  1       EVAL: PROCEDURE (ARGSTKPTR, I, J);  /* EXPAND ARGS I THRU J: EVALUATE
                                                    BUILTIN OR PUSH BACK DEFN */
                                                ; STATEMENT # 370
                          ; PROC  EVAL
          10C5  21A843          LXI     H,J+1H
          10C8  72              MOV     M,D
          10C9  2B              DCX     H
          10CA  73              MOV     M,E
          10CB  2B              DCX     H
          10CC  70              MOV     M,B
          10CD  2B              DCX     H
          10CE  71              MOV     M,C
          10CF  2B              DCX     H
          10D0  D1              POP     D
          10D1  C1              POP     B
          10D2  70              MOV     M,B
          10D3  2B              DCX     H
          10D4  71              MOV     M,C
          10D5  D5              PUSH    D
371  2            DECLARE ARGSTKPTR ADDRESS, ARGSTK BASED ARGSTKPTR (1) ADDRESS,
                         ARGNO BYTE, (I, J, K, M, N, T, TD) ADDRESS,
                         DIGITS(11) BYTE DATA (7FH,30H,31H,32H,33H,34H,35H,36H,37H,38H,39H),
                         /* PADDED ON LEFT TO SIMULATE FORTRAN ARRAY */
```

```
372  2          DODEF: PROCEDURE (ARGSTKPTR, I, J);  /* INSTAL DEFINITION
                                                        IN TABLE */
                                                ; STATEMENT # 372
                          ; PROC  DODEF
          127C  21B943          LXI     H,J+1H
          127F  72              MOV     M,D
          1280  2B              DCX     H
          1281  73              MOV     M,E
          1282  2B              DCX     H
          1283  70              MOV     M,B
          1284  2B              DCX     H
          1285  71              MOV     M,C
          1286  2B              DCX     H
          1287  D1              POP     D
          1288  C1              POP     B
          1289  70              MOV     M,B
          128A  2B              DCX     H
          128B  71              MOV     M,C
          128C  D5              PUSH    D
373  3                 DECLARE (ARGSTKPTR, I, J, A2, A3) ADDRESS,
                              ARGSTK BASED ARGSTKPTR (1) ADDRESS;
374  3                 IF J-I > 2 THEN    /* NB: "JAY MINUS EYE" (N.O.T
                                                        MINUS ONE) */
                                                ; STATEMENT # 374
          128D  01B643          LXI     B,I
          1290  11B843          LXI     D,J
          1293  CD0000          CALL    @P0098
          1296  3E02            MVI     A,2H
          1298  CD0000          CALL    @P0094
          129B  D2D712          JNC     @42
375  3                   DO;
376  4                     A2 = ARGSTK(I+2);
                                                ; STATEMENT # 376
          129E  2AB643          LHLD    I
          12A1  010400          LXI     B,ARGSTK+4H
          12A4  29              DAD     H
          12A5  09              DAD     B
          12A6  EB              XCHG
          12A7  2AB443          LHLD    ARGSTKPTR
          12AA  19              DAD     D
          12AB  5E              MOV     E,M
          12AC  23              INX     H
          12AD  56              MOV     D,M
          12AE  EB              XCHG
          12AF  22BA43          SHLD    A2
377  4                     A3 = ARGSTK(I+3);
                                                ; STATEMENT # 377
          12B2  2AB643          LHLD    I
          12B5  010600          LXI     B,ARGSTK+6H
          12B8  29              DAD     H
          12B9  09              DAD     B
          12BA  EB              XCHG
          12BB  2AB443          LHLD    ARGSTKPTR
          12BE  19              DAD     D
          12BF  5E              MOV     E,M
          12C0  23              INX     H
          12C1  56              MOV     D,M
          12C2  EB              XCHG
          12C3  22BC43          SHLD    A3
378  4                     CALL INSTAL(.EVALST(A2), .EVALST(A3));  /*
                                                        SUBARRAYS */
                                                ; STATEMENT # 378
          12C6  2ABA43          LHLD    A2
          12C9  012320          LXI     B,EVALST
          12CC  09              DAD     B
          12CD  E5              PUSH    H       ; 1
          12CE  2ABC43          LHLD    A3
          12D1  09              DAD     B
          12D2  EB              XCHG
          12D3  C1              POP     B       ; 1
          12D4  CDB50C          CALL    INSTAL
379  4                    END;
                                                ; STATEMENT # 379
                          @42:
380  3                  RETURN;
                                                ; STATEMENT # 380
          12D7  C9              RET
381  3                 END DODEF;
                                                ; STATEMENT # 381
```

```
382  2          DOIF: PROCEDURE(ARGSTK$PTR, I, J); /* EVALUATES IFELSE MACRO */
                                                ; STATEMENT # 382
                ; PROC  DOIF
     12D8 21C343        LXI     H,J+1H
     12DB 72            MOV     M,D
     12DC 2B            DCX     H
     12DD 73            MOV     M,E
     12DE 2B            DCX     H
     12DF 70            MOV     M,B
     12E0 2B            DCX     H
     12E1 71            MOV     M,C
     12E2 2B            DCX     H
     12E3 D1            POP     D
     12E4 C1            POP     B
     12E5 70            MOV     M,B
     12E6 2B            DCX     H
     12E7 71            MOV     M,C
     12E8 D5            PUSH    D
383  3                DECLARE (A2, A3, A4, A5, I, J, ARGSTK$PTR) ADDRESS,
                            ARGSTK BASED ARGSTK$PTR (1) ADDRESS;

384  3                IF J-I < 5 THEN    /* JAY MINUS EYE */
                                                ; STATEMENT # 384
     12E9 01C043        LXI     B,I
     12EC 11C243        LXI     D,J
     12EF CD0000        CALL    @P0098
     12F2 EB            XCHG
     12F3 3E05          MVI     A,5H
     12F5 CD0000        CALL    @P0096
     12F8 D2FC12        JNC     @43
385  3                  RETURN;
                                                ; STATEMENT # 385
     12FB C9            RET
                    @43:
386  3                  A2 = ARGSTK(I+2);
                                                ; STATEMENT # 386
     12FC 2AC043        LHLD    I
     12FF 010400        LXI     B,ARGSTK+4H
     1302 29            DAD     H
     1303 09            DAD     B
     1304 EB            XCHG
     1305 2ABE43        LHLD    ARGSTKPTR
     1308 19            DAD     D
     1309 5E            MOV     E,M
     130A 23            INX     H
     130B 56            MOV     D,M
     130C EB            XCHG
     130D 22C443        SHLD    A2
387  3                  A3 = ARGSTK(I+3);
                                                ; STATEMENT # 387
     1310 2AC043        LHLD    I
     1313 010600        LXI     B,ARGSTK+6H
     1316 29            DAD     H
     1317 09            DAD     B
     1318 EB            XCHG
     1319 2ABE43        LHLD    ARGSTKPTR
     131C 19            DAD     D
     131D 5E            MOV     E,M
     131E 23            INX     H
     131F 56            MOV     D,M
     1320 EB            XCHG
     1321 22C643        SHLD    A3
388  3                  A4 = ARGSTK(I+4);
                                                ; STATEMENT # 388
     1324 2AC043        LHLD    I
     1327 010800        LXI     B,ARGSTK+8H
     132A 29            DAD     H
     132B 09            DAD     B
     132C EB            XCHG
     132D 2ABE43        LHLD    ARGSTKPTR
     1330 19            DAD     D
     1331 5E            MOV     E,M
     1332 23            INX     H
     1333 56            MOV     D,M
     1334 EB            XCHG
     1335 22C843        SHLD    A4
389  3                  A5 = ARGSTK(I+5);
                                                ; STATEMENT # 389
     1338 2AC043        LHLD    I
     133B 010A00        LXI     B,ARGSTK+0AH
     133E 29            DAD     H
     133F 09            DAD     B
     1340 EB            XCHG
     1341 2ABE43        LHLD    ARGSTKPTR
     1344 19            DAD     D
     1345 5E            MOV     E,M
     1346 23            INX     H
     1347 56            MOV     D,M
     1348 EB            XCHG
     1349 22CA43        SHLD    A5
390  3                IF EQUAL(.EVALST(A2), .EVALST(A3)) = YES THEN    /* SUBARRAYS */
                                                ; STATEMENT # 390
     134C 2AC443        LHLD    A2
     134F 012320        LXI     B,EVALST
     1352 09            DAD     B
     1353 E5            PUSH    H       ; 1
     1354 2AC643        LHLD    A3
     1357 09            DAD     B
     1358 EB            XCHG
     1359 C1            POP     B       ; 1
     135A CD6A0D        CALL    EQUAL
     135D FE01          CPI     1H
     135F C27113        JNZ     @44
391  3                  CALL PBSTR(.EVALST(A4));
                                                ; STATEMENT # 391
     1362 2AC843        LHLD    A4
     1365 012320        LXI     B,EVALST
     1368 09            DAD     B
     1369 44            MOV     B,H
     136A 4D            MOV     C,L
     136B CDFA0A        CALL    PBSTR
     136E C37D13        JMP     @45
                    @44:
                    ELSE
392  3                  CALL PBSTR(.EVALST(A5));
                                                ; STATEMENT # 392
     1371 2ACA43        LHLD    A5
     1374 012320        LXI     B,EVALST
     1377 09            DAD     B
     1378 44            MOV     B,H
     1379 4D            MOV     C,L
     137A CDFA0A        CALL    PBSTR
                    @45:
393  3                  RETURN;
                                                ; STATEMENT # 393
     137D C9            RET
394  3                  END DOIF;
                                                ; STATEMENT # 394


395  2          CTOI: PROCEDURE(IN$PTR, I) ADDRESS; /* ASCII TO INTEGER CONVERSION
                                                ; STATEMENT # 395
                ; PROC  CTOI
     137E 21CF43        LXI     H,I+1H
     1381 72            MOV     M,D
     1382 2B            DCX     H
     1383 73            MOV     M,E
     1384 2B            DCX     H
     1385 70            MOV     M,B
     1386 2B            DCX     H
     1387 71            MOV     M,C
                                                AND BUMP INDEX I
396  3                  DECLARE (D, I, IN$PTR, VAL) ADDRESS, IN BASED IN$PTR (1) BYTE

