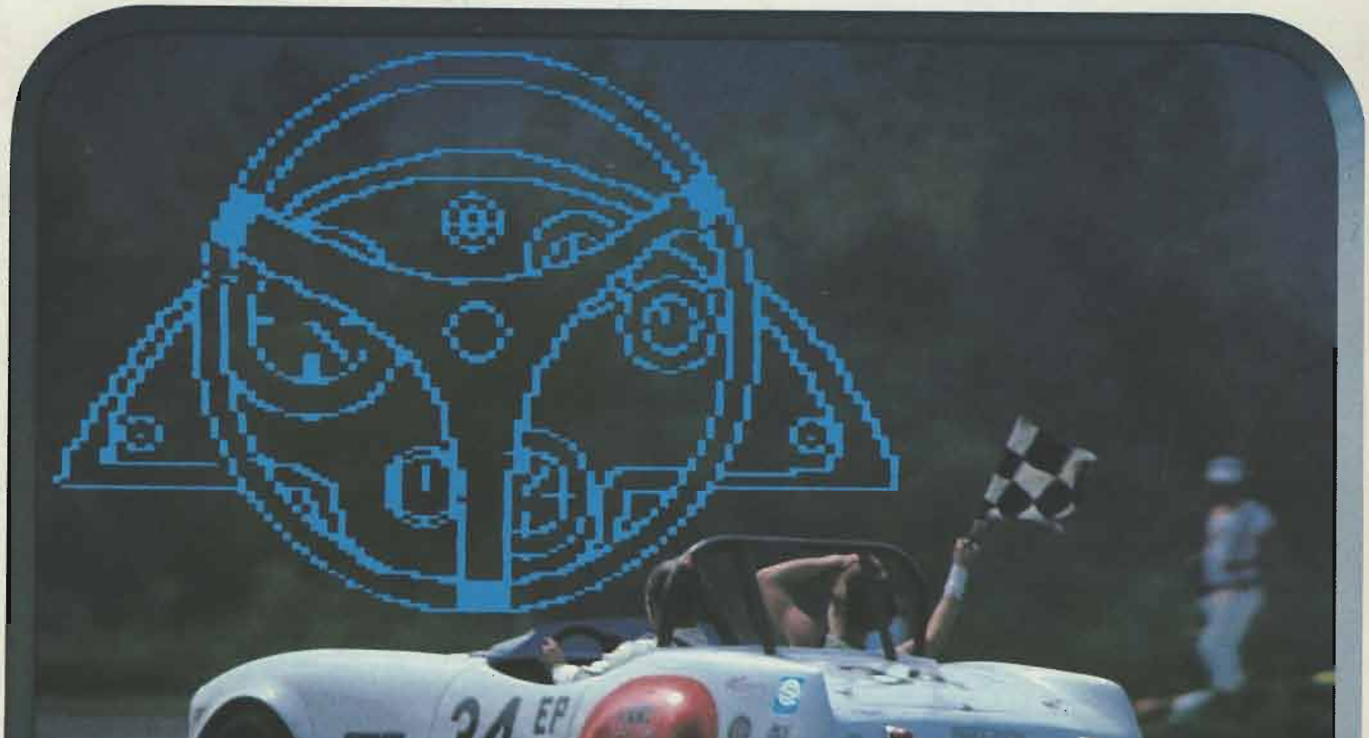
NO. 48 MAY 1982

U.S./Canada Edition: $2.50
International Edition: $2.95
United Kingdom Edition: £1.80

MICRO™

THE 6502/6809 JOURNAL



PET Feature

Growing Knowledge Trees

6502 and 6809 Memory Moves

Apple Printer Utilities

Turn your Apple into the world's most versatile personal computer.

The SoftCard™ Solution. SoftCard turns your Apple into two computers. A Z-80 and a 6502. By adding a Z-80 microprocessor and CP/M to your Apple, SoftCard turns your Apple into a CP/M based machine. That means you can access the single largest body of microcomputer software in existence. Two computers in one. And, the advantages of both.

Plug and go. The SoftCard system starts with a Z-80 based circuit card. Just plug it into any slot (except 0) of your Apple. No modifications required. SoftCard supports most of your Apple peripherals, and, in 6502-mode, your Apple is still your Apple.

CP/M for your Apple. You get CP/M on disk with the SoftCard package. It's a powerful and simple-to-use operating system. It supports more software than any other microcomputer operating system. And that's the key to the versatility of the SoftCard/Apple.

BASIC included. A powerful tool, BASIC-80 is included in the SoftCard package. Running under CP/M, ANSI Standard BASIC-80 is the most powerful microcomputer BASIC available. It includes extensive disk I/O statements, error trapping, integer variables, 16-digit precision, extensive EDIT commands and string functions, high and low-res Apple graphics, PRINT USING, CHAIN and COMMON, plus many additional commands. And, it's a BASIC you can compile with Microsoft's BASIC Compiler.

More languages. With SoftCard and CP/M, you can add Microsoft's ANSI Standard COBOL, and FORTRAN, or Basic Compiler and Assembly Language Development System. All, more powerful tools for your Apple.

Seeing is believing. See the SoftCard in operation at your Microsoft or Apple dealer. We think you'll agree that the SoftCard turns your Apple into the world's most versatile personal computer.

Complete information? It's at your dealer's now. Or, we'll send it to you and include a dealer list. Write us. Call us.

SoftCard is a trademark of Microsoft. Apple II and Apple II Plus are registered trademarks of Apple Computer. Z-80 is a registered trademark of Zilog, Inc. CP/M is a registered trademark of Digital Research, Inc.

MICROSOFT
CONSUMER PRODUCTS
MICROSOFT Inc.
10700 Northup Way • Bellevue, WA 98004

NO. 48 MAY 1982

MICRO

THE 6502/6809 JOURNAL

MACHINE-LANGUAGE AIDS

APPLE PRINTER UTILITIES

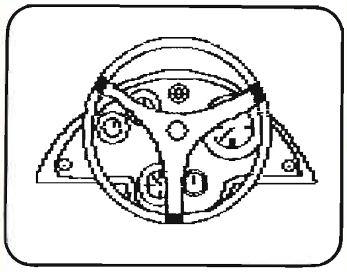
PET FEATURE

COLUMNS

DEPARTMENTS

MICRO™ Editorial

About the Cover



This month's cover gets MICRO off to a flying start at the races. Increasing software and hardware sophistication make simulations possible that put you in the driver's seat. The MIT Artificial Intelligence lab has produced a microcomputer-videodisk combination that lets you simulate a drive through Aspen, Colorado, seeing on TV just what you would see through your windshield if you were actually there.

The cover graphic was generated on an Apple Graphics Tablet, and the output was produced on an IDS Color Printer by Susan Maras at Computerland of Nashua, New Hampshire.

Cover photo: Betsey Bolton
Art Alive! Gallery
200 Merrimack St.
Lowell, MA

MICRO™ is published monthly by:
MICRO INK, Inc., Chelmsford, MA 01824
Second Class postage paid at:
Chelmsford, MA 01824 and additional mailing offices
USPS Publication Number: 483470
ISSN: 0271-9002

Send subscriptions, change of address, USPS Form 3579, requests for back issues and all other fulfillment questions to
MICRO
34 Chelmsford Street
P.O. Box 6502
Chelmsford, MA 01824
or call
617/256-5515
Telex: 955329 TLX SRVC
800-227-1617

Subscription Rates	*Per Year*
U.S.	$24.00
	2 yr. / $42.00
Foreign surface mail	$27.00
Air mail:	
Europe	$42.00
Mexico, Central America, Middle East, North Africa, Central Africa	$48.00
South America, South Africa, Far East, Australasia, New Zealand	$72.00

Present Shock

Turning on Commodore's new SuperPET is a startling experience. The system's introductory menu modestly offers access to: BASIC; Pascal; FORTRAN; APL; Assembly; and Monitor. Merely hit the appropriate key!

The development of the microcomputer is proceeding at breathtaking speed. A ferociously efficient combination of high technology, intense international competition, and ready venture capital is generating new hardware faster than society can absorb the old. How rapidly the aerospace industry progressed, we once thought, marvelling at the short span between Kitty Hawk and the Moon. But microcomputers are advancing much more swiftly. Incredibly, the arrival of the new SuperPET coincides with ongoing use of the KIM-1, a 1977 single-board system still covered by MICRO. As a special effect, such an eerie foreshortening of time belongs in a *Star Trek* episode, like Commodore's imagemaster, William Shatner.

As we admire the development of ever-more-sophisticated microcomputer hardware, we should remember that each new system requires of its users an enormous investment of time. A case in point: IBM's new Personal Computer. Announced last year to universal acclaim, the system almost entirely lacks software that exploits its 16-bit potential. While the software industry strains to fill the huge gap that appeared the day the Personal Computer was introduced, even more advanced machines are being developed.

We must learn to recognize how conservative we are with our most precious investment, time. Otherwise, costly mistakes will be hard to avoid. For example, those who promoted novel keyboard designs in hopes of replacing the standard QWERTY arrangement have convinced almost no one, the marketplace least of all. We have invested far too much time in learning QWERTY to leave it even for a significantly better keyboard design. The zeal with which microcomputer owners go on developing systems that are technological antiques should warn us that these systems, like QWERTY, will be with us for a long time to come. And as more and more new systems are introduced, sopping up more and more available time and energy, the gap between a new system's potential and the availability of resources to develop that potential seems likely to widen.

The solution to this problem is certainly not to stop building more advanced computers. The limits of the microelectronics revolution are not yet in sight, and we can look forward to ever more powerful microcomputers. What we must do is understand that the most important component in a working computer system — people — cannot fully process change at such a fast rate. Nor can we afford to discard huge investments of our time. Therefore, we must find ways to keep that investment on board. The SuperPET, for example, includes both the older 6502 and the newer 6809 processor, and can therefore run older as well as newer software. Radio Shack's recently announced Model 16 incorporates the even newer 68000 processor, yet also uses a Z-80. All older Model II software can still be used. In fact, it will even be possible to upgrade Model IIs with the 68000 board.

MICRO supports design decisions that make software compatible with different generations of a system. The same generation of people will be using many generations of computers. To stay in touch with us, the microcomputer revolution must be made compatible with the need imposed by human limits to use our time wisely.

This issue of MICRO spotlights Commodore's PET. Europe's most popular microcomputer, the PET is steadily attracting more American users. The program accompanying David Heise's feature article, "Growing Knowledge Trees," was written especially for the PET. However, the insight it offers MICRO readers into the concept of artificial intelligence makes it must reading for all.

Laurence Kepple

MICRO

Letterbox

Dear Editor:

I would strongly recommend against your readers' taking, at face value, the comments made in your December 1981 Letterbox, "Atari Ad Attacked."

Mr. Kirby does not define "adapting." If Mr. Kirby takes an Atari program and makes a "similar" program, he may end up in the Austin court he mentions in his letter. The same rules apply to a computer program as to books, etc. It is not necessary to make a 100% duplicate in order to be found guilty of copyright infringement. For example, if he were to use an unusual approach or algorithm in only a part of his new "adapted" program (assuming that the Atari program itself makes it obvious that the contents are copyrighted), the remainder of the "adapted program" could be totally different, yet a copyright infringement could easily be regarded by the court as having occurred. Mr. Kirby will then be subject to a number of possible actions, ranging from criminal penalties, damages, court injunctions... or all of them.

A helpful publication, costing about $12.00, is "The Copyright Kit," published by the National Attorney's Publications, Inc., P.O. Box 150, East Setauket, N.Y. This book explains copyrighting in layman's terms and clears up the muddy waters created by December's "intellectual property law" expert.

Stephen C. Carpenter
Mondriaanstraat 14
3262 TH Oud-Beijerland
The Netherlands

Dear Editor:

Your magazine is a very good one. My opinion might be illustrated by my collection of your issues. I started reading your magazine in late 1979. I currently have 38 issues, one reprint collection covering six more issues, and am requesting a recently missed issue. When I receive this issue, I will have access to information from 45 of the 46 issues you have placed on the market at this time.

I own an Apple II with 48K of memory, an Applesoft language card, and one DOS 3.3 disk drive. I find your coverage of the Apple to be not only very large in quantity but fine in quality. I also get a lot of ideas from the articles dealing with the other 6502/6809 machines.

Are there any plans to publish articles which describe the other CPU boards which run in the Apple? Even though you are a 6502/6809 journal, an article describing how a 6502 works with a Z80, 6800, 6809, 8088, and other chips would be very interesting.

Also, are any of your readers familiar with the new MTU 6502 machine? I recently received some literature describing it and it isn't too far from a "dream machine" itself. It appears to have hardware 18-bit addressing (yes, 18 not 16) and great bit map graphics. It also has a very sophisticated operating system.

I do have one final problem: A few months back, you had an article which described an operating system for the 6809. I believe it was OS/9 or something similar. But I don't remember seeing a manufacturer's name or address (or price for that matter). Did I simply overlook these or were they missing? Could you re-supply them? Does this operating system come in a format for the Apple II's various 6809 boards?

Larry W. Virden
1207 Rosehill Rd., Apt. 104
Reynoldsburg, OH 43068

Editor's note: The MICRO staff is very interested in hearing from readers who have experience using any of these CPU boards. Since these boards use the 6502 to handle the I/O and other functions, it would be valuable to see how the two CPU's cooperate with each other. Possible areas of coverage could include how the dual CPU's deal with cycle stealing, address translations, interrupts, parameter passing, etc.

The OS-9 operating system is available for the MILL 6809 card through Stellation Two, P.O. Box 2342, Santa Barbara, CA 93120; (805) 966-1140.

Dear Editor:

I just finished reading the March issue of MICRO. As an OSI user (I have a C2-4P MF system) I wish to thank you for your editorial "Hello, OSI?" and also for making the March issue an OSI Feature.

Let's hope that the cover photo is not a group of OSI users watching the OSI personal computer division going up in flames.

After reading the notation about the cover photo, I looked through my collection of computer manuals and found a copy of the manual prepared by Professors J.G. Kemeny and T.E. Kurtz dated June 1965. A statement of interest in the manual is: "The language that you will use is BASIC (Beginner's All-purpose Symbolic Instruction Code) which is at the same time precise, simple, and easy to understand."

J. Edward Loeffler, Jr.
Elkins Lake, Box 278
Huntsville, TX 77340

Dear Apple Owners:

In conjunction with the release of The Graphics Magician and the updated Complete Graphics System II, Penguin Software is announcing a new policy with our applications software for the Apple. The Complete Graphics System II, Special Effects, and The Graphics Magician will all now be available on non-protected disks.

We've been torn between two points of view. As computer users, we appreciate the ability to have several working copies of our applications software, and even the ability to go in and modify the code, if desired. We'd use programs such as VisiCalc or DB Master for dozens of other applications if we could have them running off several separate disks and didn't have to guard our master copies with such extreme care. Disks are fragile; we handle thousands of them, and no disk is absolutely 100% error-proof. Being programmers also, occasionally we'd like to adapt a program slightly to our system or our needs. On locked disks, much of a software product's potential usage goes untapped. *(Continued)*

Letter Box *(Continued)*

But as publishers we've been drawn into the prevailing point of view that lack of copy protection means greatly decreased sales due to casual "piracy." This is not just a crazed overreaction; we've all been to user-group meetings, homes of acquaintances, and even some computer stores, where we've been aghast at the almost encouraging attitude toward copying copyrighted software, most of which took authors months, maybe years, to perfect. The real scare here is that many of us have decided to take a risk on a very new industry and trust our livelihoods to it. Suddenly, individuals out there become statistics, some of which say that for every non-protected program sold, there are at least a dozen "pirated" copies. Those kinds of numbers could really wreak havoc on paying the bills. Scary? Yes.

From these conflicting points of view, our desire to make a good product better won, but not by much over our fear of tampering with something that is already going well. Our policies, from pricing to support, have always been very consumer-oriented. Ultimately, it is from that viewpoint that we decided to go ahead with removing the protection. We feel that you, the consumer, are entitled to software as useful as possible for the money you spend. Our hope is that the added convenience will result in more sales, not fewer, and that the software market has matured to the point where people realize that the result of illegal copying is less convenience for everyone with all software. We hope that people will think twice before accepting copies from friends, and we hope to be able to continue this policy and start a new trend toward improved usability of all applications software. Please don't abuse our trust in you.

Mark Pelczarski, President
Penguin Software
1206 Kings Circle
West Chicago, IL 60185

Dear Editor:

It would be extremely helpful if some of your readers could direct me to sources for two items: 1) a program in BASIC or machine language for OSI, Apple II, TRS-80, or PET, to score Gymnastics Meets; 2) a 16K dynamic RAM (4116) board to add to OSI Superboard II.

I have been looking for both of these for some time and have had no luck.

Bro. Felix Neussendorfer
Monasterio San Antonio Abad
Box 729
Humacao, PR 00661

If you have comments you'd like to share with MICRO's readers, why not send a letter to the editor!

Editor
MICRO
Box 6502
Chelmsford, MA 01824

Memory Map Relocator

by Preston R. Black

This short program relocates a previous memory map program. Thus, even your longest program won't be written over by the memory map program.

MEMORY MAP Relocator
requires:
Apple II

The MEMORY MAP program by Peter Cook (MICRO 36:45) is a very elegant way to assess memory usage by the Apple II computer. I have found this program to be particularly useful as a tool in program development. It gives me an up-to-date account of the size of my program and the space reserved for variable storage.

One of the drawbacks of the program is its location in memory ($3200). While this does not interfere with most of my programs, it is annoying to have to frequently reload a large program because parts of it have been written over by MEMORY MAP. This can be disastrous if the program you are developing has not been saved. Moreover, MEMORY MAP keeps reminding me that there is free space to use with my programs. Wouldn't it be great if MEMORY MAP were always loaded into that free space, leaving your program intact?

As Mr. Cook points out in his article, MEMORY MAP is not readily relocatable since there are frequent absolute references within the program. To relocate the program by hand would be most tedious. Obviously, a relocating loader for MEMORY MAP is what we need.

MEMORY MAP RELOCATOR is such a relocating loader. This program defines the free space in RAM and loads MEMORY MAP into that area. MEMORY MAP RELOCATOR then updates the relocated program to make it internally consistent and jumps to the beginning of the relocated program to run MEMORY MAP.

Memory Map Relocator

```
                ; ********************************
                ; *                              *
                ; *                              *
                ; *     MEMORY MAP RELOCATOR     *
                ; *             BY               *
                ; *     PRESTON R BLACK MD       *
                ; *          MAY 1981            *
                ; *                              *
                ; *                              *
                ; *  THIS PROGRAM WILL RELOCATE  *
                ; *    THE MEMORY MAP PROGRAM    *
                ; *   WRITTEN BY PETER A. COOK   *
                ; *   (MICRO 36:45) TO RESIDE    *
                ; *  IN RAM WHERE THERE IS FREE  *
                ; *   SPACE AND WILL NOT WRITE   *
                ; *    OVER THE BASIC PROGRAM    *
                ; *      ALREADY IN MEMORY       *
                ; *                              *
                ; ********************************
                ;
                ; EQUATES
                ;
                NADDRL EPZ $06
                NADDRH EPZ $07
                NENDL  EPZ $08
                NENDH  EPZ $09
                BUFFL  EPZ $19
                DIFF   EPZ $1B
                BUFFH  EPZ $1C
                LENGTH EPZ $2F
                PROMPT EPZ $33
                PGENDL EPZ $AF
                PGENDH EPZ $B0
                OLDADL EPZ $32
                OLDADH EPZ $39
                PGPTRL EPZ $CA
                PGPTRH EPZ $CB
                RSTPG3 EQU $9E25
                RSTRTS EQU $9E30
                INSDS2 EQU $F88E
                COUT   EQU $FDED
                ;
                       ORG $300
                ;
0300 4C1303     START  JMP INIT
0303            ;
0303            ; THIS ROUTINE RESTORES PAGE THREE DOS POINTERS
0303            ; WHICH ARE WRITTEN OVER BY THIS ROUTINE
0303            ;
0303 A960       RSTORE LDA #$60            ; PLACE 'RTS' WHERE
0305 8D309E            STA RSTRTS          ; WE WANT IT BEFORE
0308 20259E            JSR RSTPG3          ; WE JUMP TO RESTORE ROUTINE
030B            ;
030B A9AD              LDA #$AD            ; RESTORE CHANGED DOS BYTE AND
030D 8D309E            STA RSTRTS          ; JUMP TO RELOCATED MEMORY MAP
0310 6C1900            JMP (BUFFL)
0313            ;
0313            ; THE TRUE BEGINNING OF THE PROGRAM
0313            ;
0313 A533       INIT   LDA $33             ; WHICH LANGUAGE?
0315 C9BE              CMP #">"            ; INTEGER BASIC?
0317 F00C              BEQ INIT1
0319 E6B0              INC PGENDH          ; NO, THEN ADD ONE TO H.O. BYTE
031B A5B0              LDA PGENDH          ; APPLESOFT EOP POINTER
031D 8507              STA NADDRH          ; TO MAKE SURE WE ARE OVER
031F 851A              STA BUFFL+1         ; THE PROGRAM
0321 C6B0              DEC PGENDH
0323 D009              BNE INIT2
0325 38         INIT1  SEC                 ; YES, USE INTEGER BASIC
0326 A5CB              LDA PGPTRH          ; POINTERS
0328 E90C              SBC #$0C            ; ALLOW ENOUGH SPACE BELOW
032A 8507              STA NADDRH          ; PROGRAM FOR MEMORY MAP
```

(Continued)

How it Works

The first step in relocating MEMORY MAP is to define the area of free space in RAM. Both Applesoft and Integer BASIC have pointers to the end of the program stored in memory. Unfortunately, they are not the same bytes. In addition, programs are not stored the same way in the two languages. Applesoft begins storing programs at $801 and succeeding bytes are added above this. Integer BASIC begins storing programs at HIMEM and places all succeeding bytes below this. Thus, the pointers to the end of the program in the two languages must be treated differently.

For Applesoft we must load MEMORY MAP above the program already in memory. If we take the high order byte of the address of the end of the program and add one to it, we can be certain that we are above the program in memory. MEMORY MAP requires slightly less than $C00 bytes of memory if we include the area used by the printing routine. Therefore, in Integer BASIC we must go at least this far below the program to load MEMORY MAP. Otherwise we will overwrite the BASIC program already in memory.