397  3                  DO WHILE IN(I) = BLANK;
                                                ; STATEMENT # 397
                    @136:
     1388 2ACE43        LHLD    I
     138B EB            XCHG
     138C 2ACC43        LHLD    INPTR
     138F 19            DAD     D
     1390 7E            MOV     A,M
     1391 FE20          CPI     20H
     1393 C2A013        JNZ     @137
398  4                    I = I + 1;
                                                ; STATEMENT # 398
     1396 2ACE43        LHLD    I
     1399 23            INX     H
     139A 22CE43        SHLD    I
399  4                  END;
                                                ; STATEMENT # 399
     139D C38813        JMP     @136
                    @137:
400  3                  VAL = 0;
                                                STATEMENT # 400
     13A0 210000        LXI     H,0H
     13A3 22D243        SHLD    VAL
```

```
401  3                 DO WHILE IN(I) <> EOS;
                                              ; STATEMENT # 401
                       @138:
          13A6  2ACE43          LHLD    I
          13A9  EB              XCHG
          13AA  2ACC43          LHLD    INPTR
          13AD  19              DAD     D
          13AE  7E              MOV     A,M
          13AF  FEFE            CPI     0FEH
          13B1  CAF413          JZ      @139
402  4                  D = INDEX( DIGITS, IN(I));
                                              ; STATEMENT # 402
          13B4  2ACE43          LHLD    I
          13B7  EB              XCHG
          13B8  2ACC43          LHLD    INPTR
          13BB  19              DAD     D
          13BC  5E              MOV     E,M
          13BD  013500          LXI     B,DIGITS
          13C0  CD300B          CALL    INDEX
          13C3  6F              MOV     L,A
          13C4  2600            MVI     H,0
          13C6  22D043          SHLD    D
403  4                  IF D = 0 THEN              /*  NON-DIGIT  */
                                              ; STATEMENT # 403
          13C9  3E00            MVI     A,0H
          13CB  11D043          LXI     D,D
          13CE  CD0000          CALL    @P0101
          13D1  B5              ORA     L
          13D2  C2D813          JNZ     @46
404  4                   GOTO QUIT1;
                                              ; STATEMENT # 404
          13D5  C3F413          JMP     QUIT1
                       @46:
405  4                  VAL = 10*VAL + D - 1;
                                              ; STATEMENT # 405
          13D8  21D243          LXI     H,VAL
          13DB  CD0000          CALL    @P0031
          13DE  E5              PUSH    H       ; 1
          13DF  2AD043          LHLD    D
          13E2  C1              POP     B       ; 1
          13E3  09              DAD     B
          13E4  2B              DCX     H
          13E5  EB              XCHG
          13E6  2B              DCX     H
          13E7  73              MOV     M,E
          13E8  23              INX     H
          13E9  72              MOV     M,D
406  4                  I = I + 1;
                                              ; STATEMENT # 406
          13EA  2ACE43          LHLD    I
          13ED  23              INX     H
          13EE  22CE43          SHLD    I
407  4                END /*  DO-WHILE  */;
                                              ; STATEMENT # 407
          13F1  C3A613          JMP     @138
                       @139:
408  3                QUIT1:  RETURN VAL;
                                              ; STATEMENT # 408
                       QUIT1:
          13F4  2AD243          LHLD    VAL
          13F7  C9              RET
409  3                END CTOI;
                                              ; STATEMENT # 409


410  2           PBNUM: PROCEDURE(N); /*  INTEGER TO ASCII + PUSHBACK TO BUF  */
                                              ; STATEMENT # 410
                  ; PROC  PBNUM
          13F8  21D543          LXI     H,N+1H
          13FB  70              MOV     M,B
          13FC  2B              DCX     H
          13FD  71              MOV     M,C
411  3                  DECLARE (M, N, NUM) ADDRESS;

412  3                  NUM = N;
                                              ; STATEMENT # 412
          13FE  2AD443          LHLD    N
          1401  22D843          SHLD    NUM
413  3                  REPEAT:
                                              ; STATEMENT # 413
                       REPEAT:
                            DO;
414  4                       M = NUM MOD 10;
                                              ; STATEMENT # 414
          1404  2AD843          LHLD    NUM
          1407  EB              XCHG
          1408  210A00          LXI     H,0AH
          140B  CD0000          CALL    @P0029
          140E  22D643          SHLD    M
415  4                       CALL PUTBAK(DIGITS(M+1));
                                              ; STATEMENT # 415
          1411  2AD643          LHLD    M
          1414  013600          LXI     B,DIGITS+1H
          1417  09              DAD     B
          1418  4E              MOV     C,M
          1419  CDCD0A          CALL    PUTBAK
416  4                       NUM = NUM/10;
                                              ; STATEMENT # 416
          141C  2AD843          LHLD    NUM
          141F  EB              XCHG
          1420  210A00          LXI     H,0AH
          1423  CD0000          CALL    @P0029
          1426  EB              XCHG
          1427  22D843          SHLD    NUM
417  4                       END;
                            /*  UNTIL NUM = 0  */
418  3                       DO WHILE NUM <> 0;
                                              ; STATEMENT # 418
                       @140:
          142A  3E00            MVI     A,0H
          142C  11D843          LXI     D,NUM
          142F  CD0000          CALL    @P0101
          1432  B5              ORA     L
          1433  CA3C14          JZ      @141
419  4                        GOTO REPEAT;
                                              ; STATEMENT # 419
          1436  C30414          JMP     REPEAT
420  4                       END;  /* PHONIED UP REPEAT-UNTIL  */
                                              ; STATEMENT # 420
          1439  C32A14          JMP     @140
                       @141:
421  3                       RETURN;
                                              ; STATEMENT # 421
          143C  C9              RET
422  3                       END PBNUM;
                                              ; STATEMENT # 422


423  2           DOINCR: PROCEDURE(I, J);    /*  INCREMENT ARGUMENT BY ONE  */
                                              ; STATEMENT # 423
                  ; PROC  DOINCR
          143D  21DD43          LXI     H,J+1H
          1440  72              MOV     M,D
          1441  2B              DCX     H
          1442  73              MOV     M,E
          1443  2B              DCX     H
          1444  70              MOV     M,B
          1445  2B              DCX     H
          1446  71              MOV     M,C
424  3                       DECLARE (I, J, K) ADDRESS;

425  3                       K = ARGSTK(I+2);
                                              ; STATEMENT # 425
          1447  2ADA43          LHLD    I
          144A  010400          LXI     B,ARGSTK+4H
          144D  29              DAD     H
          144E  09              DAD     B
          144F  EB              XCHG
          1450  2AA343          LHLD    ARGSTKPTR
          1453  19              DAD     D
          1454  5E              MOV     E,M
          1455  23              INX     H
          1456  56              MOV     D,M
          1457  EB              XCHG
          1458  22DE43          SHLD    K
426  3                       CALL PBNUM(CTOI(.EVALST, K) + 1);
                                              ; STATEMENT # 426
          145B  2ADE43          LHLD    K
          145E  EB              XCHG
          145F  012320          LXI     B,EVALST
          1462  CD7E13          CALL    CTOI
          1465  23              INX     H
          1466  44              MOV     B,H
          1467  4D              MOV     C,L
          1468  CDF813          CALL    PBNUM
427  3                       RETURN;
                                              ; STATEMENT # 427
          146B  C9              RET
```

```
428  3                  END DOINCR;
                                        ; STATEMENT # 428


429  2          DODECR: PROCEDURE(I, J);    /* DECREMENT ARGUMENT BY ONE */
                                        ; STATEMENT # 429
                    ; PROC  DODECR
       146C 21E343          LXI     H,J+1H
       146F 72              MOV     M,D
       1470 2B              DCX     H
       1471 73              MOV     M,E
       1472 2B              DCX     H
       1473 70              MOV     M,B
       1474 2B              DCX     H
       1475 71              MOV     M,C
430  3                  DECLARE (I, J, K) ADDRESS;

431  3                  K = ARGSTK(I+2);
                                        ; STATEMENT # 431
       1476 2AE043          LHLD    I
       1479 010400          LXI     B,ARGSTK+4H
       147C 29              DAD     H
       147D 09              DAD     B
       147E EB              XCHG
       147F 2AA343          LHLD    ARGSTKPTR
       1482 19              DAD     D
       1483 5E              MOV     E,M
       1484 23              INX     H
       1485 56              MOV     D,M
       1486 EB              XCHG
       1487 22E443          SHLD    K
432  3                  CALL PBNUM(CTOI(.EVALST, K) - 1);
                                        ; STATEMENT # 432
       148A 2AE443          LHLD    K
       148D EB              XCHG
       148E 012320          LXI     B,EVALST
       1491 CD7E13          CALL    CTOI
       1494 2B              DCX     H
       1495 44              MOV     B,H
       1496 4D              MOV     C,L
       1497 CDF813          CALL    PBNUM
433  3                  RETURN;
                                        ; STATEMENT # 433
       149A C9              RET
434  3                  END DODECR;
                                        ; STATEMENT # 434


435  2          MIN: PROCEDURE(M,N) ADDRESS;
                                        ; STATEMENT # 435
                    ; PROC  MIN
       149B 21E943          LXI     H,N+1H
       149E 72              MOV     M,D
       149F 2B              DCX     H
       14A0 73              MOV     M,E
       14A1 2B              DCX     H
       14A2 70              MOV     M,B
       14A3 2B              DCX     H
       14A4 71              MOV     M,C
436  3                DECLARE (M, N, VAL) ADDRESS;

437  3                IF M <= N THEN
                                        ; STATEMENT # 437
       14A5 11E843          LXI     D,N
       14A8 01E643          LXI     B,M
       14AB CD0000          CALL    @P0098
       14AE DAB514          JC      @47
438  3                  RETURN M;
                                        ; STATEMENT # 438
       14B1 2AE643          LHLD    M
       14B4 C9              RET
                  @47:
439  3                ELSE RETURN N;
                                        ; STATEMENT # 439
       14B5 2AE843          LHLD    N
       14B8 C9              RET
                  @48:
440  3                END MIN;
                                        ; STATEMENT # 440
       14B9 C9              RET
```

```
441  2          DOSUBSTR: PROCEDURE(I, J);    /* SELECT SUBSTRING */
                                        ; STATEMENT # 441
                    ; PROC  DOSUBSTR
       14BA 21EF43          LXI     H,J+1H
       14BD 72              MOV     M,D
       14BE 2B              DCX     H
       14BF 73              MOV     M,E
       14C0 2B              DCX     H
       14C1 70              MOV     M,B
       14C2 2B              DCX     H
       14C3 71              MOV     M,C
442  3                  DECLARE (AP, FC, I, J, K, NC) ADDRESS;

443  3                  IF J-I < 3 THEN
                                        ; STATEMENT # 443
       14C4 01EC43          LXI     B,I
       14C7 11EE43          LXI     D,J
       14CA CD0000          CALL    @P0098
       14CD EB              XCHG
       14CE 3E03            MVI     A,3H
       14D0 CD0000          CALL    @P0096
       14D3 D2D714          JNC     @49
444  3                   RETURN;
                                        ; STATEMENT # 444
       14D6 C9              RET
                  @49:
445  3                  IF J-I < 4 THEN
                                        ; STATEMENT # 445
       14D7 01EC43          LXI     B,I
       14DA 11EE43          LXI     D,J
       14DD CD0000          CALL    @P0098
       14E0 EB              XCHG
       14E1 3E04            MVI     A,4H
       14E3 CD0096          CALL    @P0096
       14E6 D2F214          JNC     @50
446  3                     NC = MAXTA
                                        ; STATEMENT # 446
       14E9 217A00          LXI     H,7AH
       14EC 22F643          SHLD    NC
       14EF C31315          JMP     @51
                  @50:
                    ELSE
447  3                     DO;
448  4                       A = ARGSTK(I+4);
                                        ; STATEMENT # 448
       14F2 2AEC43          LHLD    I
       14F5 010800          LXI     B,ARGSTK+8H
       14F8 29              DAD     H
       14F9 09              DAD     B
       14FA EB              XCHG
       14FB 2AA343          LHLD    ARGSTKPTR
       14FE 19              DAD     D
       14FF 5E              MOV     E,M
       1500 23              INX     H
       1501 56              MOV     D,M
       1502 EB              XCHG
       1503 22F443          SHLD    K
449  4                       NC = CTOI(.EVALST, A);      /* NUMBER OF CHARS */
                                        ; STATEMENT # 449
       1506 2AF443          LHLD    K
       1509 EB              XCHG
       150A 012320          LXI     B,EVALST
       150D CD7E13          CALL    CTOI
       1510 22F643          SHLD    NC
450  4                       END;
                                        ; STATEMENT # 450
                  @51:
451  3                  K = ARGSTK(I+3);          /* ORIGIN */
                                        ; STATEMENT # 451
       1513 2AEC43          LHLD    I
       1516 010600          LXI     B,ARGSTK+6H
       1519 29              DAD     H
       151A 09              DAD     B
       151B EB              XCHG
       151C 2AA343          LHLD    ARGSTKPTR
       151F 19              DAD     D
       1520 5E              MOV     E,M
       1521 23              INX     H
       1522 56              MOV     D,M
       1523 EB              XCHG
       1524 22F443          SHLD    K
```

```
452  3                    AP = ARGSTK(I+2);        /*  TARGET STRING  */
                                                   ;  STATEMENT # 452
        1527  2AEC43        LHLD    I
        152A  010400        LXI     B,ARGSTK+4H
        152D  29            DAD     H
        152E  09            DAD     B
        152F  EB            XCHG
        1530  2AA343        LHLD    ARGSTKPTR
        1533  19            DAD     D
        1534  5E            MOV     E,M
        1535  23            INX     H
        1536  56            MOV     D,M
        1537  EB            XCHG
        1538  22F043        SHLD    AP
453  3                    FC = AP + OTOI( EVALST, K) - 1    /*  FIRST CHAR OF SUBSTRING  */
                                                   ;  STATEMENT # 453
        153B  2AF443        LHLD    K
        153E  EB            XCHG
        153F  012320        LXI     B,EVALST
        1542  CD7E13        CALL    OTOI
        1545  EB            XCHG
        1546  2AF043        LHLD    AP
        1549  19            DAD     D
        154A  2B            DCX     H
        154B  22F243        SHLD    FC
454  3                    IF (FC >= AP) AND (FC < AP + LENGTH( EVALST(AP))) THEN
                                                   ;  STATEMENT # 454
        154E  01F043        LXI     B,AP
        1551  11F243        LXI     D,FC
        1554  CD0000        CALL    @P0098
        1557  9F            SBB     A
        1558  2F            CMA
        1559  F5            PUSH    PSW     ;  1
        155A  2AF043        LHLD    AP
        155D  012320        LXI     B,EVALST
        1560  09            DAD     B
        1561  44            MOV     B,H
        1562  4D            MOV     C,L
        1563  CDB006        CALL    LENGTH
        1566  EB            XCHG
        1567  2AF043        LHLD    AP
        156A  19            DAD     D
        156B  11F243        LXI     D,FC
        156E  CD0000        CALL    @P0102
        1571  9F            SBB     A
        1572  C1            POP     B       ;  1
        1573  48            MOV     C,B
        1574  A1            ANA     C
        1575  1F            RAR
        1576  D2B815        JNC     @52
455  3                      DO;
456  4                        K = FC + MIN(NC, LENGTH( EVALST(FC))) - 1;
                                                   ;  STATEMENT # 456
        1579  2AF243        LHLD    FC
        157C  012320        LXI     B,EVALST
        157F  09            DAD     B
        1580  44            MOV     B,H
        1581  4D            MOV     C,L
        1582  CDB006        CALL    LENGTH
        1585  EB            XCHG
        1586  2AF643        LHLD    NC
        1589  44            MOV     B,H
        158A  4D            MOV     C,L
        158B  CD9B14        CALL    MIN
        158E  EB            XCHG
        158F  2AF243        LHLD    FC
        1592  19            DAD     D
        1593  2B            DCX     H
        1594  22F443        SHLD    K
457  4                        DO WHILE K >= FC;
                                                   ;  STATEMENT # 457
                    @142:
        1597  01F243        LXI     B,FC
        159A  11F443        LXI     D,K
        159D  CD0000        CALL    @P0098
        15A0  DAB815        JC      @143
458  5                          CALL PUTBAK(EVALST(K));
                                                   ;  STATEMENT # 458
        15A3  2AF443        LHLD    K
        15A6  012320        LXI     B,EVALST
        15A9  09            DAD     B
        15AA  4E            MOV     C,M
        15AB  CDCD0A        CALL    PUTBAK
459  5                          K = K - 1;
                                                   ;  STATEMENT # 459
        15AE  2AF443        LHLD    K
        15B1  2B            DCX     H
        15B2  22F443        SHLD    K
460  5                        END;  /*  DO-WHILE  */
                                                   ;  STATEMENT # 460
        15B5  C39715        JMP     @142
                    @143:
461  4                      END;
                                                   ;  STATEMENT # 461
                    @52:
462  3                    RETURN;
                                                   ;  STATEMENT # 462
        15B8  C9            RET
463  3                  END DOSUBSTR;
                                                   ;  STATEMENT # 463


                /*  E V A L   E X E C U T I O N   B E G I N S   H E R E  */


464  2                T = ARGSTK(I);
                                                   ;  STATEMENT # 464
        10D6  2AA543        LHLD    I
        10D9  29            DAD     H
        10DA  EB            XCHG
        10DB  2AA343        LHLD    ARGSTKPTR
        10DE  19            DAD     D
        10DF  5E            MOV     E,M
        10E0  23            INX     H
        10E1  56            MOV     D,M
        10E2  EB            XCHG
        10E3  22B043        SHLD    T
465  2                TD = EVALST(T);
                                                   STATEMENT # 465
        10E6  012320        LXI     B,EVALST
        10E9  09            DAD     B
        10EA  6E            MOV     L,M
        10EB  2600          MVI     H,0
        10ED  22B243        SHLD    TD
466  2                IF TD = DEFTYPE THEN
                                                   ;  STATEMENT # 466
        10F0  3E1F          MVI     A,1FH
        10F2  CD0000        CALL    @P0094
        10F5  B5            ORA     L
        10F6  C20C11        JNZ     @53
467  2                  CALL DODEF( ARGSTK, I, J);
                                                   ;  STATEMENT # 467
        10F9  2AA343        LHLD    ARGSTKPTR
        10FC  E5            PUSH    H       ;  1
        10FD  2AA543        LHLD    I
        1100  44            MOV     B,H
        1101  4D            MOV     C,L
        1102  2AA743        LHLD    J
        1105  EB            XCHG
        1106  CD7C12        CALL    DODEF
        1109  C37B12        JMP     @54
                    @53:
                ELSE
468  2                IF TD = IFTYPE THEN
                                                   ;  STATEMENT # 468
        110C  3E1E          MVI     A,1EH
        110E  11B243        LXI     D,TD
        1111  CD0000        CALL    @P0101
        1114  B5            ORA     L
        1115  C22B11        JNZ     @55
469  2                  CALL DOIF( ARGSTK, I, J);
                                                   ;  STATEMENT # 469
        1118  2AA343        LHLD    ARGSTKPTR
        111B  E5            PUSH    H       ;  1
        111C  2AA543        LHLD    I
        111F  44            MOV     B,H
        1120  4D            MOV     C,L
        1121  2AA743        LHLD    J
        1124  EB            XCHG
        1125  CDD812        CALL    DOIF
        1128  C37B12        JMP     @56
                    @55:
                ELSE
```