During initialization of MEMORY MAP RELOCATOR we can determine the current language by checking which prompt is used. Appropriate adjustments must be made to the high order byte of the program end. We now have a starting address within the free space to place MEMORY MAP. Calculate the ending address of the relocated program by adding the length of MEMORY MAP to the new starting address.

Next load MEMORY MAP into the free area. This is done by constructing a string consisting of "BLOAD MEMORY MAP,A$xx00". The xx is the high order byte for the new starting address that we determined during initialization. But before we can place this number into our string, it must be converted into the ASCII representation of that number. This is done by first dividing the number into two nibbles (a nibble is equal to four bits) and converting the nibbles into the Apple ASCII code for the respective numbers. The Apple ASCII codes for the numbers from 0 to 9 are $B0 to $B9 respectively. Thus, to convert these numbers, we simply add $B0 to them. (The numbers $A to $F must have $B7 added to them to convert them into ASCII.) Once the numbers have been converted to ASCII, they are added to our string to complete it. We then use COUT ($FDED) to pass the string to DOS to be executed.

Memory Map Relocator *(Continued)*

```
032C 851A           STA BUFFL+1
032E 18      INIT2  CLC
032F 6906           ADC #$06          ; FIND END OF
0331 8509           STA NENDH         ; RELOCATED PROGRAM
0333 851C           STA BUFFH         ; AND SAVE IN BUFFERS
0335 A9E0           LDA #$E0
0337 8508           STA NENDL
0339 A900           LDA #$00
033B 8506           STA NADDRL
033D 8519           STA BUFFL
033F A507    BLOAD  LDA NADDRH        ; TAKE H.O. BYTE OF NEW START
0341 290F           AND #$0F          ; LOOK AT L.O. NIBBLE
0343 C90A           CMP #$0A          ; IS IT <10
0345 9006           BLT BLOAD1
0347 18             CLC               ; NO, CONVERT TO ASCII
0348 69B7           ADC #$B7          ; FOR 'A'-'F'
034A 4C5003         JMP BLOAD2
034D 18      BLOAD1 CLC               ; YES, CONVERT TO ASCII
034E 69B0           ADC #$B0          ; FOR '0'-'9'
0350 A201    BLOAD2 LDX #$01
0352 9DF803         STA LOAD1,X       ;STORE IN STRING
0355 A507           LDA NADDRH
0357 4A             LSR               ; NOW LOOK AT H.O. NIBBLE
0358 4A             LSR               ; AND CONVERT TO ASCII
0359 4A             LSR               ; AND STORE IN STRING
035A 4A             LSR
035B C90A           CMP #$0A
035D 9006           BLT BLOAD3
035F 18             CLC
0360 69B7           ADC #$B7
0362 4C6803         JMP BLOAD4
0365 18      BLOAD3 CLC
0366 69B0           ADC #$B0
0368 CA      BLOAD4 DEX
0369 9DF803         STA LOAD1,X
036C BDE203  BLOAD5 LDA LOAD,X        ; USE COUT TO
036F F006           BEQ UPDATE        ; PASS BLOAD COMMAND TO
0371 20EDFD         JSR COUT          ; DOS WITH RELOCATING
0374 E8             INX               ; STARTING ADDRESS
0375 D0F5           BNE BLOAD5
0377 38      UPDATE SEC               ; FIRST FIND OFFSET
0378 A507           LDA NADDRH        ; BETWEEN ORIGINAL
037A E932           SBC #OLDADL       ; AND RELOCATED PROGRAM
037C 851B           STA DIFF          ; AND SAVE
037E A000    UPDAT1 LDY #$00
0380 B106           LDA (NADDRL),Y
0382 208EF8         JSR INSDS2        ; FIND LENGTH OF
0385 A52F           LDA LENGTH        ; OP CODE
0387 A8             TAY
0388 C902           CMP #$02          ; IF < 3 THEN NEXT ONE
038A D01D           BNE UPDAT3
038C B106           LDA (NADDRL),Y    ; IF = 3 THEN
038E C932           CMP #OLDADL       ; SEE IF ADDRESS IS
0390 9017           BLT UPDAT3        ; WITHIN PROGRAM
0392 C939           CMP #OLDADH       ; BOUNDARIES
0394 B013           BGE UPDAT3        ; IF NOT, THEN
0396 C938           CMP #OLDADH-1     ; GOTO NEXT OP CODE
0398 D008           BNE UPDAT2
039A 88             DEY
039B B106           LDA (NADDRL),Y
039D C8             INY
039E C9E0           CMP #$E0
03A0 B007           BGE UPDAT3
03A2 18      UPDAT2 CLC               ; IF ADDRESS WITHIN PROGRAM
03A3 B106           LDA (NADDRL),Y    ; ADD OFFSET TO CHANGE
03A5 651B           ADC DIFF          ; TO RELOCATED ADDRESS
03A7 9106           STA (NADDRL),Y
03A9 C8      UPDAT3 INY
03AA 98             TYA
03AB 18             CLC
03AC 6506           ADC NADDRL        ; UPDATE TO NEXT
03AE 8506           STA NADDRL        ; OP CODE
03B0 9002           BCC UPDAT4
03B2 E607           INC NADDRH
03B4 38      UPDAT4 SEC
03B5 A508           LDA NENDL         ; END OF THE
03B7 E506           SBC NADDRL        ; RELOCATED PROGRAM?
03B9 A509           LDA NENDH
03BB E507           SBC NADDRH
03BD 9003           BCC UPDAT5
03BF 4C7E03         JMP UPDAT1        ; NO, CONTINUE UPDATE
03C2 A51C    UPDAT5 LDA BUFFH         ; YES, REPLACE $6F
03C4 A06F           LDY #$6F          ; WITH PROPER ADDRESS
03C6 9119           STA (BUFFL),Y
03C8         ;
03C8         ; NOW CORRECT THE ADDRESSES OF THE STARTING PAGES
03C8         ; FOR PRINTING THE MOVED TEXT PAGE
03C8         ;
03C8 A51C           LDA BUFFH
03CA 8509           STA NENDH
03CC A900           LDA #$00
03CE 8508           STA NENDL
```

Memory Map Relocator *(Continued)*

```
03D0 A0B3           LDY #$B3
03D2 18      UPDAT6 CLC
03D3 B108           LDA (NENDL),Y
03D5 651B           ADC DIFF
03D7 9108           STA (NENDL),Y
03D9 C8             INY
03DA C8             INY
03DB C0E0           CPY #$E0
03DD 90F3           BLT UPDAT6
03DF                ;
03DF                ; WHEN FINISHED RESTORE PAGE THREE POINTERS
03DF                ; BEFORE RUNNING RELOCATED MEMORY MAP
03DF                ;
03DF 4C0303         JMP RSTORE
03E2 8D8D84  LOAD   HEX 8D8D84
03E5 C2CCCF         ASC "BLOAD MEMORY MAP,A$"
03E8 C1C4A0
03EB CDC5CD
03EE CFD2D9
03F1 A0CDC1
03F4 D0ACC1
03F7 A4
03F8 0000B0  LOAD1  HEX 0000B0B08D00
03FB B08D00
03FE         LENTH  EQU *-START
```

Once the program has been loaded into memory, we must update it to make internal calls consistent. The algorithm for this is as follows: First, the offset between the original program and the relocated program is calculated. This is the amount that must be added to the original addresses to make them compatible with the relocated program. Using the monitor routine INSDS2, we determine how many bytes are used by each op code. If the op code requires only one or two bytes, then any addressing will be relative and will not require updating. If, however, the op code is three bytes long, then all addresses used must be absolute.

We must also check to see if that address is within the boundaries of the original program (i.e., from $3200 to $38E0). If it is, then we add the offset to the high order byte. If it is not, we go to the next op code. We continue in this fashion until we reach the end of the relocated program. When the relocated program has been completely updated, an indirect jump to the beginning of the relocated program will run MEMORY MAP.

How to Use the Program

MEMORY MAP RELOCATOR resides on page three of memory. Since it is longer than $D0 bytes long, it overwrites important DOS vectors located on page three. To insure proper function of DOS after the program is run, a short routine to restore these pointers begins the program. It is placed at the beginning so it will not be destroyed while the pointers are restored.

The routine to restore the pointers makes use of the part of DOS which places the pointers onto page three during the bootstrap. I place an 'RTS' ($60) in the place that suits my purposes and restore the byte to what it was before performing the indirect jump to run MEMORY MAP.

Once the program has been entered and saved, BRUNning it will place MEMORY MAP into the available free space and run it. Remember that this program is written to run with a program named MEMORY MAP which is normally stored from $3200 to $32E0. With minor modifications, this program can be converted to run with a program beginning at any address, and of any length.

Please contact Mr. Black at 16 Durham St., Boston, MA 02115.

MICRO

BASIC to Machine Language Interface

by Christer Engstrom

Since the AIM lacks a SYS or CALL function, it is difficult to communicate between BASIC and machine language. This interface routine makes the USR(W) function more flexible and allows entry of the machine language address directly in hexadecimal through the BASIC variable AD%.

Interface
requires:

AIM with 4K
BASIC
AIM Assembler

I own an AIM with 4K RAM and BASIC and assembler ROMs. The BASIC interpreter is slow, but the machine itself is very fast. Furthermore, there is a lack of cooperation between BASIC and machine language. The only way to jump out from BASIC to execute other code in memory, is to use the USR(W) command. This command lets you jump to a subroutine whose address is given in locations \$04 and \$05, and also pass a floating point value in locations \$B1 - \$B6.

A frequent use of this method results in many POKEs, which are done byte-by-byte in decimal. This is not good for readability. I think we need an easier way to jump out, a better way to define where to jump, and a method to pass parameters. We need a general interface.

The best way to define an address is with (ASCII) hex characters. Here's my solution to the problem: every time BASIC executes the USR(W) command, the machine enters the general interface. The integer AD% is supposed to contain the address to jump to, in high-low order. The interface scans the variable table and searches for AD% (which in the machine is interpreted as \$C1C4 — ASCII of 'A' and 'D', each ORed with \$80). If it is found and contains an address > \$00FF, the interface converts the value W specified in the USR(W) statement, to a signed binary value in locations \$AC to \$AD (subroutine \$BEFE).

Next it loads the byte at location \$AD (LSB of the value) into the accumulator, and jumps to the subroutine. On returning from the subroutine, the accumulator is stored into \$AD, and the signed binary value in \$AC-\$AD is converted back to the floating-point register (subroutine \$C0D). Finally, a return to BASIC is made.

Examples

1. You want to jump to a subroutine at location \$0400.

```
10 AD% = X"0400"
20 L = USR(0)
```

2. Take advantage of the monitor routine at location \$E97A. Don't forget that the accumulator must be loaded with a value:

```
10 AD% = X"E97A"
20 L = USR(A)
```

I know, you're thinking that the X"0400" and the X"E97A" are not conforming to general BASIC syntax. But AD% = 1024 and AD% = −576 are! So what we now need is a way to translate all X".." expressions to their decimal equivalents before execution. That is done by the hex converter. If the general interface doesn't find AD% or if AD% is zero, all X".." and Z".." expressions are converted to decimal.

You can see that the hex converter is entered *via* the general interface. This means that it is easy to modify the interface so that it can execute more functions (with a function code in AD%). You may even want to modify the whole interface. Maybe it is better when used this way:

```
10 AD$ = "E97A"
20 L = USR(A)
```

Let's get back to the hex converter. If you want to assign an unsigned value > \$7FFF to a BASIC Integer, you must consider this: the interval \$8000 to \$FFFF equals the decimal interval −32768 to −1. This means that \$8000 = −(\$10000 - \$8000) = −(65536 -32768) = −32768.

We don't always want to translate the hex string to a signed value, so another type must be defined. This leads us to two different syntaces. To get a signed decimal value, precede the hex string in quotes with an X. For positive (unsigned) values, use a Z instead.

Example 3:

```
X"9000" = −(65536 - 36864)
        = −28672 BUT
Z"9000" = 36864
```

If you define an address, use the X type. Note that only the program part, not the variable part, is hex-converted. Also note that the string within the quotes must consist only of the hex characters (0 - 9, A - F), and have a length of 0 - 4 characters. Left-fill with zeroes is done automatically. For example:

```
10 L = USR(X"A")
20 A = X"CC"/B
```

After hex conversion,

```
10 L = USR(0010)
20 A = 00204/B
```

no compression is done. The string beginning with X or Z is replaced by a decimal value of the same length. If the hex string is not enclosed within quotation marks, BASIC will attempt to interpret some strings to function codes during input phase; "DEF" for example.

Program Description: The Interface

In my version, the interface consists of two parts: the interface and the hex converter. Since only relative branches are made, both parts are relocatable. The interface and the hex conversion need not cooperate — simply remove one of them. In my version, the interface must know the real start addresses of functions it should handle. The BASIC input buffer ($14 - $50) is used as a work area. The interface starts with a lookup of the variable table.

Here are some valuable points:

1. The start address for the table is in locations $75 and $76.
2. An integer variable name has $80 added to the first and second character of its name (thus making AD% = $C1C4, not $4144).
3. Every entry in the table consists of seven bytes, the first two for its name.
4. An integer variable has its value (signed) in the two bytes following the name.
5. "End-of-table" is flagged by $AA in the first location of an entry.

Here is a description of what is done at each label:

ENTRY—Stores the address of the variable table start in a work area.

SCAN—A search for $C1C4 or end-of-table is done.

NEW—Get address for next entry, scan again.

CHECK—Tests value in leftmost byte of integer AD%.

JUMP—Jumps to subroutine after floating point conversion. At return, converts integer back to floating point array.

OUT—Clears A,X,Y registers and returns back to BASIC.

FUNC—Jumps to functions by testing the rightmost byte of AD%. Invalid functions are ignored.

FUNC..—The functions.

Hex Converter

This is a fairly long and space-consuming part; the readability is worth more than smart programming.

1. A scan for all strings beginning with X" and Z" is done in the program part (page "CHECK PROG").
2. If such a string is found, and the rest of it conforms to the above given syntax, it is converted to the decimal equivalent (pages "MOVE RIGHT JUST" and "STRING TO HEX").
3. If it is an X" string with a value > = 32768, a sign-conversion (see example 3) is done (page "SIGN CONVERSION").
4. Finally, the converted value is edited back to the program (pages "EDIT NOW" and "MOVE TO PROG").
5. When the program part is scanned through, a return to the interface is done (page "MAIN LOOP").

Conclusion

We now have an extended and more flexible way to use BASIC with the rest of the machine. Even some monitor routines can be used without specially written routines. The hex converter allows us to specify constants in hexadecimal mode. This method also cooperates better with the rest of the machine. Finally, the interface can help during the editing of a program (function codes could be used).

Please address the author at
Gotlandsgatan, 71A, 2½ TR, S-11638,
Stockholm, Sweden.

***Listing 1:* The Interface**

```
                     ;************************
                     ;*                      *
                     ;*   BASIC INTERFACE    *
                     ;*                      *
                     ;* BY CHRISTER ENGSTROM *
                     ;*                      *
                     ;************************
                     ;
                     ;       ORG $F81
0F81 A675            ENTRY   LDX $75
0F83 8614                    STX $14
0F85 A676                    LDX $76
0F87 8615                    STX $15
0F89 A000            SCAN    LDY #$00
0F8B B114                    LDA ($14),Y
0F8D C9AA                    CMP #$AA
0F8F F051                    BEQ FUNC0
0F91 C9C1                    CMP #$C1
0F93 D007                    BNE NEW
0F95 C8                      INY
0F96 B114                    LDA ($14),Y
0F98 C9C4                    CMP #$C4
0F9A F010                    BEQ CHECK
0F9C 18              NEW     CLC
0F9D A514                    LDA $14
0F9F 6907                    ADC #$07
0FA1 8514                    STA $14
0FA3 A515                    LDA $15
0FA5 6900                    ADC #$00
0FA7 8515                    STA $15
0FA9 18                      CLC
0FAA 90DD                    BCC SCAN
0FAC C8              CHECK   INY
0FAD B114                    LDA ($14),Y
0FAF F025                    BEQ FUNC
0FB1 20FEBE          JUMP    JSR $BEFE
0FB4 A94C                    LDA #$4C
0FB6 8516                    STA $16
0FB8 A002                    LDY #$02
0FBA B114                    LDA ($14),Y
0FBC 8518                    STA $18
0FBE C8                      INY
0FBF B114                    LDA ($14),Y
0FC1 8517                    STA $17
0FC3 A5AD                    LDA $AD
0FC5 201600                  JSR $0016
0FC8 85AD                    STA $AD
0FCA A5AC                    LDA $AC
0FCC A4AD                    LDY $AD
0FCE 20D1C0                  JSR $C0D1
0FD1 A900            OUT     LDA #$00
0FD3 A8                      TAY
0FD4 AA                      TAX
0FD5 60                      RTS
0FD6 C8              FUNC    INY
0FD7 B114                    LDA ($14),Y
0FD9 C900                    CMP #$00
0FDB F005                    BEQ FUNC0
0FDD C901                    CMP #$01
0FDF F007                    BEQ FUNC1
0FE1 60                      RTS
0FE2 20000E          FUNC0   JSR $0E00
0FE5 18                      CLC
0FE6 90E9                    BCC OUT
0FE8 EA              FUNC1   NOP
0FE9 18                      CLC
0FEA 90E5                    BCC OUT
```

***Listing 2:* Hexadecimal Converter**

```
                     ;************************
                     ;*                      *
                     ;*    HEX CONVERTER     *
                     ;*                      *
                     ;* BY CHRISTER ENGSTROM *
                     ;*                      *
                     ;************************
                             ORG $0E00
0E00 A573            START   LDA $73
0E02 8514                    STA $14
0E04 A574                    LDA $74
0E06 8515                    STA $15
0E08                 ;
0E08                 ;START LOOP IN PROG
0E08                 ;
0E08 A515            LOOP    LDA $15
0E0A C576                    CMP $76
0E0C 3007                    BMI GO
0E0E A514                    LDA $14
0E10 C575                    CMP $75
0E12 3001                    BMI GO
0E14 60                      RTS
0E15                 ;
0E15 20280E          GO      JSR SUBR
0E18 18                      CLC
0E19 A514                    LDA $14
0E1B 6901                    ADC #$01
0E1D 8514                    STA $14
0E1F A515                    LDA $15
0E21 6900                    ADC #$00
0E23 8515                    STA $15
0E25 18                      CLC
0E26 90E0                    BCC LOOP
0E28                 ;
0E28                 ;CHECK PROGRAM
0E28                 ;
0E28 D8              SUBR    CLD
0E29 A000                    LDY #$00
0E2B A958                    LDA 'X
0E2D D114                    CMP ($14),Y
0E2F F006                    BEQ CONT
0E31 A95A                    LDA 'Z
0E33 D114                    CMP ($14),Y
0E35 D037                    BNE NEXT
0E37                 ;
0E37 8516            CONT    STA $16
0E39 A922                    LDA '"
0E3B C8                      INY
0E3C D114                    CMP ($14),Y
0E3E D02E                    BNE NEXT
0E40 A230                    LDX '0
0E42 8618                    STX $18
0E44 8619                    STX $19
0E46 861A                    STX $1A
0E48 861B                    STX $1B
0E4A A200                    LDX #$00
0E4C 8634                    STX $34
0E4E A218                    LDX #$18
0E50 8633                    STX $33
0E52 EA                      NOP
0E53                 ;
0E53 C8              IN      INY
0E54 B114                    LDA ($14),Y
0E56 C922                    CMP '"
0E58 F015                    BEQ UT
0E5A C006                    CPY #$06
0E5C F010                    BEQ NEXT
```

(Continued)

Listing 2 *(Continued)*

```
0E5E C930              CMP '0
0E60 300C              BMI NEXT
0E62 C93A              CMP #$3A
0E64 30ED              BMI IN
0E66 C941              CMP 'A
0E68 3004              BMI NEXT
0E6A C947              CMP 'G
0E6C 30E5              BMI IN
0E6E              ;INVALID/NO STRING
0E6E 60           NEXT  RTS
0E6F              ;
0E6F              ;MOVE RIGHT JUST
0E6F              ;
0E6F 8417         UT    STY $17
0E71 A203               LDX #$03
0E73 8636               STX $36
0E75 8435               STY $35
0E77 C635         IN2   DEC $35
0E79 A435               LDY $35
0E7B B114               LDA ($14),Y
0E7D C922               CMP '"
0E7F F009               BEQ UT2
0E81 A436               LDY $36
0E83 9133               STA ($33),Y
0E85 C636               DEC $36
0E87 18                 CLC
0E88 90ED               BCC IN2
0E8A              ;
0E8A              ;STRING TO HEX
0E8A              ;
0E8A A003         UT2   LDY #$03
0E8C B133         UT2A  LDA ($33),Y
0E8E 207DEA             JSR $EA7D
0E91 9133               STA ($33),Y
0E93 88                 DEY
```