```
470  2          IF TD = DELETYPE THEN
                                          ; STATEMENT # 470
        112B 3E1D          MVI    A,1DH
        112D 11B243        LXI    D,TD
        1130 CD0000        CALL   @P0101
        1133 B5            ORA    L
        1134 C25A11        JNZ    @57
471  2            DO;
472  3              J = ARGSTK(I+2);
                                          ; STATEMENT # 472
        1137 2AA543        LHLD   I
        113A 010400        LXI    B,ARGSTK+4H
        113D 29            DAD    H
        113E 09            DAD    B
        113F EB            XCHG
        1140 2AA343        LHLD   ARGSTKPTR
        1143 19            DAD    D
        1144 5E            MOV    E,M
        1145 23            INX    H
        1146 56            MOV    D,M
        1147 EB            XCHG
        1148 22A743        SHLD   J
473  3              CALL DELETE(.EVALST(J));
                                          ; STATEMENT # 473
        114B 2AA743        LHLD   J
        114E 012320        LXI    B,EVALST
        1151 09            DAD    B
        1152 44            MOV    B,H
        1153 4D            MOV    C,L
        1154 CD870E        CALL   DELETE
474  3            END; /* IF-DO */
                                          ; STATEMENT # 474
        1157 C37B12        JMP    @58
                  @57:
               ELSE
475  2          IF TD = INCRTYPE THEN
                                          ; STATEMENT # 475
        115A 3E1C          MVI    A,1CH
        115C 11B243        LXI    D,TD
        115F CD0000        CALL   @P0101
        1162 B5            ORA    L
        1163 C27511        JNZ    @59
476  2            CALL DOINCR(I, J);
                                          ; STATEMENT # 476
        1166 2AA543        LHLD   I
        1169 44            MOV    B,H
        116A 4D            MOV    C,L
        116B 2AA743        LHLD   J
        116E EB            XCHG
        116F CD3D14        CALL   DOINCR
        1172 C37B12        JMP    @60
                  @59:
               ELSE
477  2          IF TD = DECRTYPE THEN
                                          ; STATEMENT # 477
        1175 3E1A          MVI    A,1AH
        1177 11B243        LXI    D,TD
        117A CD0000        CALL   @P0101
        117D B5            ORA    L
        117E C29011        JNZ    @61
478  2            CALL DODECR(I, J);
                                          ; STATEMENT # 478
        1181 2AA543        LHLD   I
        1184 44            MOV    B,H
        1185 4D            MOV    C,L
        1186 2AA743        LHLD   J
        1189 EB            XCHG
        118A CD6C14        CALL   DODECR
        118D C37B12        JMP    @62
                  @61:
               ELSE
479  2          IF TD = SUBSTRTYPE THEN
                                          ; STATEMENT # 479
        1190 3E19          MVI    A,19H
        1192 11B243        LXI    D,TD
        1195 CD0000        CALL   @P0101
        1198 B5            ORA    L
        1199 C2AB11        JNZ    @63

480  2            CALL DOSUBSTR(I, J);
                                          STATEMENT # 480
        119C 2AA543        LHLD   I
        119F 44            MOV    B,H
        11A0 4D            MOV    C,L
        11A1 2AA743        LHLD   J
        11A4 EB            XCHG
        11A5 CDBA14        CALL   DOSUBSTR
        11A8 C37B12        JMP    @64
                  @63:
               ELSE
481  2            DO;
482  3              K = T + LENGTH(.EVALST(T)) - 1;
                                          ; STATEMENT # 482
        11AB 2AB043        LHLD   T
        11AE 012320        LXI    B,EVALST
        11B1 09            DAD    B
        11B2 44            MOV    B,H
        11B3 4D            MOV    C,L
        11B4 CDB006        CALL   LENGTH
        11B7 EB            XCHG
        11B8 2AB043        LHLD   T
        11BB 19            DAD    D
        11BC 2B            DCX    H
        11BD 22AA43        SHLD   K
483  3            DO WHILE (K > T) AND (K <> OFFFFH);    /* PLM INTEGERS ARE NON-NEGATIVE */
                                          ; STATEMENT # 483
                  @144:
        11C0 11B043        LXI    D,T
        11C3 01AA43        LXI    B,K
        11C6 CD0000        CALL   @P0098
        11C9 9F            SBB    A
        11CA 01FFFF        LXI    B,0FFFFH
        11CD 11AA43        LXI    D,K
        11D0 F5            PUSH   PSW      ; 1
        11D1 CD0000        CALL   @P0100
        11D4 B5            ORA    L
        11D5 C6FF          ADI    255
        11D7 9F            SBB    A
        11D8 C1            POP    B        ; 1
        11D9 48            MOV    C,B
        11DA A1            ANA    C
        11DB 1F            RAR
        11DC D26312        JNC    @145
484  4            IF EVALST(K-1) <> ARGFLAG THEN
                                          ; STATEMENT # 484
        11DF 2AAA43        LHLD   K
        11E2 2B            DCX    H
        11E3 012320        LXI    B,EVALST
        11E6 09            DAD    B
        11E7 7E            MOV    A,M
        11E8 FE24          CPI    24H
        11EA CAFB11        JZ     @65
485  4              CALL PUTBAK(EVALST(K));
                                          ; STATEMENT # 485
        11ED 2AAA43        LHLD   K
        11F0 012320        LXI    B,EVALST
        11F3 09            DAD    B
        11F4 4E            MOV    C,M
        11F5 CDCD0A        CALL   PUTBAK
        11F8 C35912        JMP    @66
                  @65:
               ELSE
486  4                DO
487  5                  ARGNO = INDEX(.DIGITS, EVALST(K)) - 1;
                                          STATEMENT # 487
        11FB 2AAA43        LHLD   K
        11FE 012320        LXI    B,EVALST
        1201 09            DAD    B
        1202 5E            MOV    E,M
        1203 013500        LXI    B,DIGITS
        1206 CD300B        CALL   INDEX
        1209 3D            DCR    A
        120A 32A943        STA    ARGNO
488  5                  IF (ARGNO <> 0FFH) AND (ARGNO < J-I) THEN    /* "JAY MINUS EYE" */
                                          ; STATEMENT # 488
        120D 3AA943        LDA    ARGNO
        1210 D6FF          SUI    0FFH
        1212 C6FF          ADI    255
        1214 9F            SBB    A
        1215 01A543        LXI    B,I
        1218 11A743        LXI    D,J
        121B F5            PUSH   PSW      1
        121C CD0000        CALL   @P0098
        121F 13            INX    D
        1220 1A            LDAX   D
```

```
                1221  CD0000          CALL    @P0094
                1224  9F              SBB     A
                1225  C1              POP     B       ; 1
                1226  48              MOV     C,B
                1227  A1              ANA     C
                1228  1F              RAR
                1229  D25212          JNC     @67
489   5                                 DO;
490   6                                   N = I + ARGNO + 1;
                                                      ; STATEMENT # 490
                122C  3AA943          LDA     ARGNO
                122F  11A543          LXI     D,I
                1232  CD0000          CALL    @P0014
                1235  23              INX     H
                1236  22AE43          SHLD    N
491   6                                   M = ARGSTK(N);
                                                      ; STATEMENT # 491
                1239  29              DAD     H
                123A  EB              XCHG
                123B  2AA343          LHLD    ARGSTKPTR
                123E  19              DAD     D
                123F  5E              MOV     E,M
                1240  23              INX     H
                1241  56              MOV     D,M
                1242  EB              XCHG
                1243  22AC43          SHLD    M
492   6                                   CALL PBSTR(.EVALST(M));
                                                      ; STATEMENT # 492
                1246  2AAC43          LHLD    M
                1249  012320          LXI     B,EVALST
                124C  09              DAD     B
                124D  44              MOV     B,H
                124E  4D              MOV     C,L
                124F  CDFA0A          CALL    PBSTR
493   6                                 END /*  IF-DO ... */;
                                                      ; STATEMENT # 493
                              @67:
494   5                             K = K - 1;      /* SKIP OVER $  */
                                                      ; STATEMENT # 494
                1252  2AAA43          LHLD    K
                1255  2B              DCX     H
                1256  22AA43          SHLD    K
495   5                           END /*  ELSE-DO; ... */;
                                                      ; STATEMENT # 495
                              @66:
496   4                           K = K - 1;
                                                      ; STATEMENT # 496
                1259  2AAA43          LHLD    K
                125C  2B              DCX     H
                125D  22AA43          SHLD    K
497   4                         END /* DO-WHILE */;
                                                      ; STATEMENT # 497
                1260  C3C011        JMP     @144
                              @145:
498   3                         IF K = T THEN    /*  DO LAST CHARACTER  */
                                                      ; STATEMENT # 498
                1263  01B043          LXI     B,T
                1266  11AA43          LXI     D,K
                1269  CD0000          CALL    @P0098
                126C  B5              ORA     L
                126D  C27B12          JNZ     @68
499   3                           CALL PUTBAK(EVALST(K));
                                                      ; STATEMENT # 499
                1270  2AAA43          LHLD    K
                1273  012320          LXI     B,EVALST
                1276  09              DAD     B
                1277  4E              MOV     C,M
                1278  CDCD0A          CALL    PUTBAK
                              @68:
500   3                       END /*  ELSE-DO ... */;
                                                      ; STATEMENT # 500
                              @64:
                              @62:
                              @60:
                              @58:
                              @56:
                              @54:
501   2                       RETURN;
                                                      ; STATEMENT # 501
                127B  C9              RET
502   2                     END EVAL;
                                                      ; STATEMENT # 502
```

```
                      /*   M A I N L I N E   D R I V E R   B E G I N S   H E R E   */


503   1     CALLSIGNON: CALL SIGNON;
                                                      ; STATEMENT # 503
                0231  310000          LXI     SP,@STACK$ORIGIN
                                CALLSIGNON:
                0234  310000          LXI     SP,@STACK$ORIGIN
                0237  CDC307          CALL    SIGNON
504   1               CALL OPEN(.SAVAFT, .('F1:KAPIAR.SAV '), 2, 0, .STATUS);
                                                      ; STATEMENT # 504
                023A  010C00          LXI     B,SAVAFT
                023D  C5              PUSH    B       ; 1
                023E  01CD01          LXI     B,$-71H
                0241  C5              PUSH    B       ; 2
                0242  010200          LXI     B,2H
                0245  C5              PUSH    B       ; 3
                0246  110E00          LXI     D,STATUS
                0249  010000          LXI     B,0H
                024C  CD0000          CALL    OPEN