Listing 2 *(Continued)*

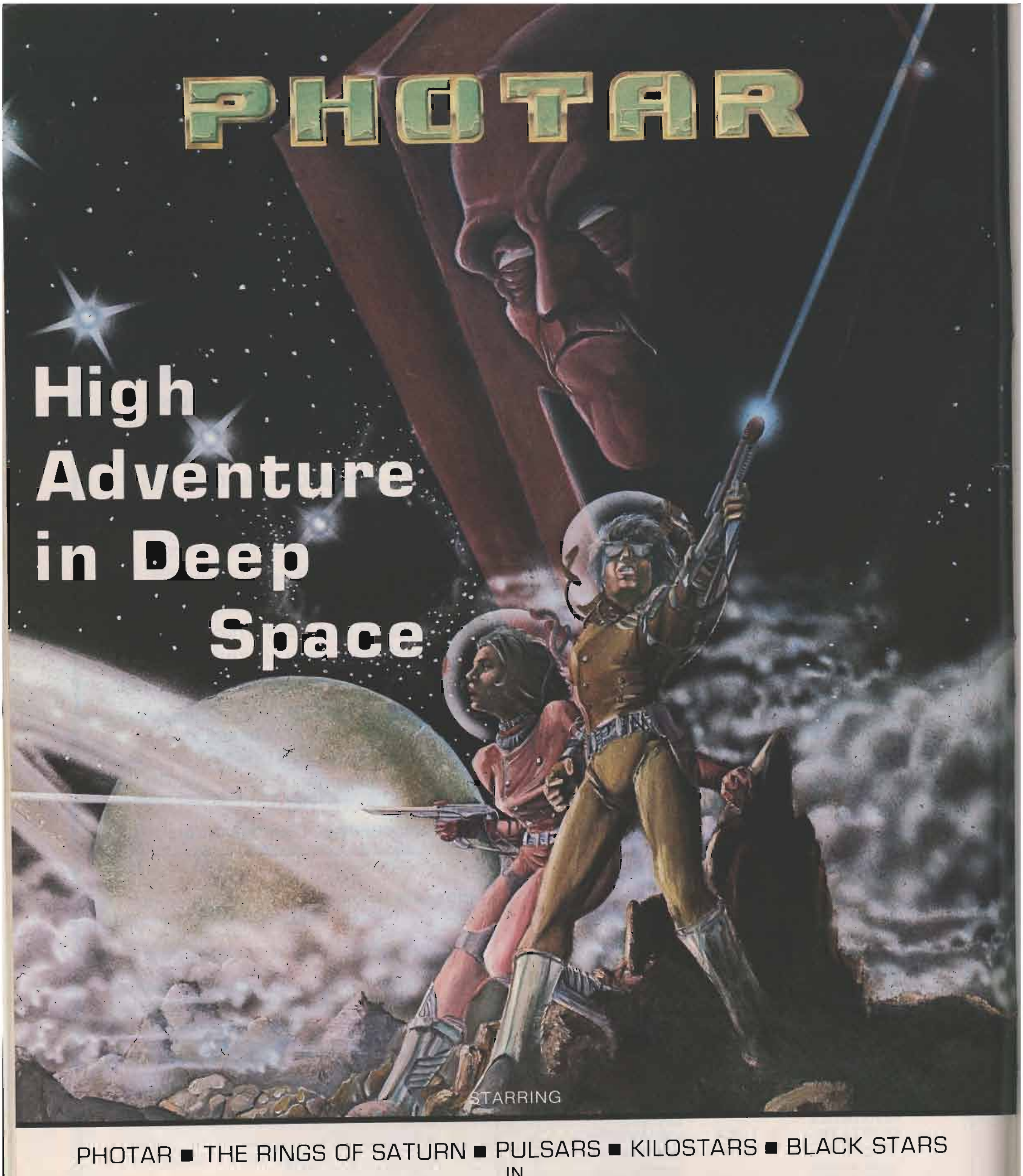
```
0E94 10F6               BPL UT2A
0E96              ;START COUNT LOOP
0E96 A200               LDX #$00
0E98 8622               STX $22
0E9A 8623               STX $23
0E9C 8624               STX $24
0E9E F8                 SED
0E9F A940               LDA #$40
0EA1 8525               STA $25
0EA3 A996               LDA #$96
0EA5 8526               STA $26
0EA7 A518               LDA $18
0EA9              ;TEST BIT 1
0EA9 C908               CMP #$08
0EAB 300D               BMI NOMI
0EAD A516               LDA $16
0EAF C958               CMP 'X
0EB1 D007               BNE NOMI
0EB3 A9A5               LDA #$A5
0EB5              ;FLAG < 0
0EB5 8516               STA $16
0EB7 18                 CLC
0EB8 9004               BCC IN3
0EBA A930         NOMI  LDA '0
0EBC              ;FLAG >= 0
0EBC 8516               STA $16
0EBE C618         IN3   DEC $18
0EC0 3005               BMI UT3
0EC2 20F60E             JSR ADD
0EC5 90F7               BCC IN3
0EC7 A902         UT3   LDA #$02
0EC9 8525               STA $25
0ECB A956               LDA #$56
0ECD 8526               STA $26
0ECF C619         IN4   DEC $19
0ED1 3005               BMI UT4
0ED3 20F60E             JSR ADD
0ED6 90F7               BCC IN4
0ED8 A900         UT4   LDA #$00
0EDA 8525               STA $25
0EDC A916               LDA #$16
0EDE 8526               STA $26
0EE0 C61A         IN5   DEC $1A
0EE2 3005               BMI UT5
0EE4 20F60E             JSR ADD
0EE7 90F7               BCC IN5
0EE9 A901         UT5   LDA #$01
0EEB 8526               STA $26
0EED C61B         IN6   DEC $1B
0EEF 301A               BMI UT6
0EF1 20F60E             JSR ADD
0EF4 90F7               BCC IN6
0EF6              ;ADD TO RESULT
0EF6 18           ADD   CLC
                  ;ADD TO RESULT
0EF7 A524               LDA $24
0EF9 6526               ADC $26
0EFB 8524               STA $24
0EFD A523               LDA $23
0EFF 6525               ADC $25
0F01 8523               STA $23
0F03 A522               LDA $22
0F05 6900               ADC #$00
0F07 8522               STA $22
0F09 18                 CLC
0F0A 60                 RTS
0F0B              ;
0F0B              ;SIGN CONVERSION
```

(Continued)

Listing 2 *(Continued)*

```
0F0B                  ;
0F0B A516             UT6    LDA $16
0F0D                  ;CHECK MINUS
0F0D C9A5                    CMP #$A5
0F0F D013                    BNE IN7
0F11 38                      SEC
0F12 A936                    LDA #$36
0F14 E524                    SBC $24
0F16 8524                    STA $24
0F18 A955                    LDA #$55
0F1A E523                    SBC $23
0F1C 8523                    STA $23
0F1E A906                    LDA #$06
0F20 E522                    SBC $22
0F22 8522                    STA $22
0F24                  ;
0F24                  ;EDIT NOW
0F24                  ;
0F24 D8               IN7    CLD
0F25 EA                      NOP
0F26 A516                    LDA $16
0F28 8535                    STA $35
0F2A A522                    LDA $22
0F2C 20560F                  JSR LEFT
0F2F 8536                    STA $36
0F31 A522                    LDA $22
0F33 205C0F                  JSR RIGHT
0F36 8537                    STA $37
0F38 A523                    LDA $23
0F3A 20560F                  JSR LEFT
0F3D 8538                    STA $38
0F3F A523                    LDA $23
0F41 205C0F                  JSR RIGHT
0F44 8539                    STA $39
0F46 A524                    LDA $24
0F48 20560F                  JSR LEFT
0F4B 853A                    STA $3A
0F4D A524                    LDA $24
0F4F 205C0F                  JSR RIGHT
0F52 853B                    STA $3B
0F54 9014                    BCC UT7
0F56 29F0             LEFT   AND #$F0
0F58 6A                      ROR
0F59 6A                      ROR
0F5A 6A                      ROR
0F5B 6A                      ROR
0F5C 290F             RIGHT  AND #$0F
0F5E 18                      CLC
0F5F 6930                    ADC '0
0F61 C93A                    CMP #$3A
0F63 1001                    BPL MORE
0F65 60                      RTS
0F66 6907             MORE   ADC #$07
0F68 18                      CLC
0F69 60                      RTS
0F6A                  ;
0F6A                  ;MOVE TO PROG
0F6A                  ;
0F6A A935             UT7    LDA #$35
0F6C 8533                    STA $33
0F6E A006                    LDY #$06
0F70 8416                    STY $16
0F72 B133             IN8    LDA ($33),Y
0F74 C616                    DEC $16
0F76 A417                    LDY $17
0F78 9114                    STA ($14),Y
0F7A A416                    LDY $16
0F7C C617                    DEC $17
0F7E 10F2                    BPL IN8
0F80 60                      RTS
```

MICRO


PHOTAR
High Adventure in Deep Space
STARRING

Memory Moves with the 6502 and 6809

by Gregory Walker and Tom Whiteside

The authors demonstrate the advantages of the 6809's direct page addressing and 16-bit index registers with a comparison of 6502 and 6809 memory moves.

In a previous article (MICRO 47:57) we illustrated the advantages of programming the 6809 over the 6502 with a comparison of multiprecision arithmetic routines. We continue in this article with a comparison of the two processors' capabilities in solving memory move problems.

With these two articles, we hope to demonstrate that the MC6809 is not only faster and more byte-efficient than the 6502, but also more straightforward to program. Because the most useful kinds of 6502 indirect addressing must be done through page zero, it is important, particularly with larger operating systems, that page-zero RAM be used wisely. In addition, the 6502 index registers are only eight bits long, limiting indexing to a 256-byte range. These limitations will show themselves especially well in these memory move examples.

Memory moves have a number of practical applications, including word processing, EPROM programming, and program relocation. Similar techniques are involved with string manipulation and table searches.

Figure 1 is a 6502 memory move for fewer than 256 bytes from a fixed absolute address. This routine is not all that useful, since it can only work on two fixed pages due to the limited range of the 6502 index registers. However, it illustrates the real power of the 6502 in terms of byte efficiency and speed over *small* ranges of memory.

The program uses the fastest form of 6502 indexed addressing — absolute indexed. The Y register will be used both as a loop index for the move and as a counter for the number of bytes to be moved. The Y register is initialized to the number of bytes to be moved and is decremented each time through the loop. When the Y register decrements to zero, the branch conditions are not met and the loop terminates. This use of the Y register eliminates the need for a CPY immediate instruction in the loop and speeds up the code. A "CNT" value of zero will move 256 bytes.

In these examples, the "LNG" column in figure 1 represents the number of bytes required per instruction. The "TIM" column is the number of machine cycles per instruction. The 6502 memory move for fewer than 256 bytes of memory required only 11 bytes of code and approximately 14 machine cycles per byte moved.

Figure 2 shows the same memory move written in MC6809 code. In this example, the 16-bit X register points to the "FROM" address and the U register points at the "TO" address. The MC6809 addressing mode used is indexed with accumulator offset. The effective address is formed by summing the two's complement contents of the B accumulator with the contents of the index register used. You will notice that the B accumulator is being used in the same manner as the 6502 Y register was in figure 3. Because the offset is two's complement, the MC6809 example is limited to 127 bytes. We included this example to show how similarly the two processors can be used to solve the same problem. The MC6809 took 15 bytes and 15 machine cycles per byte moved.

While the 6502 wins this round by four bytes and one machine cycle per

***Figure 1:* 6502 program to move fewer than 256 bytes of memory. Timing = 2 + 14 * N where N is the number of bytes to move.**

```
                           (LNG TIM)
         LDY #CNT            2   2    INITIALIZE THE BYTES TO MOVE
                                      COUNT

LOOP     LDA FROM-1, Y       3   4    LOOP: GET BYTE TO MOVE
         STA TO-1, Y         3   5      MOVE BYTE
         DEY                 1   2      DECREMENT LOOP COUNTER
         BNE LOOP            2   3      LOOP UNTIL ZERO COUNT
                            11
```

***Figure 2:* MC6809 program to move fewer than 128 bytes of memory. Timing = 8 + 15 * N.**

```
         LDX #FROM-1         3   3    INITIALIZE "FROM" POINTER
         LDU #TO-1           3   3    INITIALIZE "TO" POINTER
         LDB #CNT            2   2    INITIALIZE BYTES TO MOVE
                                      COUNT

LOOP     LDA B, X            2   5    LOOP: GET BYTE
         STA B, U            2   5      AND MOVE IT
         DECB                1   2      DECREMENT LOOP COUNT
         BNE LOOP            2   3      LOOP UNTIL COUNT IS ZERO
                            15
```

byte over the MC6809, the MC6809 code is more versatile. If this were a subroutine, "LOOP" could be called with X and U pointing anywhere in memory, while the 6502 example would be limited to the 256-byte range of its index registers. Because the MC6809 code holds the pointers in registers instead of memory locations, it is re-entrant and could be used in a real-time operating system.

Since it appears that the 6502 can keep up with the MC6809 on a short memory move, let's try another memory move. This time the code must be able to move *any* number of bytes. A real measure of a processor's power is how much its performance degrades as the complexity of its task increases. In this example, complexity is measured in terms of address range.

Figure 3 shows a 6502 program to move any size block of memory. "CNT" bytes will be moved from address "FROM" to address "TO". The bytes will be moved starting at address "FROM" plus "CNT," with "CNT" decremented each time through the loop. Since the 6502 index registers are only eight bits wide, it is necessary to use indirect indexed addressing to move more than 256 bytes. (We do not count self-modifying code as an option, but as an abomination!)

The first part of the program sets up the two 16-bit zero-page pointers "FROM" and "TO". "CNT" is a 16-bit number stored in the X register (least significant byte) and "COUNT" (most significant byte). The X register is used to store the least significant byte of "CNT" to save three machine cycles per byte moved over using a zero-page variable. The "CNT" most significant byte is initialized to one count higher than desired to eliminate the need for a load/compare step that would have used time and bytes. The pointer least significant bytes are incremented rather than the Y register, since the "FROM" and "TO" least significant bytes might not be the same.

If the above "tricks" seem confusing to you, you are not alone. Tricks cost money in terms of debug time and the time required to recode the routine when some performance requirement is changed. However, we tried to write the best 6502 code possible. The result is typical of commercial practice. Even with all these tricks, the 6502 code was 47 bytes long and approximately 32 machine cycles per byte. This is more than four times the size, and twice the cycles per byte over the short memory move. Now you see what we meant about performance degradation with increased task complexity!

Figures 4 and 5 show two versions of MC6809 memory moves which can handle memory ranges of more than 256 bytes. Both examples are identical except that one (figure 4) moves memory one byte at a time while the other (figure 5) moves two bytes at once. Both require 18 bytes of code, but the second program is eight machine cycles per byte (40%!) faster than the first.

In both these programs, the X register acts as a pointer to the "FROM" address and the U register acts as a pointer to the "TO" address, just like the program in figure 2. In these routines, however, the index registers are incremented each time through the loop as indicated by the "+" beside the indexed loads and stores. Since the second program moves two bytes at a time, the MC6809 double increment ("++") mode is used to advance to the next word. In both figures, the Y register is a counter to the number of bytes remaining to be counted. The "LEAY" instruction has no 6502 equivalent and indicates that the Y register is to be loaded with the "effective" address indicated in the operand field. In figure 4, the operand field of the LEAY instruction means to load Y with the contents of Y minus 1 like a 16-bit 6502 "DEY".

***Figure 3:* 6502 program to move any number of bytes of memory. Timing = 31 + (35 + 28/256) * N.**

```
                          (LNG  TIM)
        LDA #FROML          2    2     INITIALIZE INDIRECT "FROM" PTR
        STA FROM            2    4
        LDA #FROMH          2    2
        STA FROM+1          2    4
        LDA #TOL            2    2     INITIALIZE INDIRECT "TO" PTR
        STA TO              2    3
        LDA #TOH            2    2
        STA TO+1            2    3
        LDA #CNTH+1         2    2     INIT BYTES TO MOVE COUNT MSB
        STA COUNT           2    3       TO (COUNT / 256) + 1
        LDX #CNTL+1         2    2     INIT X TO THE COUNT LSB
        LDY #0              2    2     INITIALIZE INDIRECT POINTER

LOOP    LDA (FROM), Y       2    5     LOOP: GET A BYTE
        STA (TO), Y         2    6       AND MOVE IT
        INC TO              2    5       INCREMENT 16-BIT "TO"
        BNE NOINC1          2    4       POINTER
        INC TO+1            2    5
NOINC1  INC FROM            2    5       INCREMENT 16-BIT "FROM"
        BNE NOINC2          2    4       POINTER
        INC FROM+1          2    5
NIINC2  DEX                 1    2       DECREMENT 16-BIT "CNT"
        BNE LOOP            2    4
        DEC COUNT           2    5
        BNE LOOP            2    4     LOOP UNTIL "CNT" IS ZERO
                           47
```

***Figure 4:* MC6809 program to move any length of memory. Timing = 10 + 20 * N.**

```
                        (LNG TIM)
        LDX #FROM         3   3    INITIALIZE 16-BIT "FROM" POINTER
        LDU #TO           3   3    INITIALIZE 16-BIT "TO" POINTER
        LDY #CNT          4   4    INITIALIZE BYTES TO MOVE
                                     COUNT

LOOP    LDA , X+          2   6    LOOP: GET BYTE TO MOVE; BUMP
        STA , U+          2   6      POINTER; MOVE WORD; BUMP
        LEAY -1, Y        2   5      POINTER; DECREMENT COUNT
        BNE LOOP          2   3      BY ONE UNTIL COUNT IS ZERO
                         18
```

***Figure 5:* MC6809 program to move any length of memory. Timing = 10 + 12 * N.**

```
                        (LNG TIM)
        LDX #FROM         3   3    INITIALIZE 16-BIT "FROM" POINTER
        LDU #TO           3   3    INITIALIZE 16-BIT "TO" POINTER
        LDY #CNT          4   4    INITIALIZE BYTES TO MOVE
                                     COUNT

LOOP    LDD , X++         2   8    LOOP: GET WORD TO MOVE; BUMP
        STD , U++         2   8      POINTER; MOVE WORD; BUMP
        LEAY -2, Y        2   5      POINTER+2; DECREMENT
        BNE LOOP          2   3      COUNT BY TWO UNTIL COUNT
                                   IS ZERO

                         18
```

Since the second program moves words, Y gets decremented with the contents of Y minus 2.

The program in figure 6 combines the code from figures 4 and 5 to produce a fast, general-purpose memory move for the MC6809 which moves any number of bytes, a word (two bytes) at a time. This routine uses the powerful double-byte move code of figure 5, only without the even-byte restriction. The way this is achieved is straightforward. The "CNT" word is tested for odd length by first using the "TFR" instruction to move the "CNT" to the D register. This is followed by a "LSRB" (logical shift right B) which sets the carry bit if "CNT" is odd. If the length is even, the routine branches directly to the double-byte move routine. Otherwise, the "odd" byte is moved first using the figure 2 code. This routine is 29 bytes long and takes approximately 12 machine cycles per byte moved. The general purpose routine takes almost twice the bytes of the MC6809 short move but requires 20% less time per byte!

Figure 7 summarizes the results for the memory moves discussed in figures 1 through 6. The byte ratio column is the number of 6502 bytes divided by the MC6809 bytes for a given comparison. The cycles per byte ratio column is the 6502 cycles required, per byte moved, divided by the MC6809 cycles per byte. For example, the row labeled " <= 256 bytes" shows that the 6502 program from figure 1 used 11 bytes and needed about 14 cycles per byte moved. The MC6809 program in figure 2 needed 15 bytes and used 15 cycles per byte moved. The "byte ratio" is then 11/15 or 0.73. The "cycles per byte ratio" is 14/15 or 0.93.

As the table in figure 7 shows, the 6502 is good at moving small blocks of memory with fixed addressing. The MC6809 code for a move of fewer than 256 bytes comes close to keeping up with the 6502, but requires over a third more bytes. Our general-purpose double-byte move routine is slightly faster than the 6502 but is much more costly in terms of bytes. Since the MC6809 general purpose routine is

(Continued on page 87)

***Figure 6:* General MC6809 program to move any length of memory. Timing = 41 + 12 * N.**

```
                   (LNG TIM)
      LDX #FROM     3    3    INITIALIZE "FROM" POINTER
      LDU #TO       3    3    INITIALIZE "TO" POINTER
      LDY #CNT      4    4    INITIALIZE BYTES TO MOVE
                                COUNT

                              CHECK FOR ODD COUNT
      TFR Y, D      2    6      MOVE COUNT LSB TO B
      LSRB          1    2      REGISTER; SET CARRY IF COUNT
      BCC LOOP      2    3      IS ODD; TO LOOP IF COUNT IS
                              EVEN

      LDA ,X+       2    6    ELSE GET ODD BYTE; BUMP
      STA ,U+       2    6      POINTER; MOVE IT; BUMP
      LEAY -1, Y    2    5      POINTER; DECREMENT LOOP
      BEQ DONE      2    3      COUNT; QUIT COUNT IS ZERO

LOOP  LDD , X++     2    8    LOOP: GET NEXT WORD; BUMP
      STD , U++     2    8      POINTER+2; MOVE IT; BUMP
      LEAY -2, Y    2    5      POINTER+2; DECREMENT LOOP
      BNE LOOP      2    3      COUNT BY TWO UNTIL COUNT
                              IS ZERO

DONE  EQU *
                   29
```

***Figure 7:* 6502/MC6809 byte and cycles per byte ratios for figures 1 through 6.**

Class of Move	Byte Ratio	Cycles/Byte Ratio
< = 256 bytes		
6502-Fig 1 / MC6809-Fig 2	11 / 15 = 0.73	14 / 15 = 0.93
6502-Fig 1 / MC6809-Fig 6	11 / 29 = 0.38	14 / 12 = 1.17
> 256 bytes		
6502-Fig 3 / MC6809-Fig 4	47 / 18 = 2.61	35 / 20 = 1.75
6502-Fig 3 / MC6809-Fig 5	47 / 18 = 2.61	35 / 12 = 2.92
6502-Fig 3 / MC6809-Fig 6	47 / 29 = 1.62	35 / 12 = 2.92

easily made into a subroutine, the extra byte cost might be lessened by sharing the code with other parts of a program. The 6502 code example lacks this versatility since it is limited to fixed 256 byte ranges.

The more complex memory move of more than 256 bytes is where the MC6809 really asserts itself. MC6809 versions were presented for a single byte move, an even-length-only double-byte move, and a general-purpose "any length" move. For "byte tight" applications, the MC6809 byte mover runs 1.75 times faster than the 6502, while the 6502 uses 2.6 times the bytes of the MC6809. While the MC6809 double-byte mover from figure 5 is restricted to even-byte moves only, it rips along at almost three times the rate of the 6502 with no more code than the single-byte version. The MC6809 general-purpose double-byte mover (figure 6) maintains the blazing speed of figure 5 without being restricted to even-byte moves. The 6502 move uses 1.6 times the bytes of the MC6809 general purpose mover.

These results show clearly the degradation of speed and code size of the 6502 for memory moves across page boundaries. We feel that the MC6809 has also been easier to program. There has been no need to set up and manipulate indirect pointers with registers of only eight bits, as was necessary on the long 6502 memory move.

The authors may be contacted at Motorola, Inc., Microprocessor Design, Maildrop M2880, 3501 Ed Bluestein Blvd., Austin, Texas 78721.

MICRO

MICRO

PET Vet

By Loren Wright

PET's Powerful Screen Editing — Are you getting the most from it?