                      /*            I N I T I A L I Z A T I O N S                 */

505   1               AP, BP, EP = 0;     /*  FORTRAN TO PLM ARRAY CONVERSION  */
                                                      ; STATEMENT # 505
                024F  210000          LXI     H,0H
                0252  221000          SHLD    AP
                0255  221200          SHLD    BP
                0258  221600          SHLD    EP
506   1               CP, PLAST, TLAST = 0FFFFH;
                                                      ; STATEMENT # 506
                025B  2B              DCX     H
                025C  221400          SHLD    CP
                025F  221C00          SHLD    PLAST
                0262  222000          SHLD    TLAST
507   1               STKSAV0 = STACKPTR;
                                                      ; STATEMENT # 507
                0265  210000          LXI     H,0     ; 0
                0268  39              DAD     SP
                0269  221E00          SHLD    STKSAV0
508   1               BUF(BP) = EOS;
                                                      ; STATEMENT # 508
                026C  2A1200          LHLD    BP
                026F  012210          LXI     B,BUF
                0272  09              DAD     B
                0273  36FE            MVI     M,0FEH
509   1               START = 1;
                                                      ; STATEMENT # 509
                0275  21A33F          LXI     H,START
                0278  3601            MVI     M,1H
510   1               CALL INSTAL(.DEFNAM, .DEFTYP);
                                                      ; STATEMENT # 510
                027A  110A00          LXI     D,DEFTYP
                027D  010300          LXI     B,DEFNAM
                0280  CDB50C          CALL    INSTAL
511   1               CALL INSTAL(.CONDNAM, .CONDTYP);
                                                      ; STATEMENT # 511
                0283  111300          LXI     D,CONDTYP
                0286  010C00          LXI     B,CONDNAM
                0289  CDB50C          CALL    INSTAL
512   1               CALL INSTAL(.DELTNAM, .DELTYP);
                                                      ; STATEMENT # 512
                028C  111C00          LXI     D,DELTYP
                028F  011500          LXI     B,DELTNAM
                0292  CDB50C          CALL    INSTAL
513   1               CALL INSTAL(.INCRNAM, .INCRTYP);
                                                      ; STATEMENT # 513
                0295  112F00          LXI     D,INCRTYP
                0298  011E00          LXI     B,INCRNAM
                029B  CDB50C          CALL    INSTAL
514   1               CALL INSTAL(.DECRNAM, .DECRTYP);
                                                      ; STATEMENT # 514
                029E  113100          LXI     D,DECRTYP
                02A1  012300          LXI     B,DECRNAM
                02A4  CDB50C          CALL    INSTAL
515   1               CALL INSTAL(.SUBSTRNAM, .SUBSTRTYP);
                                                      ; STATEMENT # 515
                02A7  113300          LXI     D,SUBSTRTYP
                02AA  012800          LXI     B,SUBSTRNAM
                02AD  CDB50C          CALL    INSTAL
```

```
                /*     "A C T I V E"    C O D E    B E G I N S    H E R E      */

516  1      RESTART: T = GETTOK(.TOKEN, MAXTOK);
                                                    ; STATEMENT # 516
                      RESTART:
          02B0  1E7A            MVI     E,7AH
          02B2  01A43F          LXI     B,TOKEN
          02B5  CDBB0B          CALL    GETTOK
          02B8  322D43          STA     T
517  1                DO WHILE T <> EOFF;
                                                    ; STATEMENT # 517
                      @146:
          02BB  3A2D43          LDA     T
          02BE  FEFF            CPI     0FFH
          02C0  CA0D05          JZ      @147
518  2                IF T = ALPHA THEN                                  /*  CASE 1  */
                                                    ; STATEMENT # 518
          02C3  3A2D43          LDA     T
          02C6  FE41            CPI     41H
          02C8  C28003          JNZ     @69
519  2                  DO;
520  3                  IF LOOKUP(.TOKEN, .DEFN) = NO THEN
                                                    ; STATEMENT # 520
          02CB  112332          LXI     D,DEFN
          02CE  01A43F          LXI     B,TOKEN
          02D1  CD2410          CALL    LOOKUP
          02D4  FE00            CPI     0H
          02D6  C2E202          JNZ     @70
521  3                    CALL PUTTOK(.TOKEN);
                                                    ; STATEMENT # 521
          02D9  01A43F          LXI     B,TOKEN
          02DC  CD750F          CALL    PUTTOK
          02DF  C37D03          JMP     @71
                      @70:
                    ELSE     /*  DEFINED, PUT IT IN EVAL STACK  */
522  3                   DO;
523  4                     CP = CP + 1;    /*  THIS IS THE ONLY PLACE THAT CP
                                                          GETS INCREMENTED  */
                                                    ; STATEMENT # 523
          02E2  2A1400          LHLD    CP
          02E5  23              INX     H
          02E6  221400          SHLD    CP
524  4                    IF CP > CALLSIZE THEN
                                                    ; STATEMENT # 524
          02E9  110001          LXI     D,100H
          02EC  CD0000          CALL    @P0095
          02EF  D2FB02          JNC     @72
525  4                    DO;
526  5                      CALL REMARK(.('*** ERROR: CALL-STACK OVERFLOW ',EOS));
                                                    ; STATEMENT # 526
          02F2  01DC01          LXI     B,$-116H
          02F5  CDE306          CALL    REMARK
527  5                       GOTO RECOVER;
                                                    ; STATEMENT # 527
          02F8  C34305          JMP     RECOVER
528  5                      END;
                                                    ; STATEMENT # 528
                      @72:
529  4                    CALLST(CP) = AP;
                                                    ; STATEMENT # 529
          02FB  2A1400          LHLD    CP
          02FE  012D40          LXI     B,CALLST
          0301  29              DAD     H
          0302  09              DAD     B
          0303  E5              PUSH    H       ; 1
          0304  2A1000          LHLD    AP
          0307  EB              XCHG
          0308  E1              POP     H       ; 1
          0309  73              MOV     M,E
          030A  23              INX     H
          030B  72              MOV     M,D
530  4                    AP = PUSH(EP, .ARGSTK, AP);
                                                    ; STATEMENT # 530
          030C  2A1600          LHLD    EP
          030F  E5              PUSH    H       ; 1
          0310  2A1000          LHLD    AP
          0313  EB              XCHG
          0314  012336          LXI     B,ARGSTK
          0317  CDE70F          CALL    PUSH
          031A  221000          SHLD    AP
531  4                    CALL PUTTOK(.DEFN);    /*  STACK DEFINITION  */
                                                    ; STATEMENT # 531
          031D  012332          LXI     B,DEFN
          0320  CD750F          CALL    PUTTOK
532  4                    CALL PUTCHR(EOS);
                                                    ; STATEMENT # 532
          0323  0EFE            MVI     C,0FEH
          0325  CD320F          CALL    PUTCHR
533  4                    AP = PUSH(EP, .ARGSTK, AP);
                                                    ; STATEMENT # 533
          0328  2A1600          LHLD    EP
          032B  E5              PUSH    H       ; 1
          032C  2A1000          LHLD    AP
          032F  EB              XCHG
          0330  012336          LXI     B,ARGSTK
          0333  CDE70F          CALL    PUSH
          0336  221000          SHLD    AP
534  4                    CALL PUTTOK(.TOKEN);     /*  STACK NAME  */
                                                    ; STATEMENT # 534
          0339  01A43F          LXI     B,TOKEN
          033C  CD750F          CALL    PUTTOK
535  4                    CALL PUTCHR(EOS);
                                                    ; STATEMENT # 535
          033F  0EFE            MVI     C,0FEH
          0341  CD320F          CALL    PUTCHR
536  4                    AP = PUSH(EP, .ARGSTK, AP);
                                                    ; STATEMENT # 536
          0344  2A1600          LHLD    EP
          0347  E5              PUSH    H       ; 1
          0348  2A1000          LHLD    AP
          034B  EB              XCHG
          034C  012336          LXI     B,ARGSTK
          034F  CDE70F          CALL    PUSH
          0352  221000          SHLD    AP
537  4                    T = GETTOK(.TOKEN, MAXTOK);     /*  PEEK AT NEXT  */
                                                    ; STATEMENT # 537
          0355  1E7A            MVI     E,7AH
          0357  01A43F          LXI     B,TOKEN
          035A  CDBB0B          CALL    GETTOK
          035D  322D43          STA     T
538  4                    CALL PBSTR(.TOKEN);
                                                    ; STATEMENT # 538
          0360  01A43F          LXI     B,TOKEN
          0363  CDFA0A          CALL    PBSTR
539  4                    IF T <> LPAREN THEN     /*  ADD ()  IF NOT PRESENT  */
                                                    ; STATEMENT # 539
          0366  3A2D43          LDA     T
          0369  FE28            CPI     28H
          036B  CA7403          JZ      @73
540  4                      CALL PBSTR(.BALP);
                                                    ; STATEMENT # 540
          036E  010000          LXI     B,BALP
          0371  CDFA0A          CALL    PBSTR
                      @73:
541  4                    PLEV(CP) = 0;
                                                    ; STATEMENT # 541
          0374  2A1400          LHLD    CP
          0377  012D42          LXI     B,PLEV
          037A  09              DAD     B
          037B  3600            MVI     M,0H
542  4                  END;
                                                    ; STATEMENT # 542
                      @71:
543  3                  END /*  IF T = ALPHA ...  */;
                                                    ; STATEMENT # 543
          037D  C3FF04          JMP     @74
                      @69:
                      ELSE                                               /*  CASE 2  */
544  2                  IF T = LBRACK THEN     /*  STRIP ONE LEVEL OF  []  */
                                                    ; STATEMENT # 544
          0380  3A2D43          LDA     T
          0383  FE5B            CPI     5BH
          0385  C2DC03          JNZ     @75
545  2                    DO;
546  3                      NLB = 1;
                                                    ; STATEMENT # 546
          0388  212E43          LXI     H,NLB
          038B  3601            MVI     M,1H
547  3                    DO WHILE 1;
                                                    ; STATEMENT # 547
                      @148:
548  4                      T = GETTOK(.TOKEN, MAXTOK);
                                                    ; STATEMENT # 548
          038D  1E7A            MVI     E,7AH
          038F  01A43F          LXI     B,TOKEN
          0392  CDBB0B          CALL    GETTOK
          0395  322D43          STA     T
```