When it comes to BASIC programming, most computers employ what I call a 'teletype' mentality. You type in the line number, followed by the line contents. Then you send all the characters you typed to the computer by pressing the RETURN key. Before you hit RETURN, you can correct errors by deleting back to the error, making the correction, and then retyping the rest of the line. If you have made the mistake of hitting RETURN before you notice the error, the only way to correct it is to start over. The characters you just typed in are still sitting there on the screen, but you can't do anything with them. The computer has forgotten all about what you typed. Sure, there's a copy in your BASIC program, and you can list the line to the screen, but you can't do anything with that either. Some of these computers allow some primitive editing — if you can remember the obscure control codes and are willing to copy characters back into memory. It is usually faster to retype your BASIC line.

The PET BASIC input works quite differently. Instead of keeping track of the stream of characters you're typing, the PET just puts them up on the screen. You can move the cursor anywhere you want, draw pictures, check your disk directory, or list another line. All that counts is where the cursor is when you hit RETURN. If it's on a line that begins with a number, then the system reads the line as a BASIC statement. If the line doesn't begin with a number and the line isn't a valid direct command, then the PET will respond with a ?SYNTAX ERROR.

This system offers a lot of advantages to the PET BASIC programmer. To correct an error in the line you're typing, all you have to do is move the cursor, make the correction, move the cursor back to the end of the line, and continue typing. If you want to correct a line you've already entered into the program, just list it to the screen (if it isn't already there), make the correction, and hit RETURN with the cursor anywhere on the line to enter the new version.

Tricks of the Trade

It seems simple enough, doesn't it? If you have never tried to write a BASIC program on another computer, you probably take it all for granted. There are, however, a few traps you can fall into, and there are a few little tricks you can use to make the system work even better.

1. Clear the screen before you list lines you're going to edit. If you don't, garbage left over from your program run will appear on the same lines as your BASIC lines and those characters will be put into memory when you hit RETURN.

2. The cursor does not have to be at the end of the line when you hit RETURN. As soon as you have completed your change, you can hit RETURN.

3. If you're at the left end of a line and you want to be at the right end, the fastest way to get there is not to go forward, but rather to back up to the end of the previous line and move down one line. If you're at the right end and you want to be at the left, then the opposite holds true.

4. Don't forget the HOME key! If you're at the bottom of the screen, it's much faster to hit HOME than to move the cursor up all those lines.

5. Shifted RETURN is not the same as RETURN! It will move the cursor to the beginning of the next line, but it will not send the line to the PET for processing.

6. If you need to move a line to make room for others, just list it, change the number and hit RETURN. Remember, though, that the old copy is still there at the old number until you delete it or replace it. This technique is also particularly handy when you are writing a program that is very repetitive (e.g., a series of subroutines, where several lines are identical in each routine). Just type the line once, and for each copy, change the line number and hit RETURN.

7. If a listed line exceeds two lines, the overflow is not considered as part of the line when you try to re-enter it. This happens because you used abbreviations for BASIC keywords (like '?' for PRINT) when you originally entered the line. Using the keyword abbreviations is fine, but try to avoid using such long lines.

8. Be careful with BASIC lines that occupy only one screen line. Under some circumstances it is possible to get the next line listed on the screen entered as part of your current line. The cure is to list only one such line at a time.

9. Use the screen as a temporary storage device! This one takes some care. Let's say you have just typed in 30 lines, and you suddenly decide that only eight of them are good. You could delete each unwanted line by typing its number, but it is faster to list the lines you want to save, type NEW and RETURN. Then position the cursor on the first line, hit RETURN, and keep hitting RETURN until all the lines are restored. If any of them scroll off the screen before you re-enter them, they will have to be retyped.

Most of these tricks work fine for direct commands, too. For instance, if you misspell the file name in a LOAD command, just stop the search, move the cursor to the command line, make the change, and hit RETURN.

Programmed Cursor Mode

Another powerful feature of the PET is its character-programmable cursor commands. Cursor moves can be included as special characters in a BASIC string so that when the string is printed, the cursor moves are executed. To get these characters into the string, the PET has something called "programmed cursor mode," where pressing a cursor key causes the appropriate special character to appear on the screen instead of the cursor move itself. The programmer loses control of the cursor while in programmed cursor mode (PCM), and if you don't know what's going on, it's easy to get

PET VET *(Continued)*

frustrated. PCM is entered under only two circumstances:

1. When you type a double quote, you enter PCM; when you type another, you exit. The PET keeps track of the number of quotes in a line, but it can be fooled.

2. When you use the INSERT key, the PET counts the number of times you press it, and for that number of characters it is in PCM. The assumption is that most insertions will be within strings.

Quite often you want to be in PCM when the PET isn't, and *vice versa*. To get in or out, just type a quote and then delete it if you don't need it. The PET only recognizes when you type quotes, not when you delete them! If you've done an insertion, just type spaces for the number of characters you inserted and you will regain control of the cursor. The spaces can then be deleted.

In other instances, things get completely out of hand and you just want to start over. The answer is shift-RETURN! It will bail you out of PCM and it will preserve the original version of the line you're editing.

It also helps to know what the cursor control characters look like when they're included in strings. This depends both on which model PET you have and on which character set you're in. A few experiments, and perhaps a little crib sheet taped to your PET will help.

Editing Improvements

If you do a lot of BASIC programming even these powerful features may not be enough. Autonumber, renumber, delete functions, and repeating keys are probably the most useful enhancements. List scrolling and programmable function key capability are also useful. These functions are available in a number of commercial ROMs, such as Programmer's Toolkit, Disk-O-Pro, Command-O, POWER, EZAID, and others. Not all offer all of these editing features, but all include other capabilities.

Fat 40, 8000 Series, and VIC

These recent Commodore machines incorporate repeating keys and an ESCAPE key to get out of programmed cursor mode. The 8000 series computers have additional special characters for window, delete line, insert line, scrolling and other commands. The VIC has special characters for color commands and its eight programmable function keys.

Commodore's New Computers

With three new computers added to its existing line, Commodore will have an iron in just about every part of the microcomputer fire. The Ultimax ($149.95) is a color-and-sound computer that hooks up to any home TV set. It will compete very favorably with the Sinclair ZX-81, Mattel Intellivision, and Atari VCS. The Ultimax will support joysticks, paddles, light pens, cartridges, and cassette storage. To achieve such a low price, Commodore has provided only a limited amount of RAM and a flat membrane keyboard.

The Commodore-64 ($599) is designed to compete with the Atari 800 and Apple II with its full-size keyboard, 64K of memory, function keys, and sophisticated sound capabilities. Also announced was a 16K VIC — the SuperVIC.

MICRO

MICRO™

From Here to Atari

By James Capparell

Editor's Note: This is the last "From Here to Atari" that will appear in MICRO. We thank Jim Capparell for his efforts and wish him success with his new Atari magazine, ANTIC. And we say to the Atari Community: send us your work! We are very interested in publishing Atari material.

The "front jacks" on the Atari 400 and 800 are used for everything from game controller ports to printer and graphics ports. To get the most out of them it is necessary to understand something about the hardware. This month, I'll provide a description of the pin configuration of these jacks, the memory locations assigned to the jacks, and an example of how each pin may be individually configured as input or output.

The hardware controlling these jacks is a 6820 PIA (Peripheral Interface Adapter). This chip consists of two ports labeled "A" and "B". Port A controls Jacks 1 and 2. Jacks 3 and 4 (on your right) are controlled by Port B. The jacks are numbered left to right as you face the console. There is a numbering discrepancy: BASIC numbers them 0 to 3. The BASIC statement,

```
10?STICK(0),STICK(1),STICK(2),STICK(3)
```

will print the values read at Jack 1, Jack 2, Jack 3, and Jack 4.

Each port consists of three registers — the Control Register, the Data Direction Register (DDR), and the Data (buffer) Register (DR). The PIA is a 40-pin chip. Due to a limitation on available pins, the DDR and the DR share the same address. (See table 1.)

Bit 2 of the control registers determines whether the DDR or the DR is addressed. When set to 0, bit Z addresses the DDR, but when set to 1 the DR is addressed. The data register simply holds data. When the jacks are configured as inport ports, the DR holds the data for the Atari to read. When the jacks are configured as output ports, the DR holds data to be written to an external device. The DDR determines for the PIA which lines are input and which are output.

To configure Jack 1 as input and Jack 2 as output, it is necessary to tell the PIA the direction for each of the eight bits in Port A. To accomplish this, perform the following steps:

1. Set bit 2 of PACTL($D302) to 0. This allows us to address the DDR.
2. Write 00001111 = 15 to address $D300 (note a 1 bit indicates the associated line is output).
3. Set bit 2 of PACTL to 1. This restores address $D300 to the data register.

At this point, Jack 1 can be read normally with a STICK(0) statement. Jack 2 can't be read since it is configured as an output jack. Try the following:

```
10 POKE 54018,0 :REM Go talk to
   Data Direction Register

20 POKE 54016,15 :REM Jack 1 is input,
   Jack 2 is output

30 POKE 54018,4 :REM Reset to data
   register

40 REM connect joystick to Jacks 1 and 2

50 ?STICK(0),STICK(1) :REM Print out
   values from Jacks 1 and 2

60 GOTO 50 :REM Loop forever

70 REM Move Joysticks 1 and 2, only
   Joystick 1 will register a change.
```

Whenever your system is turned on all jacks are configured as input. That is, the operating system writes a 0 to the Data Direction Reigsters in Ports A and B. The values returned at these jacks are always a 1 when there is no input — logical 1 is false. This helps explain why a 15 is read even when there is no

Table 1

I/O Address	O.S. Shadow Address
$D300 (54016)	$278 (632)
Port A data register or data direction register when bit 2 of PACTL is 0. This address corresponds with Jack 1 and Jack 2. BASIC statements STICK(0) and STICK(1) read this port.	
$D301 (54017)	$279 (633)
Port B data register or data direction register when bit 2 of PBCTL is 0. This address corresponds with Jack 3 and Jack 4. BASIC statements STICK(2) and STICK(3) read this port.	
$D302 (54018)	$27A (634)
Port A control register. Insert a value of 4 (bit 2 = 1) and $D300 becomes the Data Register.	
$D303 (54019)	$27B (635)
Port B control register. Insert a value of 4 (bit 2 = 1) and $D301 becomes the Data Register.	

The shadow registers are updated at Stage 2 of Vertical blank processing — no more frequently than every 1/60 second. If your program requires more accurate data, read the associated hardware registers at addresses $D300 and $D301.

From Here To Atari *(Continued)*

input from a joystick. Look at the diagram in figure 1 for correspondences between bits in DDR and bits in data buffer.

When a jack is configured to input and the following BASIC statement is executed:

```
10 ?STICK(0):GOTO 10
```

the following values will be printed as the joystick is manipulated:

1111 (15) = stick neutral
1110 (14) = forward
1101 (13) = backward
1011 (12) = left
0111 (11) = right

Combinations (diagonal)

1010 (10) = forward/left
1001 (9) = backward/left
0101 (7) = backward/right
0110 (6) = forward/right

The pin configuration for each jack is as follows:

1 2 3 4 5
6 7 8 9

Console (male)

Pin 1 = forward
Pin 2 = backward
Pin 3 = left
Pin 4 = right
Pin 5 = pot (paddle control)
Pin 6 = joystick trigger at $D010-$D013 (CTIA)
Pin 7 = +5V
Pin 8 = gnd
Pin 9 = pot (paddle control)

These front jacks are versatile and easy to use. I've connected a Hewlett Packard Bar code reader to my 800. Others have used them for graphics printer interface and 10-key pad for business use as well.

Joystick Data

7	6	5	4	3	2	1	0

Jack 2 (stick 1) — Jack 1 (stick 0)

7	6	5	4	3	2	1	0

Data Direction Register

When bit = 0 then switch pressed
bit = 1 then switch not pressed

Bit
0, 4 = Forward
1, 5 = Backward
2, 6 = Left
3, 7 = Right

MICRO

MICRO™

Reviews in Brief

Product Name: **Mittendorf High-Resolution Graphics Board**
Equip. req'd: OSI
Price: $40 bare board; $185 kit
Manufacturer: Mittendorf Engineering
905 Villa Nueva Dr.
Litchfield Park, AZ 85340

Description: A 6″ × 6″ circuit board which adds 256 × 256 black and white high-resolution graphics to OSI systems. The same kit works with the superboard or 540 video board. The Mittendorf board contains 8K of 2114 memory which can be used for program storage when not using graphics.

Pluses: Combines with the present video signal to give hi-res graphics and the OSI character set on the same monitor. The graphics memory can be wired at one of several addresses.

Minuses: The Mittendorf board requires 16 connections into the OSI video circuits. The 540 version requires additional jumpers to all the bus lines. The superboard version requires removal of the data buffers which prevents further use of the 40-pin expansion port. All 256 × 256 dots are not visible. Dots are lost to overscan in the same ratio that are lost from OSI's nominal "32 × 32" characters.

Documentation: Several sets of construction plans dependent on system, software to add graphics commands to BASIC, demonstration examples.

Skill level required: Experienced builder; modification of present video circuits required.

Reviewer: Earl D. Morris

Product Name: **Visiterm**
Equip. req'd: Apple II or Apple II Plus and communications device: Apple Communications card, CCS card plus modem or D.C. Hayes Micromodem
Price: $149.95
Manufacturer: Personal Software Inc.
1330 Bordeaux Drive
Sunnyvale, CA 94086
(408) 745-7841
Author: Tom Keith
Copy Protection: Yes
Language: 6502 Machine Language

Description: A communication package for interfacing the Apple with other computers, permitting the transfer and receipt of sequential text files.

Pluses: Visiterm uses one of two high-resolution character sets and the hi-res screen to provide the Apple user with a seventy-column display when communicating with other systems. This feature is particularly valuable when accessing a mainframe computer since up to eighty columns are often transmitted to the user. The character sets are programmable, permitting the user the valuable resource of redefining keys to permit the generation of mainframe-dependent control characters. One of the most technically challenging aspects of using a micro as an intelligent terminal is the problem of required control keys such as the Break and X-on and X-off signals. The very extensive manual contains a thorough discussion of data communications. The main part of the manual contains almost 100 pages. The appendices, glossary, and detailed index contain almost as many pages. A print utility is provided that allows the user to obtain a hard copy of received data. The utility has many useful options which permit printer control and output formatting. A very powerful package with features useful for more than just communications.

Minuses: One limitation is that only sequential text files can be exchanged. However, stand-alone utility programs, outside of Visiterm, are provided to convert BASIC and binary files to and from text files. Visiterm does not provide the user with the ability to edit the data buffer exchanged. A separate text file editor may be needed by the user. (This is mentioned because at least one competing package does provide this feature.) An abbreviated summary of the Visiterm options would be helpful. For those familiar with VisiCalc II, also manufactured by Personal Software, a flow chart of commands is included which provides the user an excellent reference. It is sometimes difficult to locate the discussion of a particular topic. For example, it is possible to obtain a CATALOG of text files on a disk from within Visiterm. However, the section describing this is found under lesson Three, "File Transfer Mode." You may find it necessary to read most of the three lessons provided before you are comfortable using the package.

Skill level required: For the moderately sophisticated user, preferably with some understanding of communications problems.

Reviewer: David R. Morganstein

Product Name: **A2-3D1, A2-3D2 and A2-GE1 Graphics Package**
Equip. req'd: 48K Apple II or Apple II with disk drive
Price: $119.85
Manufacturer: SubLogic Communications Corp.
713 Edgebrook Drive
Champaign, IL

Description: Programs to help the user produce, project, and maneuver three-dimensional shapes on the two-dimensional screen medium. Contains impressive features for recording motion sequences and replaying them. Individual snapshots or slides of a motion sequence can also be recorded for later display. Provision is made for interfacing routines to Applesoft programs.

Pluses: Either eye or object movement can be commanded, thereby adding flexibility to sequence definition. Exceptional ease in interfacing to BASIC programs.

Minuses: Extensive memory and disk space is required. The included demo disk inadequately demonstrates the

(Continued on next page)

package's considerable capabilities. The potential buyer should be aware of this and not underjudge the product.

Documentation: Superb documentation leads the user through a continuing example that eventually opens a 3-D garage door. Along the way, all capabilities are presented and an example of each is given. Surprisingly well-written in a style that lends itself to use as a tutorial or a reference.

Skill level required: Competent BASIC programmer with some exposure to assembly language.

Reviewer: Chris Williams

Product Name:	**56K CMOS Static Memory Board**
Equip. req'd:	OSI 48-pin bus
Price:	4K $200; 24K $450; 56K $850
Manufacturer:	Micro-Interface 3111 So. Valley View Blvd. Suite I-101 Las Vegas, Nevada 89102

Description: The Micro-Interface board puts 56K of memory, an expanded monitor ROM and a parallel printer port all into a single bus slot. The board enable can be set at each 2K address selection, allowing any combination of 6116 CMOS RAM and/or 2716 EPROM to populate any portion of the 56K memory space. The use of CMOS RAM reduces the power requirements for 48K to less than ½ amp, allowing memory expansion without a new power supply. Decoding is also provided for a 1.75K enhanced ROM monitor between $F800 and $FFFF. Micro-Interface sells several such monitors, or you can program your own into a 2716 EPROM.

Pluses: Very low power RAM rated for 2 MHz operation. Combines functions of several boards into one bus slot. Provision is made for multi-user or memory banking. The parallel port supports either a 6821 PIA or 6522 VIA. The board is available assembled with any amount of memory between 4K and 56K. Additional memory chips are easily installed.

Minuses: For 8/16/24K the Micro-Interface board is more expensive than the same memory assembled from D&N.

Documentation: Instructions for installing jumpers, memory addresses, chip types, jumper locations, and functions are printed on the circuit board.

Reviewer: Earl Morris

Product Name:	**Cer-Comp Color Computer Editor**
Equip. req'd:	TRS-80C Color Computer with 16K memory
Price:	$19.95
Manufacturer:	Cer-Comp 5566 Ricochet Ave. Las Vegas, Nevada

Description: A screen editor based on line numbers; resides in R/W memory, distributed on cassette tape using the Color Computer format. The editor has 21 commands that modify text produced in a BASIC-like format. Two edit modes allow spaces or characters to be inserted or deleted from existing lines, and allow forward and reverse scrolling through existing text. Cursor control is either single space per keystroke (forward or back) or single keystroke to reach either end of a line. Block move and copy, search and replace, list to screen or printer with or without line numbers, load and save tapes, append a second tape to existing text, and some special commands for BASIC files are available. In addition, line numbers can be removed from a file to save space, or added to files from other editors to allow editing.

Pluses: Low price, good versatility, easy to learn, does not require Extended BASIC. Works with machine-language monitors.

Minuses: Instructions and documentation lacking, no listing supplied. Although cursor control is adequate, a repeat key function for continuous cursor scroll would be advantageous.

Skill level required: Normal typing skills, ability to visualize final page format.

Reviewer: Ralph Tenny

Product Name:	**Color Computer Disk System**
Equip. req'd:	TRS-80 Color Computer, 16K w/Extended BASIC
Price:	$600
Manufacturer:	Tandy Radio Shack P.O. Box 2625 Fort Worth, TX 76113

Description: A 35-track, double-density disk operating system for the Color Computer. Capacity is 156,672 user-available bytes, and 68 maximum files, on a standard 5¼ inch soft-sectored diskette. The system includes a single drive, a disk controller ROM pak, and a connecting cable that allows two drives at a time on line. A four drive cable is optional. System utilities include BACKUP, COPY, and FORMAT. The operating system requires 2K of RAM and no disk space (except for directory tracks). Files are cataloged with an eight-character file name, and three-letter extension. VERIFY, LSET, RSET, MKN$, and CVN$ are typical commands available to the system which are used in other DOS systems.

Pluses: Because the operating system is on ROM, it requires very little extra memory from the machine. There is no DOS to learn, as disk commands are an extension of BASIC. As there is no DOS with the COLOR disk system, all disk commands can be executed from BASIC, inside or outside a program. The Microsoft disk BASIC includes the "WRITE" command, which allows easier formatting and creation of serial data files, and random access variable length files. The disk BASIC is simple, and easy to learn.

Minuses: Utilities are lacking in sophistication, compared to TRSDOS. Backups require pre-formatted destination disks, and there are no file protection capabilities (other than the write protect tab). BACKUP also copies all bytes on a disk, whether there is one small file, or a full disk. Auto start is not supported, and there are no DO files to provide a turn-key system. This could be partially offset by running a standard file upon power-up. This file could load any machine-language routines, and finally load the desired program from a MENU. The CHAIN command is not supported, although it is possible to load and run a program from inside another program. Another useful command that is missing is the ON ERROR GOTO statement.

Documentation: The owner's manual is written to the same high standards of the other two Color Computer manuals. Instructions pre-suppose no previous experience with disk systems or programming. The style is very readable, and some fine demonstration file programs are included. Missing is the usual TRS-80 programmers card.

Skill level required: Novice.

Reviewer: John Steiner

MICRO

LISZT with Strings

by Leonard H. Anderson, Donald Cohen, Richard F. Searle

LISZT turns your Applesoft program listing into an easy-to-understand structured format. The program is designed to be flexible and works with a variety of printers.

LISZT
requires:
Apple II with Applesoft
Disk Drive
Printer

Can you understand a program you wrote six months ago? Do you remember some of those special tricks imbedded in a concatenated line of code? The "LISZT" (Logical Interpreter Statement Zeugmatic Tabulator) can help you understand BASIC source code listings by structuring printouts in a clear, orderly form with a minimum of extra characters. Written for the Apple II Plus, it can be modified for other BASIC dialects.

Credit is due Mark Capella for the first listing program.[1] Since then, two others have been published.[2,3] Not completely satisfied, we decided to start fresh with the following rules:

1. Print results so they are easy to read.
2. Make the program adaptable to various printers.
3. Gather statements in strings for flexibility.
4. Separate REMs from printed code.
5. Omit the concatenation colon and "LET."
6. Split over-long print lines at a logical character.
7. Indent FOR-NEXT loops globally.
8. Indent IF-THEN statements locally.
9. Minimize disk operations.

The main program, LISZTER, was written in linear form to accommodate different printers and to allow easy deletion or addition of special features. This article is both a program description and a partial history of program development.

Applesoft Source Code Structure

Source code structure rules the program. One line of Applesoft BASIC is shown in figure 1. Each line contains five overhead bytes: two for a pointer to the next line, two more for the number, and an end-of-line null (binary zero) byte. The last line number source code ends in three null bytes to indicate end-of-program.

All variable names, strings, and punctuation not a function are expressed as 7-bit ASCII with most-significant-bit (MSB) set false or zero. All function words (IF, NEXT, REM, etc.) are stored as one-byte "tokens" with MSB set true or high. There are 107 Applesoft tokens.[4]

Starting the Program Organization

Figure 2 is the initial flow chart. Each program byte is examined, beginning with decimal memory location 2049. ("Standard" ROM Applesoft code begins here. It can be changed and will vary for other BASICs.) String variables hold the line number in N$, statement text in a "gather" string, G$, and the "combination" printout string, C$.

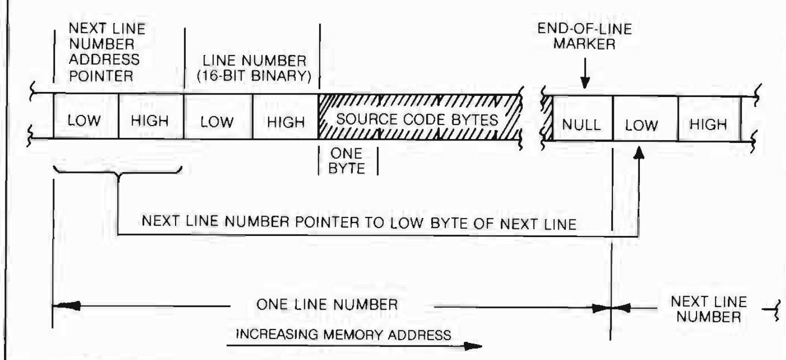
A remarks flag is set if a REM token is encountered. The first decision separates remarks by blank print lines but groups successive remarks without blank lines. Remarks are highlighted without appearing to be part of the main coding.

ASCII characters and token bytes are parsed next with tokens reconverted to the original function word. This section and the print line formatting section receive the most attention. A prime example is separating concatenated statements and allowing indication of over-long text lines.

Holding Two BASIC Programs in Memory

Applesoft reserves two bytes in page zero (first 256 bytes) for the starting address. Start location is *normally* decimal 2049 for ROM BASIC, stored in locations 103 (low byte) and 104 (high byte). End-of-program in memory is in locations 175 (low) and 176 (high). Either can be changed from the keyboard or program in memory.

Figure 1: **Structure of one source-code line number in Applesoft.**



Apple's DOS allows the simulation of keyboard commands with an EXEC Text File. An EXEC file loads statements into the keyboard buffer. Each statement is then executed as if it were a keyboard command.

The program to be listed is loaded first. The EXEC file is called next by typing "EXEC LISZT." LISZT then changes normal program start address to the end of program plus two, loads and runs the LISZTER working program. Loading LISZTER will automatically set the new end-of-program address.

Although two programs are now in memory, Applesoft will only execute LISZTER as indicated by the starting address changed by EXEC file LISZT. Original start and end addresses are held in page zero scratchpad locations; LISZTER resets start and end from these scratch locations on completion of printout.

EXEC file LISZT is generated by the short program in listing 1. MAKE LISZT may be deleted after generating LISZT. LISZT EXECution commands are those indicated within quotes in MAKE LISZT line numbers 225 through 265.

LISZTER start location is set slightly higher than normally expected. This and the extra nulls will insure that the listed program can be RUN normally after LISZTER resets start and end addresses on printout completion. Normal source code ending must be three successive null bytes.

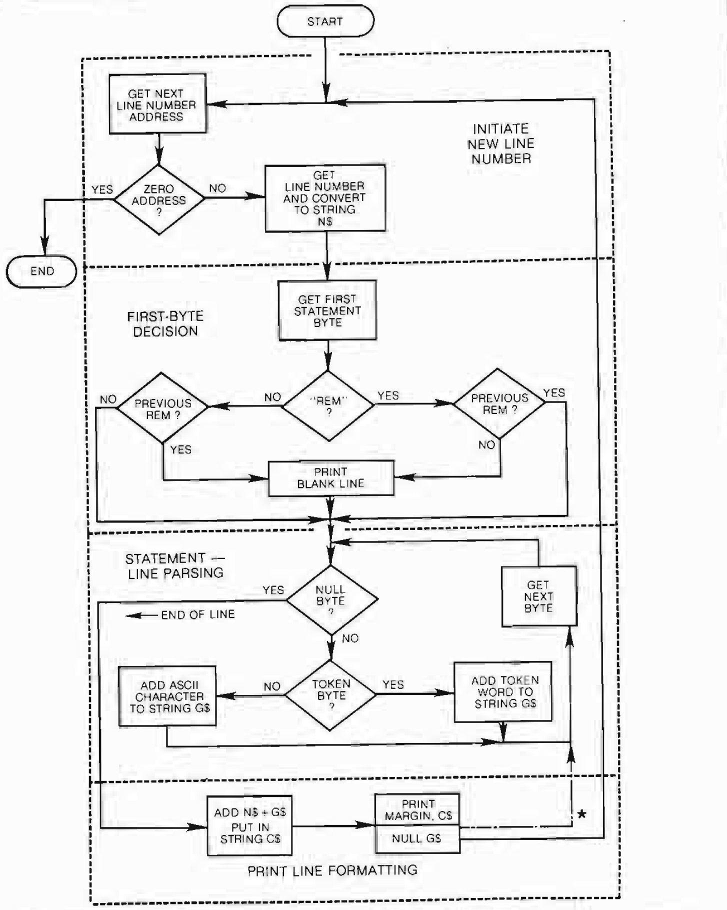
Setting Up LISZTER

LISZTER begins execution at line number 82 by initializing the variables. Initializing will speed up execution, especially with string variables in Applesoft. Token array T$ contains the 107 function words expressed as literal strings in the DATA statements. Direct expression as strings allows spaces to be added for clarity in gathering and converting the tokens.

The REM token word was changed to an asterisk. It is left as an isolated DATA declaration for those desiring another symbol or word. LET appears as a null string in line 88 to permit completion of the array; token parsing will skip over a LET.

Screen prompts in lines 94 to 100 are optional. Printed page length is normally 60 lines per page including the header. Indent spacing is normally four column spaces, fitting the REM asterisk with three following blanks.

***Figure 2:* Initial Flow Chart and Sectioning of LISZTER**



* Asterisk indicates possible path for next print line of same line number.

***Listing 1:* MAKE LISZT "EXEC" file generator printed with the LISZTER program in listing 2.**

```
                         MAKE LISZT
                     LEONARD H. ANDERSON
                  PRINTOUT ON 20 AUGUST 1981
                           Page 1

200      *      "MAKE LISZT" GENERATOR FOR "LISZT" EXEC FILE
205      *       LEONARD H. ANDERSON      7/5/81
210   D$ = Chr$(4)
      C$ = Chr$(13)+D$
215   Print C$"OPEN LISZT"
220   Print C$"WRITE LISZT"
225   Print  "POKE208,PEEK(103)"
      Print  "POKE209,PEEK(104)"
      Print  "POKE210,PEEK(175)"
      Print  "POKE211,PEEK(176)"
230   Print  "POKE104,PEEK(211)"
235   Print  "IF PEEK(210)<254 THEN POKE103,(PEEK(210)+1)"
240   Print  "IF PEEK(210)>253 THEN_
          POKE103,(PEEK(210)-254):POKE104,(PEEK(211)+1)"
245   Print  "POKE(PEEK(210)+PEEK(211)*256-2),0"
250   Print  "POKE(PEEK(210)+PEEK(211)*256-1),0"
255   Print  "POKE(PEEK(210)+PEEK(211)*256),0"
260   Print  "POKE(PEEK(210)+PEEK(211)*256+1),0"
265   Print  "POKE(PEEK(210)+PEEK(211)*256+2),0"
270   Print  "RUN LISZTER"
275   Print D$"CLOSE"
280   End
                      End of Listing

Program Length = 642 Bytes,   Total of 17 Line Numbers
19 Total Non-Rem Statements,   2 Total Remarks
```

Listing 2

```
                    LISZTER 'UNIVERSAL' VERSION
                       LEONARD H. ANDERSON
                          24 JULY 1981
                             Page 1
0   Goto 82

1       *    "GET BYTE" SUBROUTINE

2   P = P+1
    B = Peek(P)
    Return

3       *    "BLANK LINE PRINT" SUBROUTINE

4   D = 0
     Gosub 6
    Print B$
    Return

5       *    "TEST PAGE" SUBROUTINE.  NOTE: SINGLE CHARACTER SET
             PRINTERS SHOULD DELETE "GOSUB 111" & "GOSUB 112"
             THROUGHOUT, LINES 110 THRU 113.

6   LC = LC+1
    If LC = <LP Then
         Return
          *    NOT A NEW PAGE

7    Gosub 111
    LC = 6
    PC = PC+1
    Print B$
    Print BB$;LB$;"(continued)"

8       *    A FORM-FEED TO GET TOP OF NEXT PAGE AFTER 'CONTINUED'
             NOTICE.  NOTE: PRINTERS WITH F-F COMMAND CAN USE "PRINT
             CHR$(12)" IN PLACE OF FOR-NEXT LOOP; THE SINGLE-SPACE
             "PRINT S$" (THROUGHOUT) IS REQUIRED BY CENTRONICS 737.

9   For K = LP To 63
         Print B$
    Next

10      *    PRINT THE HEADER, BRACKET LINE NUMBER FOR STATEMENTS
             SPLIT OVER TWO PRINT PAGES

11  H$(4) = "Page "+Str$(PC)
    For K = 1 To 4
         E = Int((LL-Len(H$(K)))/2)+1
         Print M$;Left$(BB$,E);H$(K)
    Next
    K = Fre(0)
    Print B$
    If Not D Then
         Return

12      *    PUT LINE NUMBER IN BRACKETS AS A STATEMENT
             IDENTIFICATION ON NEXT PAGE

13  N$ = Str$(Val(N$))
    K = Len(N$)
        *    N$ IS NOW WITHOUT SPACES; BRACKET N$ AND ATTACH TO
             STATEMENT CHARACTERS

14  C$ = Right$((Left$(LB$,(6-K))+Chr$(91)+N$+Chr$(93)+B$),8)+
         Right$(C$,(Len(C$)-8))
    K = Fre(0)
    Return

15      *    GET THE TWO-BYTE POINTER TO NEXT LINE NUMBER; NULL
             INDICATES END.

16   Gosub 2
    D = B
     Gosub 2
    D = 256*B+D
    If D>0 Goto 27

17      *    END OF PROGRAM, PRINT NOTICE

18   Gosub 111
     Gosub 4
     Gosub 4
     Gosub 6
    Print M$;LB$;"End of Listing"
                                                          (continued)
```

Line 104 assumes the printer interface located in peripheral slot 1. The "POKE 33,30" sets screen window width to 30 columns, necessary only for certain Apple I/O interfaces. Line 106 is the main printer control statement.

Subroutines

"GET BYTE" simply advances program byte pointer P and fetches a decimal value of the byte in B. "BLANK LINE PRINT" is used mainly in REM separation. String S$ is a single blank space and required by Centronics 737 printers; most other printers will accept a single PRINT without a variable.

"TEST PAGE" (lines 6 through 14) does several things: line count (LC) advance and test, header printing and a "continued" indicator print on the current page if another page is to be printed. Since concatenated statements are separated, some may be printed on the following page. Lines 13 and 14 place brackets around the line number and print that on the following page for clarity.

Upper and Lower Case

The mixture of upper and lower case characters is due primarily to the T$ array DATA declarations. We decided that program statements would appear better with the familiar function words in a mixture, non-token statements in normal upper-case only. The choice is up to the user.

Since none of us has direct lower-case control, an available utility program was used to "zap" the desired characters directly on the disk.[6] You should do this on a copy file in disk; strange results occur if a token is accidently changed!

First-Byte Decision

In the section occupying lines 29 through 33 on listing 2 remarks are separated and grouped. A single, first-byte colon in Applesoft is equivalent to a REM; first-byte colons are changed to REMs since the asterisk-equivalent voids the original source code separation intent.

Statement Line Parsing

Line 34 begins a number of decisions. A null program byte indicates end-of-line in Applesoft and jumps to the print line formatting routine at line

54. A decimal value between 1 and 127 is an ASCII character byte; any value above 127 is a token.

The double quote test at line 36 allows colons *within* quotes or remarks. Any other colons are treated as delimiter characters and tested at line 37. A delimiter forces a new print line but not a new line number as in the case of a null value byte.

Control characters are converted to upper case equivalents. Besides making control characters visible, conversion allows a printer to continue without suddenly switching to a new mode! We enclosed control characters in vertical bars because that print character has little use in normal printing.

Token byte values are changed to allow you to gather them from the T$ array. A token value out of normal range is made into a distinctive word at line 40. A test-true here would indicate an error.

The REM flag set at line 43 is primarily for concatenated remarks. The remarks counter is optional and used only for end-of-listing statistics. REM spacing variable RS is set to one for indenting remarks. While remarks are highlighted, we also wanted their appearance out of the normal program flow.

The FOR flag sets up the start of global FOR-NEXT indenting. The FOR spacing counter is advanced in print line formatting to allow completion of the entire FOR statement. The NEXT test at line 48 removes one FOR indent space. This space is held at zero in case an intermediate (but legal) NEXT is used with the loop.

Conditional tests add an indent space on completion of a THEN. Anything following a THEN, even if only a line number, is considered a separate statement. An IF-GOTO is considered a single statement. The choice was arbitrary to reduce total code.

A LET token is ignored by choice. Omitting line 47 allows you to print a LET.

DATA flag (DF) is used solely in print formatting. When set, it allows splitting an over-long print line only on commas. This is useful when DATA declarations contain strings with spaces as in LISZTER itself.

Listing 2 *(Continued)*

```
19      *   OPTIONAL STATISTICS

20   Gosub 4
     Gosub 4
     Gosub 6
    Print M$;"Program Length =_
        ";(Peek(211)-Peek(209))*256+Peek(210)-Peek(208);" Bytes, _
            Total of ";TN;" Line Numbers"
21   Gosub 4
     Gosub 6
    Print M$;(TS-TR);" Total Non-Rem Statements,  ";TR;" Total_
        Remarks"
     Gosub 4
     Gosub 6
    Print M$;"END"

22      *   TURN OFF PRINTER, DISPLAY END PROMPT ON SCREEN

23  Pr# 0
    Poke 33,40
    Home
    VTab 12
    HTab 11
    Inverse
    Print " END OF LISTING "
    Normal

24      *   RESET PAGE 0 POINTERS FOR THE LISTED PROGRAM

25  Poke 105,Peek(210)
    Poke 106,Peek(211)
    Poke 107,Peek(210)
    Poke 108,Peek(211)
    Poke 109,Peek(210)
    Poke 110,Peek(211)
    Poke 111,Peek(115)
    Poke 112,Peek(116)
    Poke 103,Peek(208)
    Poke 104,Peek(209)
    Poke 175,Peek(210)
    Poke 176,Peek(211)
    End

26      *   MAKE THE LINE NUMBER STRING

27  TN = TN+1
     Gosub 2
    D = B
     Gosub 2
    K = B*256+D
    D = Len(Str$(K))
    N$ = Right$((Left$(LB$,(7-D))+Str$(K)+"  "),8)

28      *   BEGIN LINE PARSING WITH FIRST-BYTE DECISION

29  TS = TS+1
    D = 0
     Gosub 2
    If B = 58 Then
        B = 178
            *   CONVERT "SIMPLE REM" (A ":" FIRST-BYTE) TO ORDINARY
                "REM"

30  If B = 178 And Not RF Then
          Gosub 4
          Goto 34
            *   "REM" FLAGS ARE SET AFTER SEPARATION OF TOKENS;
                REM-GROUPS SEPARATED BY BLANK PRINT LINES.

31  If B = 178 And RF Goto 34
        *   BYPASS RF RESET

32  If RF Then
        RF = 0
         Gosub 4

33      *   RE-ENTRY POINT FOR NEXT BYTE IN STATEMENT DECISION FLOW

34  If B = 0 Goto 54
        *   FORCE A NEW LINE ON THE END-OF-LINE NULL MARKER

35  If B>127 Then
      B = B-127
      Goto 41
        *   BYTE IS A TOKEN; REMAINDER ARE CHARACTERS
```

(continued)

Listing *(Continued)*

```
36  If B = 34 Then
        QF = -QF
          *   TOGGLE QUOTE FLAG FOR COLON-PRINT TEST IN NEXT LINE

37  If B = 58 And Not RF And QF<1 Then
        TS = TS+1
         Goto 54
          *   OMIT THE CONCATENATION ":" AND FORCE A NEW LINE, ELSE
              PRINT THE COLON AS A CHARACTER

38  If B<32 Then
        B = B+64
        G$ = G$+Chr$(124)+Chr$(B)
        B = 124
          *   PRINT CONTROL CHARACTERS AS UPPER-CASE BETWEEN
              VERTICAL BARS; INDICATOR OF CONTROL CHARACTER
              OPTIONAL.

39  G$ = G$+Chr$(B)
     Gosub 2
     Goto 34

40        *   INDICATE UNUSED TOKENS AND CONTINUE

41  If B>107 Then
        G$ = G$+" ?! "
         Gosub 2
         Goto 34

42        *   ACCEPTABLE TOKENS...

43  If B = 51 Then
        TR = TR+1
        RF = 1
        RS = 1
          *   SET BOTH FLAGS AND TOTAL-COUNT ON "REM"

44  If B = 2 Then
        FF = 1
          *   A "FOR" IS STARTED

45  If B = 69 Then
        CF = 1
        G$ = G$+T$(B)
         Goto 54
          *   FORCE A NEW LINE AFTER PRINTING A "THEN"

46  If B = 4 Then
        DF = 1
          *   "DATA" STATEMENT BEGUN; WILL AFFECT INDENTING LATER
47  If B = 43 Then
        Gosub 2
        Goto 34
          *   IGNORE A "LET" (IT IS A NULL STRING IN DATA STATEMENT
              LINE #88)

48  If B = 3 Then
        FS = FS-1
        If FS<0 Then
            FS = 0
              *   "NEXT" TOKEN REMOVES A "FOR" LOOP INDENT

49  G$ = G$+T$(B)
     Gosub 2
     Goto 34

50        *   ADD EXTRA INDENT FOR EACH SPLIT LINE, LIMITING FOR
              LINE-UP OF "REM" AND "DATA" PRINT-OUTS

51  SF = 0
    RS = RS+1
    If RS>2 Then
        RS = 2
52  If DF And RS>1 Then
        RS = 1

53        *   GET TOTAL INDENT SPACES FOR PRINT LINE PLUS LOW-LIMIT FOR
              SPLIT-POINT ("E")

54  K = IM*(FS+CS+RS)
    E = K+13
    If K>0 Then
        G$ = Left$(BB$,K)+G$
```

(continued)

Print Line Formatting

The next part of the program sets up indent spaces and splits over-long print lines on a selected character. Splitting is done on ASCII characters since gather string G$ contains only ASCII values on entrance at line 54.

FOR, REM, and IF spacing counters are added at line 54, multiplied by IM (default value of four), and inserted ahead of G$. Temporary variable D is an indicator to insert the line number on the first statement. G$ is set into C$, then tested for length at line 59. If the C$ string length is too long, it is split with the right side remainder replacing the former contents of G$.

Splitting has two priorities. The first priority split occurs at the right-most available space, if it is not a DATA statement. The second priority is an arithmetic operator character (ASCII, not token) or comma; DATA statements split only on commas. While the second priority choice seems arbitrary, it is convenient in terms of ASCII values.

Splitting character search is right to left, beginning with the last available print line column determined by LL. Left limit is determined by temporary variable E (line 54). The original program had an undesired zero left limit; a few print lines were endless blanks!

Another undesired condition occurred with spaces in long strings or PRINTs going beyond the right limit. There was no way to determine if a space existed in the printout. This is solved by lines 74 and 75 adding an underline at the right-most space of the first line, or left-most space of the next line.

Final Print and Cleanup

Every new line calls the TEST PAGE subroutine. This determines if a new page is called for and, if so, prints the "continued" reminder at the bottom, form-feeds, then prints a header on the next page.

Deciding on a one-statement-per-line format gave us the possibility of one or more unnumbered statements on the next page. Holding the readability rule, we decided on placing the next-page line number in brackets (seldom used in Applesoft) while holding the number print justification. Lines 13 and 14 take care of this. An early version used two colons between the number and statement but conflicted with Backus-Naur notation.

Uncompleted split lines jump to line 51 for extra indents. A remark allows one extra indent count to line up the remark second line with first line text. The REM symbol used here takes four columns or one default indent space. The DATA declaration single indent (line 52) seemed to be most readable.

Separate flag and counter variables on FOR and IF statements allow for concatenation in one line number of source code *and* the global or local indenting in printout. Local indenting of conditionals is reset on a new line number but global indenting of FOR loops is decremented only on a NEXT token at line 45.

A new source code line number is begun only when the program byte contains the end-of-line null.

Ending it All

Applesoft indicates the end of a listing by three successive nulls. This would appear as a zero line number — a second zero line number, since LISZTER begins with line number zero. This second zero line number falls through the IF in line 16 to begin optional statistical printouts at lines 18, 20, and 21.

Line 23 disables all Apple peripherals by "PR#0", resets screen width to normal by "POKE 33,40", and indicates a finish on the screen. The print command at line 106 allowed the screen to be active at all times even though lower case characters appear as nonsense on a standard Apple.

The POKEs in line 25 reset the start and end pointers to their original values prior to the EXEC file command. Variable and array space pointers are also reset permitting the user to RUN the program after LISZTing.

Optional Starting Prompts

The "RUN 23" notice in line 94 should remain until the user is very familiar with LISZT. It is the only way to restore start and end pointers after a RESET. Address locations in line 95 are optional, useful only with very long programs.