```
549  4                    IF T = LBRACK THEN
                                          ; STATEMENT # 549
          0398  3A2D43          LDA     T
          039B  FE5B            CPI     5BH
          039D  C2A703          JNZ     @76
550  4                    NLB = NLB + 1;
                                          ; STATEMENT # 550
          03A0  212E43          LXI     H,NLB
          03A3  34              INR     M
          03A4  C3D003          JMP     @77
                          @76:
                        ELSE
551  4                    IF T = RBRACK THEN
                                          ; STATEMENT # 551
          03A7  3A2D43          LDA     T
          03AA  FE5D            CPI     5DH
          03AC  C2BF03          JNZ     @78
552  4                      DO;
553  5                      NLB = NLB - 1;
                                          ; STATEMENT # 553
          03AF  212E43          LXI     H,NLB
          03B2  35              DCR     M
554  5                      IF NLB = 0 THEN
                                          ; STATEMENT # 554
          03B3  7E              MOV     A,M
          03B4  FE00            CPI     0H
          03B6  C2BC03          JNZ     @79
555  5                        GOTO QUIT;
                                          ; STATEMENT # 555
          03B9  C3D903          JMP     QUIT
                          @79:
556  5                      END;
                                          ; STATEMENT # 556
          03BC  C3D003          JMP     @80
                          @78:
                        ELSE        /*  END OF ARGUMENT LIST  */
557  4                    IF T = EOFF THEN
                                          ; STATEMENT # 557
          03BF  3A2D43          LDA     T
          03C2  FEFF            CPI     0FFH
          03C4  C2D003          JNZ     @81
558  4                      DO;
559  5                        CALL REMARK(.('*** ERROR: EOF IN STRING ',EOS));
                                          ; STATEMENT # 559
          03C7  01FC01          LXI     B,$-1CBH
          03CA  CDE306          CALL    REMARK
560  5                        GOTO RECOVER;
                                          ; STATEMENT # 560
          03CD  C34305          JMP     RECOVER
561  5                      END;
                                          ; STATEMENT # 561
                          @81:
                          @80:
                          @77:
                        CALL PUTTOK(.TOKEN);
                                          ; STATEMENT # 562
          03D0  01A43F          LXI     B,TOKEN
          03D3  CD750F          CALL    PUTTOK
563  4                    END /*  DO-WHILE  */;
                                          ; STATEMENT # 563
          03D6  C38D03          JMP     @148
                          @149:
564  3     QUIT:
                                          ; STATEMENT # 564
                          QUIT:
          03D9  C3FF04          JMP     @82
                          @75:
                        END /*  IF T = LBRACK ...  */;
                        ELSE                                      /*  CASE 3  */
565  2                  IF CP = 0FFFFH THEN     /*  NOT IN A MACRO AT ALL  */
                                          ; STATEMENT # 565
          03DC  01FFFF          LXI     B,0FFFFH
          03DF  111400          LXI     D,CP
          03E2  CD0000          CALL    @P0100
          03E5  B5              ORA     L
          03E6  C2F203          JNZ     @83
566  2                    CALL PUTTOK(.TOKEN);
                                          ; STATEMENT # 566
          03E9  01A43F          LXI     B,TOKEN
          03EC  CD750F          CALL    PUTTOK
          03EF  C3FF04          JMP     @84
                          @83:
                          ELSE                                    /*  CASE 4  */
```

```
567  2                  IF T = LPAREN THEN
                                          ; STATEMENT # 567
          03F2  3A2D43          LDA     T
          03F5  FE28            CPI     28H
          03F7  C21304          JNZ     @85
568  2                    DO;
569  3                      IF PLEV(CP) > 0 THEN
                                          ; STATEMENT # 569
          03FA  2A1400          LHLD    CP
          03FD  012D42          LXI     B,PLEV
          0400  09              DAD     B
          0401  3E00            MVI     A,0H
          0403  BE              CMP     M
          0404  D20D04          JNC     @86
570  3                        CALL PUTTOK(.TOKEN);
                                          ; STATEMENT # 570
          0407  01A43F          LXI     B,TOKEN
          040A  CD750F          CALL    PUTTOK
                          @86:
571  3                      PLEV(CP) = PLEV(CP) + 1;
                                          ; STATEMENT # 571
          040D  2A1400          LHLD    CP
          0410  012D42          LXI     B,PLEV
          0413  09              DAD     B
          0414  34              INR     M
572  3                    END;
                                          ; STATEMENT # 572
          0415  C3FF04          JMP     @87
                          @85:
                          ELSE                                    /*  CASE 5  */
573  2                  IF T = RPAREN THEN
                                          ; STATEMENT # 573
          0418  3A2D43          LDA     T
          041B  FE29            CPI     29H
          041D  C27E04          JNZ     @88
574  2                    DO;
575  3                      PLEV(CP) = PLEV(CP) - 1;
                                          ; STATEMENT # 575
          0420  2A1400          LHLD    CP
          0423  012D42          LXI     B,PLEV
          0426  09              DAD     B
          0427  35              DCR     M
576  3                      IF PLEV(CP) > 0 THEN
                                          ; STATEMENT # 576
          0428  2A1400          LHLD    CP
          042B  09              DAD     B
          042C  3E00            MVI     A,0H
          042E  BE              CMP     M
          042F  D23B04          JNC     @89
577  3                        CALL PUTTOK(.TOKEN);
                                          ; STATEMENT # 577
          0432  01A43F          LXI     B,TOKEN
          0435  CD750F          CALL    PUTTOK
          0438  C37B04          JMP     @90
                          @89:
                        ELSE  /*  END OF ARGUMENT LIST  */
578  3                      DO;
579  4                        CALL PUTCHR(EOS);
                                          ; STATEMENT # 579
          043B  0EFE            MVI     C,0FEH
          043D  CD320F          CALL    PUTCHR
580  4                        CALL EVAL( ARGSTK, CALLST(CP), AP-1);
                                          ; STATEMENT # 580
          0440  012336          LXI     B,ARGSTK
          0443  C5              PUSH    B       ; 1
          0444  2A1400          LHLD    CP
          0447  012D40          LXI     B,CALLST
          044A  29              DAD     H
          044B  09              DAD     B
          044C  E5              PUSH    H       ; 2
          044D  2A1000          LHLD    AP
          0450  2B              DCX     H
          0451  E3              XTHL            ; 2
          0452  4E              MOV     C,M
          0453  23              INX     H
          0454  46              MOV     B,M
          0455  D1              POP     D       ; 2
          0456  CDC510          CALL    EVAL
```

```
581  4                    AP = CALLST(CP);      /*  POP EVAL STACK  */
                                                ; STATEMENT # 581
     0459  2A1400         LHLD    CP
     045C  012D40         LXI     B,CALLST
     045F  29             DAD     H
     0460  09             DAD     B
     0461  5E             MOV     E,M
     0462  23             INX     H
     0463  56             MOV     D,M
     0464  EB             XCHG
     0465  221000         SHLD    AP
582  4                    EP = ARGSTK(AP);
                                                ; STATEMENT # 582
     0468  012336         LXI     B,ARGSTK
     046B  29             DAD     H
     046C  09             DAD     B
     046D  5E             MOV     E,M
     046E  23             INX     H
     046F  56             MOV     D,M
     0470  EB             XCHG
     0471  221600         SHLD    EP
583  4                    CP = CP - 1;   /*  THIS IS THE ONLY PLACE CP
                                                    GETS DECREMENTED  */
                                                ; STATEMENT # 583
     0474  2A1400         LHLD    CP
     0477  2B             DCX     H
     0478  221400         SHLD    CP
584  4                  END; /*  ELSE-DO  */
                                                ; STATEMENT # 584
                   @90:
585  3                   END; /*  IF-DO  */
                                                ; STATEMENT # 585
     047B  C3FF04           JMP     @91
                   @88:
                   ELSE                                         /*  CASE 6  */
586  2                   IF ((T = COMMA) OR (T = EQUALS)) AND (PLEV(CP) = 1) THEN
                                                        /*  NEW ARGUMENT  */
                                                ; STATEMENT # 586
     047E  3A2D43           LDA     T
     0481  D62C             SUI     2CH
     0483  D601             SUI     1
     0485  9F               SBB     A
     0486  F5               PUSH    PSW     ; 1
     0487  3A2D43           LDA     T
     048A  D63D             SUI     3DH
     048C  D601             SUI     1
     048E  9F               SBB     A
     048F  C1               POP     B       ; 1
     0490  48               MOV     C,B
     0491  B1               ORA     C
     0492  2A1400           LHLD    CP
     0495  012D42           LXI     B,PLEV
     0498  09               DAD     B
     0499  F5               PUSH    PSW     ; 1
     049A  7E               MOV     A,M
     049B  D601             SUI     1H
     049D  D601             SUI     1
     049F  9F               SBB     A
     04A0  C1               POP     B       ; 1
     04A1  48               MOV     C,B
     04A2  A1               ANA     C
     04A3  1F               RAR
     04A4  D2C904           JNC     @92
587  2                     DO;
588  3                       CALL PUTCHR(EOS);
                                                ; STATEMENT # 588
     04A7  0EFE             MVI     C,0FEH
     04A9  CD320F           CALL    PUTCHR
589  3                       AP = PUSH(EP, ARGSTK, AP);
                                                ; STATEMENT # 589
     04AC  2A1600           LHLD    EP
     04AF  E5               PUSH    H       ; 1
     04B0  2A1000           LHLD    AP
     04B3  EB               XCHG
     04B4  012336           LXI     B,ARGSTK
     04B7  CDE70F           CALL    PUSH
     04BA  221000           SHLD    AP
590  3                     END;
                                                ; STATEMENT # 590
     04BD  C3FF04           JMP     @93
                   @92:
                   ELSE                                         /*  CASE 7  */
```