Page length, left margin, and indent spacing are useful only if different paper is used. If available, different vertical printer spacing could be added to

Listing 2 *(Continued)*

```
55       *   ADD LINE NUMBER OR EQUIVALENT-SPACE BLANK

56   If Not D Then
         C$ = N$+G$
57   If D Then
         C$ = LB$+G$

58       *   TEST FOR LONG LINE, SPLIT IF NECESSARY

59   K = Len(C$)-LL
     If K<1 Goto 73
         *   NOT A SPLIT LINE

60   G$ = Right$(C$,K)
     C$ = Left$(C$,LL)
     SF = 1
     If DF Goto 65

61       *   START SPLIT WITH A SPACE FIRST IF NOT "DATA"

62   D = LL
63   If Mid$(C$,D,1) = S$ Goto 71
64   D = D-1
     If D>E Goto 63
65   D = LL
         *   SPLIT NEXT AT ARITHMETIC OPERATOR OR COMMA

66   K = Asc(Mid$(C$,D,1))
     If K<42 Or K>47 Goto 69
67   If DF And K = 44 Goto 71
         *   "DATA" STATEMENTS SPLIT ONLY ON COMMAS

68   If Not DF And K<>46 Goto 71
         *   OTHER STATEMENTS SPLIT BY ALL BUT PERIOD

69   D = D-1
     If D>E Goto 66
70    Goto 73
         *   FALL-THROUGH INDICATES END-OF-PRINT-LINE SPLIT
71   K = LL-D
     If K>0 Then
         G$ = Right$(C$,K)+G$
         C$ = Left$(C$,D)

72       *   TEST PAGE LINE-COUNT, INSERT SPACES AS ALLOWED, THEN
             PRINT AT LINE #76.  NOTE: SINGLE CHARACTER SET PRINTERS
             SHOULD USE ONLY "PRINT M$;C$" BEFORE "K = FRE(0)" IN LINE
             #76.

73    Gosub 6
     K = Len(C$)
     If SF = 0 Or K<2 Or RF Then
         76
74   If Mid$(C$,K,1) = S$ Then
         C$ = Left$(C$,(K-1))+Chr$(95)
             *   PUT A TRAILING UNDERLINE IN PLACE OF THE LAST SPACE
                 AS A MARKER FOR THE LEFT-HAND STRING

75   If Len(G$)>2 And Left$(G$,1) = S$ Then
         G$ = Chr$(95)+Right$(G$,(Len(G$)-1))
             *   PUT A LEADING UNDERLINE IN PLACE OF THE FIRST SPACE
                 OF RIGHT-HAND STRING AS A MARKER

76    Gosub 111
     K = Len(C$)
     Print M$;Left$(C$,8);
      Gosub 112
     Print Right$(C$,(K-8))
     K = Fre(0)
     If SF Then
         D = 1
          Goto 51
             *   PRINT REST OF A SPLIT LINE

77   QF = -1

     RS = 0
     DF = 0
     If FF Then
         FS = FS+1
         FF = 0
78   D = 0
     If CF Then
         CS = CS+1
         CF = 0
```

(continued)

the page length prompt at line 98. A variable left margin requires the BB$ string to be slightly longer than one-half print line width.

We recommend that you retain the inverse video reminder at line 102. Concentration on program development makes us forget the right buttons to push at crucial moments!

Final Thoughts

A "REM-less" version of LISZTER is about 3.9K long and will run in 5.5K of free memory. Disk operations are not required after the intitial EXEC LISZT command.

Hesitation in execution occurs only in parsing long character lines. LISZTER's line 76 takes about 20 seconds to gather, split, and begin printing. The 256-byte string maximum has not yet been reached, including one LISZTing over 30 print pages.

Lack of concatenation character does not seem to hamper reading. Those familiar with the interpreter syntax will know it is always there. Statement separation is easier to understand and is improved further with indenting.

Thanks are due to Cliff Bruhn, Dennis Kaloi, Sterling Tate, Wes Ten, and Bob Keene of Candid Computers for their trial runs, comments, and suggestions.

References

1. "FP LISTER," Mark Capella, *CALL-A.P.P.L.E.*, May, 1980.
2. "Apple BASIC Prettyprinter," Michael Keith, *Microcomputing* August, 1981, pp.110, 113.
3. "Program Listing Formatter," Robert C. Clardy, M. Capella, J. Morrisset, C. Anson, V. Golding, *CALL-A.P.P.L.E. In Depth*, Number One (September 1981), pp. 102, 105.
4. *Applesoft Reference Manual*, Apple Computer Incorporated.
5. *Apple II Disk Operating System Reference Manual*, Apple Computer Incorporated.
6. "The Inspector," Omega Microware, Inc., (one of several "zappers" available).

Listing 2 *(Continued)*

```
79  SF = 0
    G$ = ""
    If B = 0 Then
        CS = 0
         Goto 16
            *   GET ANOTHER PRINT LINE IF NOT E-O-L NULL, ELSE FALL
                THROUGH AND GET ANOTHER LINE NUMBER

80   Gosub 2
    D = 1
     Goto 34

81      *   INITIALIZATION OF VARIABLES

82  Dim T$(107),H$(4)

83      *   INITIAL VARIABLE SETTING HAS AN 80-CHARACTER WIDE PRINT
            LINE AND 60-LINE PAGE LENGTH (INCLUDING HEADER, EXCLUDING
            'CONTINUED' INDICATOR); CHANGE LL AND LP AS DESIRED FOR
            OTHER FORMAT SIZE.
84      *   THE "P=2048" IN LINE #85 ASSUMES A NORMAL APPLESOFT ROM
            START AT DECIMAL ADDRESS 2049.  CHANGE FOR APPLESOFT IN
            RAM.

85  P = 2048
    B = 0
    RS = 0
    CS = 0
    FS = 0
    RF = 0
    CF = 0
    FF = 0
    DF = 0
    SF = 0
    QF = -1
    LL = 80
    LP = 60
    IM = 4
    E = 0
    TN = 0
    TS = 0
    TR = 0
    S$ = " "
86  C$ = ""
    N$ = ""
    G$ = ""
    M$ = ""
    H$(0) = ""
    LB$ = "        "
    BB$ = "
                                                  "
87  Data "End","For ","Next ","Data ","Input ","Del ","Dim ","Read ",
         "Gr","Text","Pr# ","In# ","Call ","Plot ","HLin ","VLin ",
         "HGr2","HGr","HColor = ","HPlot ","Draw ","XDraw ","HTab ",
         "Home","Rot = "
88  Data "Scale = ","ShLoad","Trace","NoTrace","Normal","Inverse",
         "Flash","Color = ","Pop","VTab ","Himem : ","Lomem : ",
         "OnErr ","Resume","Recall ","Store ","Speed = ",""," Goto ",
         "Run","If ","Restore","& "," Gosub ","Return"
89  Data "*    "
         *   ^ CHANGE "REM" TOKEN WORD INDICATOR AS DESIRED

90  Data "Stop","On ","Wait ","Load ","Save ","Def ","Poke ","Print ",
         "Cont","List ","Clear","Get ","New","Tab("," To ","Fn ",
         "Spc("," Then "," At ","Not "," Step ","+","-","*"
91  Data "/","^"," And "," Or ",">"," = ","<","Sgn","Int","Abs","Usr",
         "Fre","Scrn(","Pdl","Pos","Sqr","Rnd","Log","Exp","Cos","Sin",
         "Tan","ArcTan","Peek","Len","Str$","Val","Asc","Chr$","Left$",
         "Right$","Mid$"
92  For K = 1 To 107
         Read T$(K)
    Next

93       *   SCREEN PROMPTS AND ALTERNATE LISTING CONSTANTS

94  Home
    VTab 3
    Flash
    Print " RUN 23 ";
    Normal
    Print " RESTORES ORIGINAL AFTER RESET"
         *   'RUN 23' RESTORES POINTERS FOR PROGRAM START AND END TO
             ORIGINAL VALUES AND RESETS SCREEN
```

(continued)

Listing 2 *(Continued)*

```
95  Print
    Print "START OF PROGRAM LISTED: ";Peek(209)*256+Peek(208)
    Print "   END OF PROGRAM LISTED: ";Peek(211)*256+Peek(210)
    Print "          END OF 'LISZTER': ";Peek(176)*256+Peek(175)

96       *    ABOVE IS OPTIONAL, CHECKS TO SEE IF THE 'LISZT' EXEC-FILE
              IS OPERATING PROPERLY

97  Print
    Input "PROGRAM NAME: ";H$(1)
    Input "   PROGRAMMER: ";H$(2)
    Input "          DATE: ";H$(3)
    Print
98  Print
    Print "PAGE LENGTH IS 60 LINES, WANT OTHER?"
    Get H$(0)
    If H$(0) = "Y" Then
         Input "  PAGE LENGTH: ";LP
         If LP>62 Goto 98
              *   LIMIT TO 11" LENGTH AND HEADER START-POSITION; CAN
                  CHANGE WITH SMALLER-SPACING PRINTERS.

99  Print
    Print "NO LEFT MARGIN, WANT ONE ?"
    Get H$(0)
    If H$(0) = "Y" Then
         Input "  MARGIN SPACES: ";K
         If K>0 And K<49 Then
              M$ = Left$(BB$,K)
              LL = LL-K
                   *   MARGIN & LINE-LENGTH UNTOUCHED ON WRONG INPUT,
                       REMAINS AT DEFAULT VALUE

100 Print
    Print "INDENT SPACING = 4, WANT OTHER ?"
    Get H$(0)
    If H$(0) = "Y" Then
         Input "  SPACING: ";IM
         If IM<0 Or IM>12 Goto 100

101      *    REMINDER FOR PRINTER SET-UP
102 Home
    Inverse
    Print " SET PAPER TO TOP OF FORM "
    Print "           THEN            "
    Print "      TURN ON PRINTER      "
    Normal
    Print
    Get H$(0)

103      *    SET SCREEN WIDTH, TURN ON PROPER PORT

104 Home
    Poke 33,30
    Pr# 1

105      *    SET-UP FOR EPSON MX-80 PRINTER WITH ORANGE MICRO
              'GRAPPLER' OR CENTRONICS-COMPATIBLE PARALLEL INTERFACE
              CARD.  CCS CARD MUST ADD 'CHR$(9)"K"' TO REMOVE EXTRA
              LINE FEED.

106 Print Chr$(9)"82N"Chr$(9)"I"

107      *    RESERVED LINE FOR OPTIONAL PRINTER CONTROL
108      *    CHR$(9) = "CONTROL-I"

109 LC = 6
    PC = 1
     Gosub 11
     Goto 16

110      *    MX-80 ITALICS/STANDARD CHARACTER SET SWITCHING
              SUBROUTINES (APPLIES ONLY TO "GRAFTRAX"-AUGMENTED
              PRINTERS)

111 Print Chr$(27)"5";
    Return
         *    ESC-5 IS STANDARD SET

112  Gosub 111
    If RF Then
         Print Chr$(27)"4";
              *   ESC-4 IS ITALICS SET USED FOR "REM"S

113 Return
```

(continued)

Problems You May Encounter with LISZT with Strings

1. A colon ending a line causes a stop and 'error at line 76' display. The best solution is to use a line editor program or keyboard to correct the program line to remove the extraneous byte. Usually appears to be a 'forgotten' removal during program editing.

2. A double colon starting a line causes LISZTER to think the first colon is a REM, but the second colon causes reversion to gathering tokens and characters in the usual manner. Using an italics set on the printer will make this line look like a REM splat, but has both upper and lower case contents. Best solution is to edit out the extra colons.

3. A statement ending nested FOR loops such as "NEXT J,K,L" executes in Applesoft as if they were three separate NEXT statements. Since LISZTER will only recognize *one* NEXT token, all following lines will retain the FOR-NEXT indent(s) for the remainder of printout.

   We don't have a simple solution for this — yet. Changing the program to "NEXT J:NEXT K:NEXT L" will add only two bytes and bring the left margin back to normal. The two added bytes are the NEXT tokens; concatenation colons take the place of the commas.

4. On any mid-printout deliberate stop, such as RESET, you must key in RUN 23 to restore the program start and end pointers. Failure to do so may attach LISZTER to the program being listed.

Richard F. Searle is manager of software and control systems at an aerospace corporation and a consultant on the application of solar energy. You can write to Mr. Searle at 4511 Adam Road, Simi Valley, California 93063.

Donald Cohen is an electronics engineer specializing in microprocessor control of aircraft emergency power systems. He is also a programming consultant and partner of Donald Cohen Associates. Contact Mr. Cohen at Donald Cohen Associates, 4613 Wolfe Way, Woodland Hills, California 91364.

Leonard H. Anderson is a hardware and systems engineer, who uses his Apple II for number crunching (when not playing games). He is also a contract writer and currently an Associate Editor with *Ham Radio* Magazine. Address correspondence to 10048 Lanark Street, Sun Valley, California 91352.

Listing 2 *(Continued)*

```
  114    *        "LISZTER"
  115    *    Working Program to
  116    *        re-format
  117    *    APPLESOFT Programs for
  118    *         Printing
  119    *
  120    *            BY
  121    *    LEONARD H. ANDERSON
  122    *
  123    *    Version 4.1.3, 7/24/81
  124    *     (lower-case version)
  125    *     MX-80 & "GRAFTRAX"
  126    *      (ITALICS ON REMS)
  127    *
  128    *   DESCRIPTION OF VARIABLES:
  129    *
  130    *   B     PROGRAM BYTE DECIMAL VALUE
  131    *   BB$   'BIG BLANK' STRING OF 48 SPACES
  132    *   CF    "IF" FLAG: 1 = "IF" STARTED; 0 = NO "IF"
  133    *   CS    "IF" (CONDITIONAL) INDENT SPACE COUNTER
  134    *   C$    CHARACTER AND TOKEN STRING TO BE PRINTED
  135    *   D     'DIRECTION', A TEMPORARY
  136    *   DF    "DATA" FLAG (ALLOWS SPLIT ON COMMA ONLY)
  137    *           1 = "DATA" EXISTS ON LINE; 0 = NO "DATA"
  138    *   E     TEMPORARY, PARTLY FOR SPLIT-LINE LIMITS
  139    *   FF    "FOR" FLAG: 1 = "FOR" STARTED; 0 = NO "FOR"
  140    *   FS    "FOR" INDENT SPACING COUNTER
  141    *   G$    'GATHER' STRING TO BUILD STATEMENT LINE
  142    *   H$    HEADER ARRAY FOR PAGE TITLE
  143    *   IM    INDENT SPACE MULTIPLIER
  144    *   K     TEMPORARY
  145    *   LB$   'LITTLE BLANK' STRING OF 8 SPACES
  146    *   LC    LINE COUNTER FOR PAGINATION TEST
  147    *   LL    LINE-LENGTH (WIDTH) CONSTANT
  148    *   LP    LINES-PER-PAGE CONSTANT
  149    *   M$    LEFT MARGIN SPACING STRING
  150    *   N$    LINE NUMBER STRING
  151    *   P     POINTER TO PROGRAM BYTE (DECIMAL VALUE)
  152    *   PC    PAGE COUNTER FOR HEADER ON EACH PAGE
  153    *   QF    QUOTE FLAG TO ALLOW/DISALLOW COLON PRINTING
  154    *           -1 = NO QUOTE OR SECOND QUOTE OF PAIR EXISTS
  155    *           +1 = FIRST QUOTE OF PAIR EXISTS, ALLOW COLONS
  156    *   RF    "REM" FLAG: 1 = "REM" STARTED; 0 = NO "REM"
  157    *   RS    "REM" INDENT SPACING COUNTER
  158    *   SF    SPLIT-LINE FLAG; SET IF PRINT LINE MUST BE SPLIT
  159    *   S$    SINGLE SPACE STRING
  160    *   TN    TOTAL LINE NUMBER COUNTER
  161    *   TR    TOTAL REMARK-STATEMENT COUNTER
  162    *   TS    TOTAL STATEMENT COUNTER
  163    *
  164    *   AN EXAMPLE OF INDENTS ON NESTED "FOR" LOOPS:

  165  For J = 1 To 25
  166      For K = J To 26
  167          If MT(J,K) = 0 Goto 170
  168          For L = J To K
                   If MT(J,K)<>0 Then
                       MT(K,L) = MT(K,L)-(MT(J,K)*MT(J,L))
                       *     BEGINS WITH "LET MT(..."

  169          Next L
  170      Next K
  171  Next J

  172    *
 1790    *       /I/    /G/    /I/
18000    *   THE PRECEDING LINE CONTAINED TWO CONTROL-I CHARACTERS
             SEPARATED BY A CONTROL-G (BELL).

       End of Listing

Program Length = 10061 Bytes,  Total of 175 Line Numbers

271 Total Non-Rem Statements,  119 Total Remarks

END
```

MICRO

Apple Graphics for Okidata Microline 80

by Charles F. Taylor, Jr.

Programs are given, in Apple/UCSD Pascal and 6502 assembly language, to dump the Apple high-resolution graphics screen(s) to an Okidata Microline 80 printer. This article should also be of interest to owners of the Epson MX-80 printer because graphics on the two printers are implemented similarly.

These Apple Graphics routines require:
- Apple
- Okidata Microline 80, 82A, or 83A
- Epson MX-80

Pascal version requires:
- Apple Pascal system

When I purchased my Okidata Microline 80 dot matrix impact printer, I was attracted to its relatively low price, the 200,000,000-character print head warranty, and the flexible form handling (friction and pin feed). I really didn't consider its graphics, which were advertised as "TRS-80-compatible."

After I'd had the printer for a while, I decided to take another look at its graphics capabilities. The basic graphical unit for the Microline 80 is the graphics character. Each graphics character may be thought of as a 3-row by 2-column matrix, as depicted in figure 1.

The individual elements of the graphics character are numbered 1-6 as in figure 1. Each element of the character may be "on" (black) or "off" (white), which means that there are 2 to the 6th power, or 64, possible distinct graphics characters. An element that is "on" is represented by what appears (under magnification) to be a 3 by 3 matrix of dots. The total graphics character, then, is a 9-row by 6-column matrix. This is achieved with a 7-pin print head by making two passes for every line which contains a graphics character, advancing the paper slightly between passes.

With the printer set for 16.5 characters per inch on an 8-inch line, the horizontal resolution is 0.030 inches (0.77 millimeters). At eight lines per inch, the vertical resolution is 0.042 inches (1.06 millimeters). In other words, the smallest "dot" that can be printed is an element of a graphics character which is a rectangle 0.030 inches wide by 0.042 inches high.

Each graphics character is sent to the printer as a single byte with the high-order bit (bit 8) set (1). Bit 7 may be either 0 or 1. Bits 1 through 6 are set (1) or clear (0) as the correspondingly numbered element of the graphics character is "on" or "off." (See figure 1 again.)

Software could be written to utilize these graphics characters directly. This would include, as a minimum, routines to set and clear individual elements of graphics characters and to draw straight lines between any two points. Because Applesoft BASIC and Apple/UCSD Pascal each provide these graphics primitives for use with the Apple high-resolution screen, a better approach is to develop a utility program to dump, point by point, the contents of the hi-res screen to the printer. This was the approach I took, first in Pascal, then in 6502 assembly language. (The latter version can be called from BASIC programs.)

The basic unit of Apple hi-res graphics is of course the "pixel" or dot. The hi-res screen is organized as a 192-row by 280-column matrix of individually addressable pixels. The display is bit-mapped; that is, there exists a mapping between each pixel on the screen and a bit somewhere in memory.

There were three principal problems to be resolved in designing the program: the first problem was how to address the bit representation of each pixel in order to determine whether it is on or off. The second problem was to decide whether to print the screen image horizontally or vertically on the printer. Finally, a means had to be found to map six pixels to each graphics character.

Pascal Solution

The solution in Pascal is presented first because it is simpler. This is because of the existence of the SCREENBIT function, which is provided as part of the TURTLEGRAPHICS unit. SCREENBIT(x,y) is a Boolean function which returns the value TRUE if pixel

***Figure 1:* Microline 80 Graphics Character**

1	2
3	4
5	6

***Figure 2:* Model of High Resolution Screen (6 x 6)**

	0	1	2	3	4	5	
5	*						0
4	*	*					1
3	*		*				2
2	*			*			3
1	*				*		4
0	*					*	5
	0	1	2	3	4	5	

(x,y) is on (not black), and FALSE if it is off (black). This makes the first problem cited above almost trivial.

Since only a maximum of 132 characters can be printed on a line (16.5 characters per inch times 8 inches), and each character is two elements wide, the maximum number of pixels which can be presented on a printed line is 264. Because the Apple hi-res screen is 280 pixels wide, two choices are possible: (1) print the screen image vertically on the printer, 192 elements across and 280 down; or, (2) print the screen image horizontally, but print only 264 of the 280 columns. The former choice was made for the Pascal version and the latter for the 6502 assembly language version.

The Pascal program is shown in listing 1. The main program queries the user as to whether to print all or a specified portion of the screen. Procedure SETUP handles the details of turning on the printer and selecting the proper print size and vertical spacing. Procedure TURNOFF turns the printer off again. The real work is done in the procedure called SCREENDUMP.

How the algorithm works can best be illustrated by example. Assume that the Apple hi-res screen consists of a 6-row by 6-column grid as shown in figure 2, and that an arbitrary pattern has been plotted on it. An "*" is used to indicate which grid elements are "on." In Pascal (as opposed to BASIC) the origin is at the lower left corner of the grid, so the numbers along the left side refer to the row of y-coordinates. The numbers along the right side will not be needed until later.

The Pascal program will reduce this grid to two lines of three graphics characters each. The first line will represent columns (x-coordinates) 0, 1, and 2 of the grid and the second line columns 3, 4, and 5. Remember that the image on the printer will be rotated 90 degrees, so that columns on the screen correspond to rows on the printer and *vice versa*.

The first graphics character of the first line will represent rows (y-coordinates) 0 and 1 of the columns 0, 1, and 2. We may imagine the grid of figure 1 superimposed on the grid of

Table 1

Graphics Characters Bit	Screen Grids Position
1	(0,0)
2	(0,1)
3	(1,0)
4	(1,1)
5	(2,0)
6	(2,1)

Listing 1

```
PROGRAM PRINTSCREEN;

  (*S+*)
  (* DUMPS ENTIRE PASCAL SCREEN *)
  (* TO OKIDATA MICROLINE 80    *)
  USES TURTLEGRAPHICS;

VAR XMIN, XMAX, YMIN, YMAX : INTEGER;


PROCEDURE SCREENDUMP(XMIN, XMAX, YMIN, YMAX : INTEGER);


  (* DUMPS PASCAL GRAPHICS TO *)
  (* OKIDATA MICROLINE 80     *)

  VAR H, I, J, K : INTEGER;
      LINE     : ARRAY [1..96] OF CHAR;
      BIT      : ARRAY [1..6] OF 0..1;
      OKI      : TEXT;

  PROCEDURE SETUP;

    BEGIN
      (* OPEN PRINTER FILE *)
      REWRITE (OKI,'PRINTER:');
      (* SET CENTRONICS CARD FOR 132 COLS *)
      WRITELN (OKI,CHR(9),'132N');
      (* SET PRINTER FOR 16.5 CPI & 8 LPI *)
      WRITELN (OKI,CHR(29),CHR(27),'8',CHR(27),'B');
    END;  (* SETUP *)

  PROCEDURE TURNOFF;

    (*  RESETS OKIDATA  *)

    BEGIN
      WRITELN (OKI,CHR(30),CHR(27),'6',CHR(27),'A');
    END;  (* TURNOFF *)

  FUNCTION GCHAR : CHAR;

    (* RETURNS GRAPHICS CHARACTER *)
    (* DEFINED BY BIT ARRAY       *)

    VAR NR : 0..255;
        POWER, I : INTEGER;

    BEGIN
      POWER := 1;
      NR := 128;
      FOR I := 1 TO 6 DO
        BEGIN
          NR := NR + POWER* BIT[I];
          POWER := 2*POWER;
        END;
```

Listing 1 *(Continued)*

```
      GCHAR := CHR(NR)
    END;  (* GCHAR *)


  BEGIN  (* PROCEDURE SCREENDUMP *)
    SETUP;
    FOR K := 1 TO 96 DO
      LINE [K]  := CHR(128);
    I := XMIN;
    REPEAT
      J := YMIN;
      K := 1 + YMIN DIV 2;
      REPEAT
        FOR H := 1 TO 6 DO BIT[H]   := 0;
        IF SCREENBIT(I,J)       THEN  BIT[1]  := 1;
        IF SCREENBIT(I,J+1)     THEN  BIT[2]  := 1;
        IF SCREENBIT(I+1,J)     THEN  BIT[3]  := 1;
        IF SCREENBIT(I+1,J+1)   THEN  BIT[4]  := 1;
        IF SCREENBIT(I+2,J)     THEN  BIT[5]  := 1;
        IF SCREENBIT(I+2,J+1)   THEN  BIT[6]  := 1;
        LINE[K] := GCHAR;
        K := K + 1;
        J := J + 2
      UNTIL J+1 > YMAX;
      FOR K := 1 TO 96 DO
        BEGIN
          WRITE (OKI,LINE [K]);
          LINE [K]  := CHR(128)
        END;
      WRITELN(OKI);
      I := I + 3
    UNTIL I+2 > XMAX;
    TURNOFF;
  END;  (* SCREENDUMP *)


BEGIN  (* MAIN PROGRAM *)
  REPEAT
    WRITE ('FIRST COLUMN TO PRINT (0..279):');
    READLN(XMIN)
  UNTIL (XMIN >= 0) AND (XMIN <= 279);
  REPEAT
    WRITE ('LAST COLUMN TO PRINT  (XMIN..279):');
    READLN(XMAX)
  UNTIL (XMAX >= XMIN) AND (XMAX <= 279);
  REPEAT
    WRITE ('FIRST ROW TO PRINT (0..191):');
    READLN(YMIN)
  UNTIL (YMIN >= 0) AND (YMIN <= 191);
  REPEAT
    WRITE ('LAST ROW TO PRINT  (YMIN..191):');
    READLN (YMAX)
  UNTIL (YMAX >= YMIN) AND (YMAX <= 191);
  SCREENDUMP(XMIN, XMAX, YMIN, YMAX);
END.
```

figure 2, with figure 1 rotated 90 degrees (counter-clockwise). Thus bits of the graphics characters correspond to screen grids as shown in table 1. In this case bits 3, 4, 5, and 6 will be 0 and bits 1 and 2 will be 1. Bit 7 will (arbitrarily) be 0, and bit 8 will be 1, as discussed earlier. The resulting graphics character is, therefore, binary 10000011 ($83 or decimal 131).

The next graphics character of the line will be constructed from grid elements (0.2), (1,2), (2,2), (0,3), (1,3), and (2,3), which correspond to, respectively, bits 1, 3, 5, 2, 4, and 6. In this case bits 1, 2, and 6 will be 1 and bits 3, 4, and 5 will be 0. The resulting graphics character is binary 10100011 ($A3, or decimal 163). The third and final graphics character of the line will be binary 10000111, ($87, or decimal 135).

At this point the program has constructed the first full line of graphics characters. Using the CHR function, the computed decimal values have been converted to their character equivalents and are stored in the array LINE. Now that LINE is full, it is sent to the printer, one character at a time, and is followed by the usual carriage return and line feed (WRITELN).

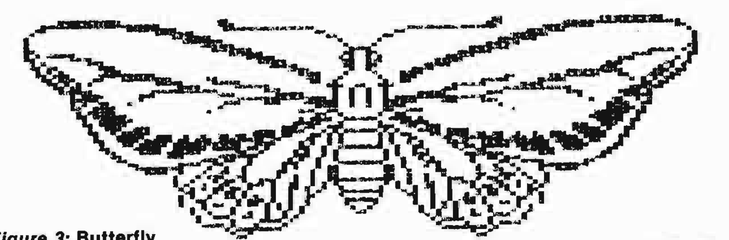


Figure 3: **Butterfly**

The program then constructs the next line of graphics characters from columns 3, 4, and 5 of the screen grid. These characters will be, in decimal notation, 152, 129, and 128, respectively.

The procedure described above is carried out by the procedure SCREENDUMP. The function GCHAR uses simple arithmetic to convert the binary representation of the character to decimal. Then it uses CHR to convert the decimal value to a character. Experienced Pascal programmers may notice that I could have accomplished the binary-to-character conversion directly using a free-union variant record. That technique would have been faster and more efficient, but less clear. Readers who wish to pursue this topic should refer to an article by David Casseres of Apple Computer Inc., which appeared in the October 1981 issue of *BYTE* magazine.

Figure 3 is an example of output produced by this program. The butterfly image was created on the screen by a demonstration program furnished with the Apple Pascal system and then printed on the Microline 80 by this program.

Assembly Language Solution

The most difficult part of the 6502 assembly language solution was to develop an algorithm to step through memory, addressing of each pixel's bit representation in the proper sequence. (Recall that this was doen for us in Pascal by SCREENBIT.) The task is complicated by the fact that, for various reasons, Apple chose to represent the hi-res screen in memory in what appears to be a rather peculiar sequence. The mapping used is documented in the *Apple II Reference Manual* and was the subject of a 1978 article in MICRO (7:43) by Andrew H. Eliason. Rather than reiterate the details here, I have chosen to present a short Applesoft BASIC program (listing 2) which prints out the beginning address (in decimal and in hex) of each of the 192 rows of hi-res screen 1. (To get the corresponding values for screen 2, change line 100.) The 280 pixels of each row are represented by seven bits of each of the 40 bytes beginning at the location given. The program prints a screen, then prompts the user to press the space bar before running another screen.

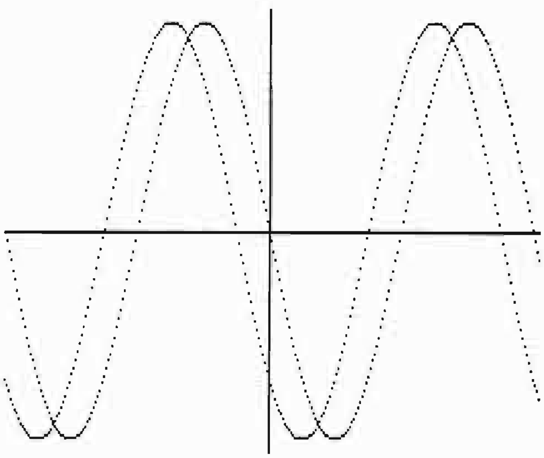
As mentioned above, I decided to represent the screen horizontally on the printer in this version (which considerably simplifies the arithmetic). This means that only 264 columns of the hi-res screen could be printed. The first 264 were arbitrarily selected.

The 6502 assembly language program is shown as listing 3. Instructions for its use are contained in the program's introductory comments. The printer interface I used was the Apple

Listing 2

```
10  REM  PROGRAM TO DEMONSTRATE
20  REM  THE SEQUENCE OF STORAGE
30  REM  LOCATIONS USED BY THE
40  REM  APPLE HI-RES SCREEN
50  REM
60  REM  BY C.F. TAYLOR, JR.
70  REM  JULY 24, 1981
80  REM
90  HOME
95 DIGIT$ = "0123456789ABCDEF"
100 B0 = 8192: REM  FOR SCREEN 1
110 SLIN = 0: REM  SCREEN LINE NR
120  FOR I = 1 TO 3
130 B1 = B0
140  FOR J = 1 TO 8
150 CLIN = B1
160  FOR K = 1 TO 8
170  GOSUB 500: REM  CONVERT TO HEX
180  PRINT SLIN,CLIN,HL$
190 SLIN = SLIN + 1
200 CLIN = CLIN + 1024
210  IF SLIN -  INT (SLIN / 23) * 23 <  > 0 THEN  GOTO 240
220  PRINT "PRESS (SPACE) TO CONTINUE";
230  GET A$: PRINT
240  NEXT K
250 B1 = B1 + 128
260  NEXT J
270 B0 = B0 + 40
280  NEXT I
290  END
500 X = CLIN: REM  SUBROUTINE CONVERT TO HEX
510 HL$ = ""
520 HL$ =  MID$ (DIGIT$,X -  INT (X / 16) * 16 + 1,1) + HL$
530 X =  INT (X / 16)
540  IF X > 0 THEN  GOTO 520
550 HL$ = "$" + HL$
560  RETURN
```

***Figure 4:* Sine Curves**



Centronics Parallel Interface; some modifications will likely be required for use with other interface cards. What is critical is that bit 8 must be controllable (high for graphics, low for text). Some interfaces may not use bit 8 at all, or may force it low. The Epson interface board has bit 8 wired to ground, but a jumper is provided for changing this. If you do modify the setting of this jumper, however, you will have to make some other provision for forcing bit 8 low for text. My recommendation is to replace the jumper with a single-pole, double-throw switch. This is, in effect, what I have done to my Apple Centronics Interface card.

How the assembly language program works can also be illustrated by example. Refer again to figure 2. This time we will use the row numbers (y-coordinates) along the right edge, recalling that BASIC refers to the upper left corner as (0,0). This time rows 0, 1, and 2 will be used to construct the first line of graphics characters and rows 3, 4, and 5 the second line.

The first graphics character will therefore represent columns 0 and 1 of rows 0, 1, and 2. We may imagine the grid of figure 1 superimposed on the grid of figure 2, but this time without rotation. Thus bits 1-6 of the graphics character will represent, respectively, the screen grid positions (0,0), (1,0), (0,1), (1,1), (0,2), and (1,2). Bits 1, 3, 4, and 5 will be 1 and bits 2 and 6 will be 0. As before, bit 7 will be 0 and bit 8 will be 1. Therefore, the first graphics character is binary 10011101 ($9D, or decimal 157).

For the second graphics character, bits 1-6 correspond, respectively, to coordinates (2,0), (3,0), (2,1), (3,1), (2,2), and (3,2). Bit 5 is 1 and bits 1, 2, 3, 4, and 6 are 0. This translates to binary 10010000 ($90, or decimal 144). The third and last graphics character of the first line is binary 10000000 ($80, or decimal 128). The decimal values of the three graphics characters of the second line are 149, 130, and 164.

The algorithm illustrated in the BASIC program of listing 2 is used to find the beginning of each of three consecutive rows of the screen in memory. The bytes representing the pixels of these lines are then transferred to working buffers. (Only 38 bytes out of 40 are used because only 264 out of 280 columns are plotted.) The subroutines DUMP and DUMPY then extract the appropriate bits from the buffers and rotate them into a page zero location called CHAR. From there each is sent to the printer.

Figure 4 shows a typical plot of two out-of-phase sine curves. More sophisticated plots (3-D, etc.) are of course possible; anything you can put on the screen, you can print! The one limitation is color since the printer only prints black and white!

Execution time for the assembly language version is typically about six minutes. The Pascal version takes about 2.5 times as long to print a full screen.

Although the programs presented here were reasonably involved to write, they are simple to use. Best of all, they transform a fairly unsophisticated graphics capability on an inexpensive printer into a powerful graphics tool, rivaling printers costing several times as much.

The author may be contacted at 587F Sampson Lane, Monterey, CA 93940.

Listing 3

```
                 0010         ;HI-RES SCREEN DUMP
                 0020         ;APPLE II PLUS TO
                 0030         ;OKIDATA MICROLINE 80 VIA
                 0040         ;APPLE CENTRONICS PARALLEL INTERFACE
                 0050         ;
                 0060         ;WRITTEN BY C. F. TAYLOR, JR.
                 0070         ;24 JULY 1981
                 0080         ;
                 0090         ;INSTRUCTIONS:
                 0100         ;
                 0110         ;  SET HIMEM: 37760 BEFORE LOADING
                 0120         ;  CALL 37888 FOR HI-RES PAGE 1
                 0130         ;  CALL 37904 FOR HI-RES PAGE 2
                 0140         .BA $9400
                 0150         .OS
                 0160 ESCHR   .DE $638     ;LOCATIONS USED
                 0170 FLAGS   .DE $6B8     ;PRINTER INTERFACE
                 0180 PWDTH   .DE $4B8     ;ROM
                 0190 MODE    .DE $5B8
                 0200 DOS     .DE $03EA    ;DOS RE-ENTRY POINT
                 0210 CLIN    .DE $06      ;PAGE 0 LOCATIONS
                 0220 BUFF    .DE $08
                 0230 CHAR    .DE $E3
                 0240 DRIVER  .DE $C102    ;PRINTER DRIVER
                 0250         ;
                 0260         ;ENTRY POINT FOR HI-RES PAGE 1
                 0270         ;
9400- A9 20      0280 PAGE1   LDA #$20     ;INITIALIZE POINTERS
9402- 8D 97 95   0290         STA BASE+1
9405- 8D 99 95   0300         STA SAVBAS+1
9408- A9 40      0310         LDA #$40
940A- 8D 9D 95   0320         STA CURLIM+1
940D- 4C 1D 94   0330         JMP START
                 0340         ;
                 0350         ;ENTRY POINT FOR HI-RES PAGE 2
                 0360         ;
9410- A9 40      0370 PAGE2   LDA #$40     ;INITIALIZE POINTERS
9412- 8D 97 95   0380         STA BASE+1
9415- 8D 99 95   0390         STA SAVBAS+1
9418- A9 60      0400         LDA #$60
941A- 8D 9D 95   0410         STA CURLIM+1
                 0420         ;
941D- A9 00      0430 START   LDA #$00     ;COMMON POINTER VALUES
941F- 8D 96 95   0440         STA BASE
9422- 8D 98 95   0450         STA SAVBAS
9425- 8D 9B 95   0460         STA SLIN
9428- 8D 9C 95   0470         STA CURLIM
942B- 20 9D 94   0480         JSR SETUP    ;INITIALIZE PRINTER
942E- A9 02      0490         LDA #2       ;INITIALIZE BUFFER LINE NR
9430- 8D 9A 95   0500         STA LINE
                 0510         ;
9433- AD 96 95   0520 LOOPA   LDA BASE     ;CLIN := BASE
9436- 85 06      0530         STA *CLIN
9438- AD 97 95   0540         LDA BASE+1
943B- 85 07      0550         STA *CLIN+1
                 0560         ;
943D- 20 C4 94   0570 LOOPB   JSR BUFLIN   ;INCREMENT BUFFER LINE NR
9440- EE 9B 95   0580         INC SLIN     ;AND SCREEN LINE NR
9443- A0 25      0590         LDY #37      ;TRANSFER LINE TO BUFFER
9445- B1 06      0600 B1      LDA (CLIN),Y
9447- 91 08      0610         STA (BUFF),Y
9449- 88         0620         DEY
944A- 10 F9      0630         BPL B1
944C- AD 9A 95   0640         LDA LINE     ;TIME TO DUMP BUFFER?
944F- C9 02      0650         CMP #2
9451- D0 03      0660         BNE CONT
9453- 20 0A 95   0670         JSR DUMP
9456- A5 07      0680 CONT    LDA *CLIN+1
```

Listing 3 *(Continued)*

```
9458- 18         0690         CLC
9459- 69 04      0700         ADC #$4      ;INCREMENT MEMORY LOCATION
945B- 85 07      0710         STA *CLIN+1
945D- CD 9D 95   0720         CMP CURLIM+1 ;TIME TO ADJUST BASE?
9460- 90 DB      0730         BCC LOOPB    ;NO
9462- AD 9B 95   0740         LDA SLIN     ;SCREEN LINE NR
9465- C9 40      0750         CMP #64      ;READY TO SHIFT?
9467- F0 06      0760         BEQ SC0      ;YES
9469- C9 80      0770         CMP #128
946B- F0 02      0780         BEQ SC0      ;YES
946D- D0 15      0790         BNE SC1      ;NO
946F- AD 98 95   0800 SC0     LDA SAVBAS
9472- 18         0810         CLC
9473- 69 28      0820         ADC #$28     ;SHIFT BASE
9475- 8D 98 95   0830         STA SAVBAS
9478- 8D 96 95   0840         STA BASE
947B- AD 99 95   0850         LDA SAVBAS+1
947E- 8D 97 95   0860         STA BASE+1
9481- 4C 33 94   0870         JMP LOOPA
9484- C9 C0      0880 SC1     CMP #192     ;DONE?
9486- F0 14      0890         BEQ EXIT     ;YES
9488- 18         0900         CLC          ;ADJUST BASE
9489- AD 96 95   0910         LDA BASE
948C- 69 80      0920         ADC #$80
948E- 8D 96 95   0930         STA BASE
9491- AD 97 95   0940         LDA BASE+1
9494- 69 00      0950         ADC #0       ;ADD IN CARRY
9496- 8D 97 95   0960         STA BASE+1
9499- 4C 33 94   0970         JMP LOOPA
949C- 60         0980 EXIT    RTS          ;DONE
                 0990         ;
                 1000         ;SUBROUTINES FOLLOW
                 1010         ;
                 1020 SETUP   ;FIRST SETUP DRIVER
949D- A2 C1      1030         LDX #$C1     ;FOR SLOT 1
949F- A9 09      1040         LDA #$09     ;INITIALIZE DRIVER
94A1- 9D B8 06   1050         STA FLAGS,X  ;VIDEO OFF
94A4- A9 FF      1060         LDA #$FF
94A6- 9D B8 04   1070         STA PWDTH,X  ;PRINT WIDTH
94A9- 9D 38 06   1080         STA ESCHR,X  ;ESCAPE CHAR
94AC- A9 00      1090         LDA #0
94AE- 9D B8 05   1100         STA MODE,X   ;CLEAR 'AFTER ESC' MODE
94B1- 20 EA 03   1110         JSR DOS      ;REPLACE WITH 3 NOP'S
                 1120                      ;FOR CASSETTE SYSTEM
                 1140         ;NOW SETUP PRINTER
94B4- A9 1D      1150         LDA #$1D     ;SET 16.5 CPI ON PRINTER
94B6- 20 02 C1   1160         JSR DRIVER
94B9- A9 1B      1170         LDA #$1B     ;SET 8 LINES
94BB- 20 02 C1   1180         JSR DRIVER   ;PER INCH
94BE- A9 38      1190         LDA #$38     ;VERTICAL SPACING
94C0- 20 02 C1   1200         JSR DRIVER   ;ON PRINTER
94C3- 60         1210         RTS
                 1220         ;
94C4- EE 9A 95   1230 BUFLIN  INC LINE
94C7- AD 9A 95   1240         LDA LINE
94CA- C9 03      1250         CMP #3
94CC- D0 0E      1260         BNE BL1
94CE- A9 81      1270         LDA #L,LINE0 ;SET BUFFER LINE 0
94D0- 85 08      1280         STA *BUFF
94D2- A9 93      1290         LDA #H,LINE0
94D4- 85 09      1300         STA *BUFF+1
94D6- A9 00      1310         LDA #0
94D8- 8D 9A 95   1320         STA LINE
94DB- 60         1330         RTS
94DC- C9 01      1340 BL1     CMP #1       ;SET BUFFER LINE 1
94DE- D0 09      1350         BNE BL2
94E0- A9 A7      1360         LDA #L,LINE1
94E2- 85 08      1370         STA *BUFF
```

Listing 3 *(Continued)*

```
94E4- A9 93      1380          LDA #H,LINE1
94E6- 85 09      1390          STA *BUFF+1
94E8- 60         1400          RTS
94E9- A9 CD      1410 BL2      LDA #L,LINE2 ;SET BUFFER LINE 2
94EB- 85 08      1420          STA *BUFF
94ED- A9 93      1430          LDA #H,LINE2
94EF- 85 09      1440          STA *BUFF+1
94F1- 60         1450          RTS
                 1460          ;
94F2- AD 9B 95   1470 SCRLIN   LDA SLIN      ;CHECK FOR SHIFT OF BASE
94F5- C9 40      1480          CMP #64
94F7- D0 10      1490          BNE NX1
94F9- C9 80      1500          CMP #128
94FB- D0 0C      1510          BNE NX1
94FD- AD 98 95   1520          LDA SAVBAS
9500- 18         1530          CLC
9501- 69 28      1540          ADC #$28
9503- 8D 98 95   1550          STA SAVBAS
9506- 8D 96 95   1560          STA BASE
9509- 60         1570 NX1      RTS
                 1580          ;
950A- A2 00      1590 DUMP     LDX #0        ;DUMP BUFFERS TO PRINTER
950C- A9 00      1600 DUMP1    LDA #0
950E- 85 E3      1610          STA *CHAR
9510- A0 02      1620          LDY #2
9512- 20 37 95   1630          JSR DUMPY     ;Y+1 CHARS TO PRINTER
9515- 20 68 95   1640          JSR TRANS     ;TRANSITION TO NEXT BYTE
9518- E8         1650          INX
9519- A0 02      1660          LDY #2
951B- 20 37 95   1670          JSR DUMPY     ;REST OF BYTE
951E- E8         1680          INX
951F- E0 24      1690          CPX #36       ;DONE?
9521- 30 E9      1700          BMI DUMP1
9523- A0 02      1710          LDY #2        ;FINISH COLS 127-132
9525- 20 37 95   1720          JSR DUMPY
9528- 20 68 95   1730          JSR TRANS
952B- E8         1740          INX
952C- A0 01      1750          LDY #1
952E- 20 37 95   1760          JSR DUMPY
9531- A9 0D      1770          LDA #$0D      ;CARRIAGE RETURN
9533- 20 02 C1   1780          JSR DRIVER
9536- 60         1790          RTS
                 1800          ;
9537- 7E 81 93   1810 DUMPY    ROR LINE0,X   ;Y+1 BYTES TO PRINTER
953A- 66 E3      1820          ROR *CHAR     ;BIT 1
953C- 7E 81 93   1830          ROR LINE0,X
953F- 66 E3      1840          ROR *CHAR     ;BIT 2
9541- 7E A7 93   1850          ROR LINE1,X
9544- 66 E3      1860          ROR *CHAR     ;BIT 3
9546- 7E A7 93   1870          ROR LINE1,X
9549- 66 E3      1880          ROR *CHAR     ;BIT 4
954B- 7E CD 93   1890          ROR LINE2,X
954E- 66 E3      1900          ROR *CHAR     ;BIT 5
9550- 7E CD 93   1910          ROR LINE2,X
9553- 66 E3      1920          ROR *CHAR     ;BIT 6
9555- 18         1930          CLC
9556- 66 E3      1940          ROR *CHAR     ;BIT 7 = 0
9558- 38         1950          SEC
9559- 66 E3      1960          ROR *CHAR     ;BIT 8 = 1
955B- A5 E3      1970          LDA *CHAR
955D- 20 02 C1   1980          JSR DRIVER    ;PRINT
9560- A9 00      1990          LDA #0
9562- 85 E3      2000          STA *CHAR
9564- 88         2010          DEY
9565- 10 D0      2020          BPL DUMPY
9567- 60         2030          RTS
9568- 7E 81 93   2040 TRANS    ROR LINE0,X   ;FINISH BYTE
                 2050                        ;AND START NEXT
```

Listing 3 *(Continued)*

```
956B- 66 E3      2060          ROR *CHAR
956D- 7E 82 93   2070          ROR LINE0+1,X
9570- 66 E3      2080          ROR *CHAR
9572- 7E A7 93   2090          ROR LINE1,X
9575- 66 E3      2100          ROR *CHAR
9577- 7E A8 93   2110          ROR LINE1+1,X
957A- 66 E3      2120          ROR *CHAR
957C- 7E CD 93   2130          ROR LINE2,X
957F- 66 E3      2140          ROR *CHAR
9581- 7E CE 93   2150          ROR LINE2+1,X
9584- 66 E3      2160          ROR *CHAR
9586- 18         2170          CLC
9587- 66 E3      2180          ROR *CHAR     ;BIT 7 = 0
9589- 38         2190          SEC
958A- 66 E3      2200          ROR *CHAR     ;BIT 8 = 1
958C- A5 E3      2210          LDA *CHAR
958E- 20 02 C1   2220          JSR DRIVER    ;SEND TO PRINTER
9591- A9 00      2230          LDA #0
9593- 85 E3      2240          STA *CHAR
9595- 60         2250          RTS
9596-            2260 BASE     .DS 2
9598-            2270 SAVBAS   .DS 2
959A-            2280 LINE     .DS 1
959B-            2290 SLIN     .DS 1
959C-            2300 CURLIM   .DS 2
                 2310          .BA $9381

9381-            2320 LINE0    .DS 38
93A7-            2330 LINE1    .DS 38
93CD-            2340 LINE2    .DS 38
                 2350          .EN
```

MICRO

MICRO™

Microbes and Updates

Jim Sherman of Huntsville, AL, called in with these corrections to "Saucer Launch" by Mike Dougherty (42:53):

On page 59 the listing is out of order. Lines 108E through 1092 belong at the bottom of the page (after line 108C). On page 60, line 10D2 should read: F0 06 BEQ NOXP.