```
591  2                    IF ((T = 26H) AND (BUF(BP) = 0DH) AND (BUF(BP+1) = 0AH)) THEN
                                                ; STATEMENT # 591
     04C0  3A2D43           LDA     T
     04C3  D626             SUI     26H
     04C5  D601             SUI     1
     04C7  9F               SBB     A
     04C8  2A1200           LHLD    BP
     04CB  012210           LXI     B,BUF
     04CE  09               DAD     B
     04CF  F5               PUSH    PSW     ; 1
     04D0  7E               MOV     A,M
     04D1  D60D             SUI     0DH
     04D3  D601             SUI     1
     04D5  9F               SBB     A
     04D6  C1               POP     B       ; 1
     04D7  48               MOV     C,B
     04D8  A1               ANA     C
     04D9  2A1200           LHLD    BP
     04DC  012310           LXI     B,BUF+1H
     04DF  09               DAD     B
     04E0  F5               PUSH    PSW     ; 1
     04E1  7E               MOV     A,M
     04E2  D60A             SUI     0AH
     04E4  D601             SUI     1
     04E6  9F               SBB     A
     04E7  C1               POP     B       ; 1
     04E8  48               MOV     C,B
     04E9  A1               ANA     C
     04EA  1F               RAR
     04EB  D2F904           JNC     @94
592  2                    BP = BP + 2;   /*  '&' (CONTINUATION CHARACTER)
                                                    GETS THROWN AWAY  */
                                                ; STATEMENT # 592
     04EE  2A1200           LHLD    BP
     04F1  23               INX     H
     04F2  23               INX     H
     04F3  221200           SHLD    BP
     04F6  C3FF04           JMP     @95
                   @94:
                   ELSE                                         /*  CASE 8  */
593  2                     CALL PUTTOK(.TOKEN);     /*  JUST STACK IT  */
                                                ; STATEMENT # 593
     04F9  01A43F           LXI     B,TOKEN
     04FC  CD750F           CALL    PUTTOK
                   @95:
                   @93:
                   @91:
                   @87:
                   @84:
                   @82:
                   @74:
594  2                     T = GETTOK(.TOKEN, MAXTOK);
                                                ; STATEMENT # 594
     04FF  1E7A             MVI     E,7AH
     0501  01A43F           LXI     B,TOKEN
     0504  CDBB0B           CALL    GETTOK
     0507  322D43           STA     T
595  2                   END /*  DO-WHILE  */;
                                                ; STATEMENT # 595
     050A  C3BB02           JMP     @146
                   @147:
596  1                   IF CP <> 0FFFFH THEN
                                                ; STATEMENT # 596
     050D  01FFFF           LXI     B,0FFFFH
     0510  111400           LXI     D,CP
     0513  CD0000           CALL    @P0100
     0516  B5               ORA     L
     0517  CA2305           JZ      @96
597  1                     DO;
598  2                       CALL REMARK(.('*** ERROR: UNEXPECTED EOF ',EOS));
                                                ; STATEMENT # 598
     051A  011602           LXI     B,$-304H
     051D  CDE306           CALL    REMARK
599  2                       GOTO RECOVER;
                                                ; STATEMENT # 599
     0520  C34305           JMP     RECOVER
600  2                   END;
                                                ; STATEMENT # 600
                   @96:
601  1                   IF NOT INTERACTIVE THEN
                                                ; STATEMENT # 601
     0523  3A2220           LDA     INTERACTIVE
     0526  1F               RAR
     0527  DA3505           JC      @97
```

```
602  1     LOGOFF:   CALL CLOSE(INPAFT, .STATUS);
                                          ; STATEMENT # 602
                 LOGOFF:
          052A 2A0A00        LHLD    INPAFT
          052D 44            MOV     B,H
          052E 4D            MOV     C,L
          052F 110E00        LXI     D,STATUS
          0532 CD0000        CALL    CLOSE
                 @97:
603  1           CALL CLOSE(SAVAFT, .STATUS);
                                          ; STATEMENT # 603
          0535 2A0C00        LHLD    SAVAFT
          0538 44            MOV     B,H
          0539 4D            MOV     C,L
          053A 110E00        LXI     D,STATUS
          053D CD0000        CALL    CLOSE
604  1           CALL EXIT;
                                          ; STATEMENT # 604

          /*  RECOVER TRIES TO BACK UP THE CRITICAL POINTERS TO PREVIOUSLY
              VIABLE VALUES.  IF  32  FAILURES OCCUR IN SUCCESSION THEN THESE
              POINTERS ARE SET TO PROGRAM-ENTRY INITIAL VALUES  */

          0540 CD0000        CALL    EXIT
605  1     RECOVER:  /*  NB:  THIS IS  N O T  A PROCEDURE  */
                                          ; STATEMENT # 605
                 RECOVER:
          0543 310000        LXI     SP,@STACK$ORIGIN
          0546 3A2220        LDA     INTERACTIVE
          0549 1F            RAR
          054A D27306        JNC     @98
                    IF INTERACTIVE THEN
606  1               DO;
607  2                 DECLARE (J, K) BYTE, I LITERALLY '31';

608  2                 AP = APSAV(I);    /*  TOPS OF STACKS POP TO THEIR POINTERS  */
                                          ; STATEMENT # 608
          054D 2A613E        LHLD    APSAV+3EH
          0550 221000        SHLD    AP
609  2                 CP = CPSAV(I);
                                          ; STATEMENT # 609
          0553 2AA13E        LHLD    CPSAV+3EH
          0556 221400        SHLD    CP
610  2                 EP = EPSAV(I);
                                          ; STATEMENT # 610
          0559 2AE13E        LHLD    EPSAV+3EH
          055C 221600        SHLD    EP
611  2                 PLAST = PLASTSAV(I);
                                          ; STATEMENT # 611
          055F 2A213F        LHLD    PLASTSAV+3EH
          0562 221C00        SHLD    PLAST
612  2                 TLAST = TLASTSAV(I);
                                          ; STATEMENT # 612
          0565 2A613F        LHLD    TLASTSAV+3EH
          0568 222000        SHLD    TLAST
613  2                 STACKPTR = STKPTRSAV(I);
                                          ; STATEMENT # 613
          056B 2AA13F        LHLD    STKPTRSAV+3EH
          056E F9            SPHL
614  2                 DO J = 0 TO 30;          /*  ALL VALUES MOVE UP ONE PLACE  */
                                          ; STATEMENT # 614
          056F 21F843        LXI     H,J
          0572 3600          MVI     M,0H
                 @150:
          0574 3E1E          MVI     A,1EH
          0576 21F843        LXI     H,J
          0579 BE            CMP     M
          057A DA4306        JC      @151
615  3                    K = I - J;
                                          ; STATEMENT # 615
          057D 21F843        LXI     H,J
          0580 3E1F          MVI     A,1FH
          0582 96            SUB     M
          0583 23            INX     H
          0584 77            MOV     M,A
616  3                    APSAV(K) = APSAV(K-1);
                                          ; STATEMENT # 616
          0585 3D            DCR     A
          0586 4F            MOV     C,A
          0587 0600          MVI     B,0
          0589 21233E        LXI     H,APSAV
          058C 09            DAD     B
          058D 09            DAD     B
          058E E5            PUSH    H       ; 1
          058F 2AF943        LHLD    K
          0592 2600          MVI     H,0
          0594 01233E        LXI     B,APSAV
          0597 29            DAD     H
          0598 09            DAD     B
          0599 E3            XTHL            ; 1
          059A 4E            MOV     C,M
          059B 23            INX     H
          059C 46            MOV     B,M
          059D E1            POP     H       ; 1
          059E 71            MOV     M,C
          059F 23            INX     H
          05A0 70            MOV     M,B
617  3                    CPSAV(K) = CPSAV(K-1);
                                          ; STATEMENT # 617
          05A1 3AF943        LDA     K
          05A4 3D            DCR     A
          05A5 4F            MOV     C,A
          05A6 0600          MVI     B,0
          05A8 21633E        LXI     H,CPSAV
          05AB 09            DAD     B
          05AC 09            DAD     B
          05AD E5            PUSH    H       ; 1
          05AE 2AF943        LHLD    K
          05B1 2600          MVI     H,0
          05B3 01633E        LXI     B,CPSAV
          05B6 29            DAD     H
          05B7 09            DAD     B
          05B8 E3            XTHL            ; 1
          05B9 4E            MOV     C,M
          05BA 23            INX     H
          05BB 46            MOV     B,M
          05BC E1            POP     H       ; 1
          05BD 71            MOV     M,C
          05BE 23            INX     H
          05BF 70            MOV     M,B
618  3                    EPSAV(K) = EPSAV(K-1);
                                          ; STATEMENT # 618
          05C0 3AF943        LDA     K
          05C3 3D            DCR     A
          05C4 4F            MOV     C,A
          05C5 0600          MVI     B,0
          05C7 21A33E        LXI     H,EPSAV
          05CA 09            DAD     B
          05CB 09            DAD     B
          05CC E5            PUSH    H       ; 1
          05CD 2AF943        LHLD    K
          05D0 2600          MVI     H,0
          05D2 01A33E        LXI     B,EPSAV
          05D5 29            DAD     H
          05D6 09            DAD     B
          05D7 E3            XTHL            ; 1
          05D8 4E            MOV     C,M
          05D9 23            INX     H
          05DA 46            MOV     B,M
          05DB E1            POP     H       ; 1
          05DC 71            MOV     M,C
          05DD 23            INX     H
          05DE 70            MOV     M,B
619  3                    PLASTSAV(K) = PLASTSAV(K-1);
                                          ; STATEMENT # 619
          05DF 3AF943        LDA     K
          05E2 3D            DCR     A
          05E3 4F            MOV     C,A
          05E4 0600          MVI     B,0
          05E6 21E33E        LXI     H,PLASTSAV
          05E9 09            DAD     B
          05EA 09            DAD     B
          05EB E5            PUSH    H       ; 1
          05EC 2AF943        LHLD    K
          05EF 2600          MVI     H,0
          05F1 01E33E        LXI     B,PLASTSAV
          05F4 29            DAD     H
          05F5 09            DAD     B
          05F6 E3            XTHL            ; 1
          05F7 4E            MOV     C,M
          05F8 23            INX     H
          05F9 46            MOV     B,M
          05FA E1            POP     H       ; 1
          05FB 71            MOV     M,C
          05FC 23            INX     H
          05FD 70            MOV     M,B
```

```
620  3                    TLASTSAV(K) = TLASTSAV(K-1);
                                                ; STATEMENT # 620
          05FE  3AF943          LDA     K
          0601  3D              DCR     A
          0602  4F              MOV     C,A
          0603  0600            MVI     B,0
          0605  21233F          LXI     H,TLASTSAV
          0608  09              DAD     B
          0609  09              DAD     B
          060A  E5              PUSH    H       ; 1
          060B  2AF943          LHLD    K
          060E  2600            MVI     H,0
          0610  01233F          LXI     B,TLASTSAV
          0613  29              DAD     H
          0614  09              DAD     B
          0615  E3              XTHL            ; 1
          0616  4E              MOV     C,M
          0617  23              INX     H
          0618  46              MOV     B,M
          0619  E1              POP     H       ; 1
          061A  71              MOV     M,C
          061B  23              INX     H
          061C  70              MOV     M,B
621  3                    STKPTRSAV(K) = STKPTRSAV(K-1);
                                                ; STATEMENT # 621
          061D  3AF943          LDA     K
          0620  3D              DCR     A
          0621  4F              MOV     C,A
          0622  0600            MVI     B,0
          0624  21633F          LXI     H,STKPTRSAV
          0627  09              DAD     B
          0628  09              DAD     B
          0629  E5              PUSH    H       ; 1
          062A  2AF943          LHLD    K
          062D  2600            MVI     H,0
          062F  01633F          LXI     B,STKPTRSAV
          0632  29              DAD     H
          0633  09              DAD     B
          0634  E3              XTHL            ; 1
          0635  4E              MOV     C,M
          0636  23              INX     H
          0637  46              MOV     B,M
          0638  E1              POP     H       ; 1
          0639  71              MOV     M,C
          063A  23              INX     H
          063B  70              MOV     M,B
622  3                    END;  /* DO-LOOP */
                                                ; STATEMENT # 622
          063C  21F843          LXI     H,J
          063F  34              INR     M
          0640  C27405          JNZ     @150
                          @151:

                       /*  BOTTOM OF DEQUES GET PROGRAM-ENTRY INITIAL VALUES  */

623  2                    CPSAV(0), PLASTSAV(0), TLASTSAV(0) = OFFFFH;
                                                ; STATEMENT # 623
          0643  21FFFF          LXI     H,0FFFFH
          0646  22633E          SHLD    CPSAV
          0649  22E33E          SHLD    PLASTSAV
          064C  22233F          SHLD    TLASTSAV
624  2                    APSAV(0), BP, EPSAV(0) = 0;
                                                ; STATEMENT # 624
          064F  210000          LXI     H,0H
          0652  22233E          SHLD    APSAV
          0655  221200          SHLD    BP
          0658  22A33E          SHLD    EPSAV
625  2                    STACKPTR = STKSAV0;
                                                ; STATEMENT # 625
          065B  2A1E00          LHLD    STKSAV0
          065E  F9              SPHL
626  2                    BUF(BP) = EOS;
                                                ; STATEMENT # 626
          065F  2A1200          LHLD    BP
          0662  012210          LXI     B,BUF
          0665  09              DAD     B
          0666  36FE            MVI     M,0FEH
627  2                    START = 0;
                                                ; STATEMENT # 627
          0668  21A33F          LXI     H,START
          066B  3600            MVI     M,0H
628  2                    GOTO RESTART;
                                                ; STATEMENT # 628
          066D  C3B002          JMP     RESTART
```

PROGRAMMABLE I/O CONTROLLER

News Release *Received: 77 Aug 25*

I O R announces the PIO4800, a six port programmable I/O controller. The PIO4800 is a 5 X 10 ench printed circuit board, designed to meet a wide range of user needs. All data transfer is parallel; a single input or output being accomplished by executing one processor instruction. It is compatible with the S-100 bus.

The controller will interface the computer to printers, keyboards, CRTs or any other parallel device with or without handshaking strobes. It may be operated with or without interrupts, in either the isolated I/O or in the memory-mapped mode.

The PIO4800 contains two channels with three different modes for each channel. Each channel may be programmed for up to three eight bit ports, which may be operated simultaneously. Whether a port is to be an input, an output, or a bi-directional port is determined by a control word. This control word determines the mode of each port, the direction, strobes, and interrupt capabilities.

Price of the PIO4800 kit is $149. Dealer inquiries send attention Dept. D-200.

For further information contact: I O R, Box 28823, Dallas, Texas 75228, (214) 358-2671.

EXPANDOR'S BLACK BOX PRINTER

News Release *Received: 77 Sept 23*

Expandor, Inc. has announced a printer for the computer hobbyist. The BLACK BOX Printer is a low cost ($396), fully assembled, 80 column, 10 character per second impact printer. Utilizing a print cylinder (not a dot matrix) containing a 64 ASCII character set, up to three copies are possible on tractor (or pressure) fed 8½" wide paper. The printer is shipped ready to connect to (almost) any microprocessor parallel port. The printer has a parallel interface included, requiring 7 data bits, a ready and a strobe. AC power is supplied for the printer, but the TTL logic interface requires +8 to +10 VDC from the micro.

Full documentation is supplied with the printer including trouble shooting guides, installation and maintenance instructions, printer and interface schematics, plus instructions on how to wire up to the I-0 parallel port. Expandor states that most users are able to service their own printer with the documentation supplied. However, Expandor does in-shop service, or provides parts to the user. A 90 day warranty is offered.

The BLACK BOX Printer is only 4.5" H, 13" W, & 10" D, weighing 11 lbs, which makes it easily transported. The printing mechanism is a time-proven, reliable, long life design. The printer may be pulsed character by character with the last character and line visible. The only option is the base and cover for $29.95. Otherwise, the printer is shipped complete — ready to connect and use. Detailed literature is available from Expandor, Inc., 612 Beatty Road, Monroeville, PA 15146, or by phoning (412) 373-0300.

```
629  2                    END; /* IF-DO */
                                                ; STATEMENT # 629
          0670  C37606          JMP     @99
                          @98:
                        ELSE
630  1                      GOTO LOGOFF;
                                                ; STATEMENT # 630
          0673  C32A05          JMP     LOGOFF
                          @99:
                           /*  END RECOVER  */


631  1       END KAPIAR$MODULE;
                                                ; STATEMENT # 631
          0676  FB              EI
          0677  76              HLT


MODULE INFORMATION:

     CODE AREA SIZE     = 15B9H   5561D
     VARIABLE AREA SIZE = 43FAH   17402D
     MAXIMUM STACK SIZE = 0014H     20D
     903 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

11th HOUR INFO ON 8080 PILOT

Dear Jim, *Received: 77 Nov 3*

On page 24 of the August issue there is an unfortunate publication of a letter I wrote you in May, describing what at that time I thought would be the arrangements by which those interested in PILOT could get a tape of the object code as published in *DDJ*. Prior to publication of the Guide and the assembly listing (April and May issues but actually later than the May letter), I called in the correct information and that is what was included in the original publication.

Ah well, bugs lie in wait and are not easily stamped out completely. Machine-readable versions of various types can be obtained from either:

PILOT-8080
Computer Technology Branch
Lister Hill Center, National Library of Medicine
8600 Rockville Pike
Bethesda, MD 20014
(301) 496-1647

or: The PILOT Exchange
P.O. Box 354
Palo Alto, CA 94301
(415) 326-4039

I am working with Processor Technology to create an extended SOL/CUTER version of PILOT, having cursor and tape controls, editing, etc. – close to being finished. They will distribute this version.

After 13 years of being Acting Director or Director of the Computer Center, I have returned to being a Professor full time. Please note my new address and phone.

John A. Starkweather, Ph.D.
Langley Porter Institute
University of California
San Francisco, CA 94143
(415) 681-8080, ext 464

LIGHT THRUPUT ON PAPER TAPE

Dear Jim: *Received: 77 Nov 28*

In his letter published in the *DDJ* #18, Phil Karn mentions problems regarding semi-transparent paper tape. I wonder if he has tried using polarized light. Two inexpensive pieces of polarizing material, one above the tape and one below, might solve the problem. Light would still filter through the tape, but would be de-polarized in the process and would be rejected by the second polarizer. Light passing through the holes would still be polarized and be picked up by the sensors. I haven't tried it, but it should work. (Are there people still using paper tape?)
Sincerely,

Jim Day 17042 Gunther St.
Granada Hills, CA 91344

DATA ENCRYPTION HARDWARE GETS NBS VALIDATION

news release *Received: 77 Oct 31*

A small but vital piece of hardware containing a microelectronic chip only 1 cm square has been tested and validated at the Commerce Department's National Bureau of Standards (NBS)–marking the first NBS validation of a commercial implementation of the Federal Data Encryption Standard published early this year.

The device is capable of enciphering or deciphering 64 bits of computer data in 40 microseconds. The metal oxide semiconductor device was developed by Rockwell International's Collins Group.

Other companies, among them IBM and Motorola, Inc., have announced their intention to build data encryption hardware for NBS validation under the standard, designed for use by Federal departments and agencies in protecting valuable or sensitive computer data during transmission.

Additional standards are being developed for applying data encryption to communications and data storage systems. Besides these anticipated uses, according to Dr. Dennis K. Branstad, NBS project leader for the data encryption standard, are applications in personal identification systems and in visa validation procedures. "The devices will be commercially available for use outside the government in operations requiring computer security, as in electronic fund transfers, credit card verification, and proprietary data transfer", Branstad notes.

Electronic encryption devices can be imbedded in computer terminals, communications processors, cash issuing machines and identification mechanisms. Stand-alone devices that have high physical security as well as electronic effectiveness are being built with data encryption features.

Capabilities of the NBS Computer Security Laboratory–part of the Bureau's Institute for Computer Sciences and Technology–are based on sophisticated equipment including: a data encryption standard device designed and implemented at NBS, a microprocessor, a minicomputer, and the necessary interfaces and software for testing a variety of commercial devices.

One of the validation tests consists of 12 million enciphering and deciphering operations using random data and keys. In the microprocessor, results of a commercial device can be tested against results of the NBS device. The 12 million tests can be performed in nine hours.

Upon successful completion of the tests, a validation certificate is issued to the manufacturer. The certificate allows the manufacture and sale of identical devices to Federal agencies desiring to use the standard.

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*

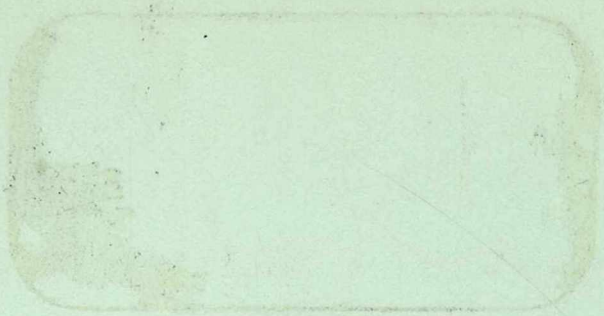
Van Buer, Darrel *14/6-7*; 30/4-11
Van Der Wateren, Frits *28/24-25*; 20/22-29
Van Dyke, Don 21/38-46
Van Valzah, Bob 10/19-32, *12/44-45*

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

5484-X
$18.95



HAYDEN BOOK COMPANY, INC.
Rochelle Park, New Jersey

ISBN 0-8104-5484-X