Here are a few corrections to the RUNZMENU article by Frank Shyjka (45:67):

On page 68 in the far right column, the third line of BCDF should read:

85 39 20 51 A8 A9 **8C** 8D 6D

BA8D should read:

D2 D5 CE DA CD C5 CE D5 **BF**

Erken Heinzjosef from West Germany wrote in with this update:

In MICRO 43 you published a Call Routine for the Superboard. Although it is very good, I have found a simpler way. My routine is only nine bytes long but it has a disadvantage: you cannot use hex addresses. But you can use labels! See listing 1 for the machine-language routine and listing 2 for the equivalent BASIC load. The syntax must be

Z (or any alpha) =
USR (any argument) 65030

or

Z (or any alpha =
USR (any argument) SC

The label "SCREEN CLEAR" gives syntax error as BASIC thinks it should be SQR.

65030 = hex FE06 = Screen clear in the C 1 S Monitor ROM from Aardvark. If you use labels, don't forget to define the label:

```
10 SC = 65030
20 WARMSTART = 0
30 X = USR(X) SC
40 X = USR(X) WARMSTART
```

You have to set the USR Vector at first by POKE 11,64: POKE 12,2. My BASIC load does it, but after a BREAK you have to reset the vector.

Listing 1

```
0000                    ;CALL-ROUTINE FOR SUPERBOARD
0000                    ;
0000                    ;H. J. Erken, West Germany
0000                    ;
0000                    ;
0000                    ;Addresses can be decimal values or labels
0000                    ;
0000                    ;To use, first set up the USR Vector by
0000                    ;POKE 11,64:POKE12,2
0000                    ;
0000                    ;
0240                    * = $0240
0240                    ;
0240 20ADAA             JSR $AAAD      ;Evalute any expression
0243 2008B4             JSR $B408      ;Convert floating to fix
0246 6C1100             JMP ($11)      ;Hex value of expression is
0249                                   ;stored in $11/$12
```

Listing 2

```
10 REM CALL FOR SUPERBOARD
20 REM H.J. ERKEN, WEST GERMANY
30 REM
40 FOR X = 576 TO 584
50 READ A: POKE X,A: NEXT
60 POKE 11,64: POKE 12,2: REM INIT OF USR FUNCTION
70 NEW
80 DATA 32, 173, 170, 32, 8, 180, 108, 17, 0
```

MICRO

APPLE • ALTOS • ATARI • MAXELL • DYSAN • EPSON • CCS • SHARP • CASIO • HP • VERBATIM • MEMOREX • SOROC • CORVUS • ADDS

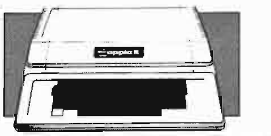
MICRO
BUSINESS WORLD INC.

Information Line (213) 996-2252
TOLL FREE MAIL ORDER LINES
(800) 423-5886 Outside Calif.

SPECIAL OF THE MONTH

APPLE II PLUS 48K W/16K EXPANSION BOARD = 64K

Our price $1199.00



List price $1780.00
Our price $1199.00
Save $581.00

COMPUTERS

apple computer

	List	Our Price	SAVE
Apple II-16K	$1330.00	$999.00	$331.00
Apple II-32K	1430.00	SPECIAL	$
Apple II-48K	1530.00	CALL	$
Apple II-64K	1725.00		$
Apple III 128-K	3915.00	2997.00	918.00
ALTOS 8CS 8000-15	5990.00	4450.00	1540.00
SHARP-64K Z80 FULL KBD YX-3200			CALL
NEC PC 8001	1295.00	1050.00	245.00

ATARI PERSONAL COMPUTERS	List	Our Price	SAVE
400 16K Bytes of Ram	595.00	337.45	257.05
800 16K Bytes of Ram	$1080.00	739.00	332.00
410 Program Recorder	90.00	77.00	13.00
810 Disk Drive	600.00	457.00	143.00
825 Printer (80 col- Centronic 737)	999.95	769.00	230.00
820 Printer (40 col impact)	450.00	353.00	97.00
830 Acoustic Modem	200.00	155.00	45.00
850 Interface Module	220.00	192.00	48.00
Atari Visicalc	200.00	164.00	36.00
Atari 400-16K	595.00	327.00	304.00

HEWLETT PACKARD	List	Our Price	SAVE
HP-125 Microcomputer	3750.00	2990.00	760.00
HP-85 Microcomputer	3250.00	2475.00	775.00
HP-83 Microcomputer	2250.00	1777.00	473.00
16K Exp-Memory Module	295.00	255.00	40.00
Graphics Plotter 7225	2450.00	2075.00	375.00
Personality Mod. for 7225	750.00	665.00	85.00
2631B Impact/printer/ hvy dty	3950.00	3250.00	700.00
Option 020 for 2631B	150.00	125.00	25.00
8 Drives to choose from 82902S	1300.00	1125.00	195.00
9895A 8" Dual Drive	6850.00	5500.00	1350.00
Graphics Tablet 9111A	2050.00	1678.00	374.00
HP-41 CV New 2.2 bytes mem	325.00	250.00	75.00
HP-41-C Calculator	250.00	185.00	65.00
Card Reader for 41CV/C	215.00	162.00	53.00
Printer for 41CV/C	385.00	289.00	101.00
Optical Wand for 41 CV/C	125.00	97.00	28.00
Quad Ram equals 4 Mem. Mods	95.00	81.00	14.00
Memory Modules for 41C		25.00	$
HP-97 Programable Printer	750.00	595.00	175.00
HP-67 Programble Calculator	375.00	295.00	80.00
HP-34C Programble Scientific	150.00	117.00	33.00
HP-38C Programble Bus. R/E	150.00	117.00	33.00
HP-32E Adv. Scientific	55.00	48.00	7.00
HP-37E Business Mgmt.	75.00	57.00	18.00

We carry a large inventory of Libraries, accessories and supplies.

PRINTERS

EPSON	List	Our Price	SAVE
MX 80 FT	745.00	549.00	196.00
MX 80 IMPACT	645.00	450.00	195.00
MX 70 IMPACT	500.00	390.00	110.00
MX 100	995.00	765.00	230.00
APPLE SILENTYPE	645.00	299.00	346.00
ANADEX 9501	1650.00	1299.00	351.00
NEC			
5510	3195.00	2445.00	750.00
5515	3295.00	2545.00	750.00
3510	2495.00	1795.00	700.00
3515	2545.00	1849.00	696.00
OKIDATA			
MICROLINE 80	545.00	395.00	150.00
MICROLINE 82	649.00	549.00	100.00
MICROLINE 83	1050.00	769.00	281.00
PAPER TIGER			
445G with Graphics	795.00	695.00	100.00
460G with Graphics	1,394.00	899.00	495.00
560G New full size	1,695.00	1,139.00	556.00
DIABLO (LETTER QUALITY)			
630 R102 bi-directional tractors	2,965.00	2,350.00	615.00
1640K109 keyboard tractors	4,000.00	2,899.00	1,100.05
630 RO Receive Only	2,710.00	2,250.00	460.00
1650K 136 keyboard tractors	4,000.00	3,100.00	900.00

APPLE II STUDENT SYSTEM

- Apple II Plus System-48K RAM
- Disk II Floppy Disk & Interface (DOS 3.3)
- 12" Grn. Phs. Video Monitor

SAVE $655

List $2450.00

Our Price $1795

MONITORS

	List	Our Price	SAVE
NEC Grn. Phs. 12"	$285.00	239.99	$46.00
BMC Green Phs. 12"	275.00	229.00	46.00
Zenith 12"	159.00	119.00	30.00

SANYO MONITORS
High Resolution, Number 1 seller

	List	Our Price	SAVE
13" Color (new) high quality	$550.00	388.00	$162.00
12" Green Phosphorous	360.00	240.00	120.00
12" Black & White	340.00	217.00	123.00
15" Black & White	370.00	235.00	135.00
9" Black & White The Best Seller!	235.00	145.00	90.00

DISKETTES

SOLD IN BOXES OF 10
(Min. Purchase)
BUY $100 of the Following Diskettes

	List	Our Price	SAVE
DYSAN		PRICE PER DISKETTE	
104/1 5" SOFT SECTOR	$ 6.00	3.99	$ 2.01
104/1D = " DBL DEN.-SOFT SEC	6.40	4.60	1.80
3740/1 8" SOFT SECTOR	7.25	4.75	2.50
3740/1D 8" DBL DEN.-SOFT SECTOR	10.75	6.90	3.85
MAXELL			
MD-1 5" SOFT SECTOR/DBL. DEN.	5.00	4.00	1.00
MD-2 = " SOFT SECTOR/DBL. SIDE/DBL. DEN.	7.00	4.90	2.10
FD-1 8" SOFT SEC./DBL. DEN.	6.50	4.50	2.00
FD-2 8" SOFT SEC./DBL SIDE/DBL. DEN.	8.50	5.95	2.55

TERMINALS

TELEVIDEO	List	Our Price	SAVE
910	699.00	599.00	100.00
912C	950.00	699.00	251.00
920C	995.00	795.00	200.00
950C	1,195.00	949.00	246.00

DRIVES

CORVUS			
5 MBYTES	3,750.00	3,050.00	700.00
10 MBYTES	5,350.00	4,449.00	901.00
20 MBYTES	6,450.00	6,450.00	1,125.00

1 Year Extended WARRANTY $99.00 INQUIRE

$ AVINGS $ ERVICE $ ELECTION $ ATISFACTION

MEMBER BBB

SOFTWARE
FOR APPLE II

	List	Our Price	SAVE
Apple Fortran	$ 200.00	$ 147.00	$ 53.00
DOS 3.3 (16 Sector)	60.00	45.00	15.00
Apple PILOT (16 Sector)	150.00	119.00	31.00
Apple FORTRAN (requires A2B0006) (16 Sector)	200.00	159.00	41.00
Language System with Apple Pascal	495.00	399.00	96.00
BPI General Ledger System	395.00	319.00	76.00
BPI Inventory Package	395.00	319.00	76.00
Visidex	200.00	159.00	41.00
Visicalc	200.00	159.00	41.00
Desktop Plan II	200.00	159.00	41.00
Microlab Database System	150.00	119.00	31.00
Stoneware DB Master	229.00	189.00	40.00
Muse Supertext II	150.00	119.00	31.00
Softape Magic Window	99.00	72.00	27.00

TEXAS INSTRUMENTS

TI 99/4A Console New Improved	950.00	385.95	564.05
10" Color Monitor High Resolution	374.95	317.95	57.00
32K Memory Module	399.95	312.95	87.00
Speech Synthesizer	149.95	127.95	22.00
Disk Memory Drive	499.95	390.95	109.00
RF Modulator	49.95	42.50	7.45
Telephone Coupler (Modem)	224.95	185.95	39.00
Printer (Solid State)	399.95	315.95	84.00

MODEMS

NOVATION CAT MODEM	189.95	140.00	49.95
NOVATION D-CAT	199.00	150.00	49.00
NOVATION APPLE-CAT	349.00	310.00	39.00
HAYES MICROMODEM	379.00	295.00	84.00
HAYES SMARTMODEM	279.00	239.00	40.00

CALCULATORS

CASIO			
HR-10	49.95	39.00	10.95
HR-12	54.95	42.00	12.95
FR-100	79.95	59.00	20.95
FR-1210	129.95	99.00	30.95
PQ-20	29.95	23.00	6.95
LC-785	12.95	10.00	2.95
LC-3165	12.95	10.00	2.95
FX-68 Scientific	29.95	23.00	6.95
FX-81 Scientific	19.95	17.00	2.95
FX-3600P Scientific	39.95	29.95	10.00
FX-602P "Computer Talk" 88 Memories Programming Upper & Lower Case Dot Matrix	129.95	99.00	30.95
FX-702P Solves Problems with Alpha-Numeric Clarity, uses Basic Language	199.95	159.00	40.95

TELE. ANSW. DEVICES

PHONE MATE			
900	119.95	86.00	33.95
905 Remote	199.95	144.00	55.95
910	159.95	115.00	44.95
920	199.95	144.00	55.95
925 Remote	239.95	173.00	66.95
930 Remote	299.95	216.00	83.95
950 Remote	339.95	245.00	94.95
960 Remote	399.95	288.00	111.95

MICRO BUSINESS WORLD MAIL ORDER WAREHOUSE
18720 Oxnard, #108 Tarzana, CA 91356
Telex 182852 Answer: MICRO TZNA

OUTSIDE CA CALL TOLL FREE 1 (800) 423-5886 IN CA (213) 996-2252

Name (Please Print) ____
Address ____
City ____ State ____ Zip ____

Qty	Make	Model	Description	Price	Total

TOTAL ORDER $ ____
TAX IF APPLICABLE* ____
SHIPPING & HANDLING** ____
TOTAL ENCLOSED $ ____

*California residents add 6% sales tax.
**Add 3% for Shipping & Handling — Add 3% surcharge for credit cards. Orders cannot be shipped unless accompanied by payment including shipping handling and tax where applicable.

Certified Check or M.O. Bank Wire Transfer
Cash/Credit Card(Allow 2 weeks clearance for checks)

CREDIT CARD - ____
Exp. Date ____ Signature ____

We reserve the right to correct typographical errors. This ad supercedes all previous ads.

GTE • PERSONAL SOFTWARE • PAPER TIGER • D.C. HAYS MICROCOMPUTERS • NOVATION • CANON • HAYDEN • HITACHI • VERBATIM • AMPEX • HAZELTINE • NEC • CENTRONIC • BMC

HP • D.C. HAYS MICROCOMPUTERS • PAPER TIGER • GTE • DIABLO • BMC • CENTRONIC • NEC • HAZELTINE • MEMOREX • AMPEX • PERSONAL SOFTWARE • ADDS • XEROX • DIABLO

MAXELL • DYSAN • EPSON • CCS • SHARP • CASIO • HP • VERBATIM • MEMOREX • SOROC • CORVUS • PERSONAL SOFTWARE • CCS

MICRO™

The Single Life

By Brad Rinehart

In my previous columns I have explained various HDE Disk BASIC features. This month I will cover some of the most unusual commands, as well as file handling techniques.

HDE Disk BASIC's more powerful commands include INSTR (in string), VARPTR (variable pointer), EXEC, MERGE, GET!, PUT!, PRINT USING, MKI$, MKS$, CVI, and CVS.

The INSTR command returns the position of a search string within a target string. The syntax for the command is

```
VAR1 = INSTR ( VAR2, "STRING1",
"STRING2" )
```

VAR2 is the character position within the target string where you begin searching. STRING1 is the target string and STRING2 is the key. VAR2, or the starting character position, is optional. If you omit it the search starts at the beginning of the target string. After you evaluate the function, VAR1 will contain the character position within the target string where the key was found. For example, if you have the string "THIS IS A TEST", and the key "IS", the statement

```
X = INSTR ("THIS IS A TEST", "IS")
```

will return the variable X with the value 3. You may have expected X to equal 6. However, the word 'THIS' contains an 'IS' in it. The statement

```
X = INSTR (4, "THIS IS A TEST", "IS")
```

will return X with the value 6. This time, we specified the fourth character as the starting point for the search.

The INSTR command also acts as the argument for an IF-THEN-ELSE statement.

```
IF INSTR("THIS IS A TEST","IS")
THEN GOSUB 1000 : ELSE PRINT "NO"
```

will cause a GOSUB to line 1000. The statement following the ELSE will not be executed. However, if you change this statement to

```
IF INSTR(7,"THIS IS A TEST","IS")
THEN GOSUB 1000 : ELSE PRINT "NO"
```

control will pass to the ELSE statement and the word 'NO' will be printed to the screen.

The VARPTR command returns the memory address (in decimal) of:

1. The exponent of the variable A,
2. The least significant byte of the two-byte integer A%,
3. The byte defining the length of the string A$.

To use this information, you must understand how BASIC stores variable data in memory. Numeric variables such as A and A% are stored in five-byte and two-byte locations, respectively. VARPTR will return the address of the beginning of this memory location sequence. In the case of string variables, a three-byte descriptor defines the string. The first byte is the length of the string in memory, and the second and third bytes are the address pointer to the string.

VARPTR may also be used to determine whether or not a variable exists. For example, if you study the statement

```
X = VARPTR (A$(1))
```

you will see that X will be returned with the value zero if the variable does not exist. Frequently, I need to know if an array has already been dimensioned. Without VARPTR, the only recourse is to redefine it and trap the error with an ON ERROR GOTO statement. I avoid ON ERROR GOTO statements; they make it too easy to build hidden 'BUGS' into a program.

The EXEC and MERGE commands each accept input from the disk as though it were entered from the keyboard. Either command accepts input from a SEQUENTIAL DATA file or a LIST# (ASCII) file.

The MERGE command enters program lines from the disk file as opposed to entering them from the keyboard. This feature is useful when standard subroutines are to be used in several programs. For example, you may have a particular subroutine that is used to address the cursor on your terminal. Rather than manually entering the program lines each time you want to build a new program, the subroutine may first be entered from the keyboard, then LIST#ed out to a file called CURSR. Then whenever you want to use the subroutine within a program you simply enter MERGE "CURSR". This command, entered from the keyboard, will open the CURSR file and insert the lines into the program. You can save quite a bit of development time here!

The EXEC command will EXECute the command lines as they are read from the file. But with an EXEC file, the commands must be legal direct commands, such as PRINT, A=1, OPEN, CLOSE, PUT, and GET. Examples of commands that are not legal direct commands are INPUT and PRINT USING. Therefore, they may be used in files that are to be MERGEd, but not in files that are to be EXECed.

The EXEC command is useful for repetitive tasks. For example, when you have several programs you want to list to the printer, you can create an EXEC file that will initialize the output device, load the first file, list it, load the next file, list it, and so on. This can all be done without any human intervention. Remember, any sequence of commands you enter repetitively from the keyboard may be put into an EXEC file and reused.

The EXEC command also accepts input from a string variable. This feature lets you build a command in the variable A$, and then execute and EXEC A$ command. Any string variable may be used. However, your commands may be no longer than 250 bytes. Of course, if several commands are to be EXECed, they could be constructed in a string array and executed in a FOR-NEXT loop as in:

```
FOR X = 1 TO 5
EXEC A$(X)
NEXT
```

(Continued on next page)

The Single Life *(continued)*

You might use this feature when you execute routines that are to be invisible to the user.

Some Printing Conventions

HDE has implemented a command, 'CALL', for directing output to peripherals such as printers and modems. The syntax for the command is CALL "DEVICE NAME", where the DEVICE NAME is a three character name associated with a binary or machine-language program stored on the system disk.

To use the CALL command you must either write or purchase the device driver program. This device driver is then SAVed to the system disk (drive 0 to 1). The CALL command will load and initialize the driver. With the driver initialized, output may be directed to the screen, the device, or both. To output to the device, commands such as PRINT, LIST, FIND, and LIB are followed by an exclamation point (!), as in PRINT!, LIST!, FIND!, and LIB!. To output to the screen, even while the device is enabled, eliminate the exclamation point. To disable the device, use the command CALL0 (call zero). Once the device is disabled, output from statements such as PRINT! will be directed to the screen. To change the output device from a printer to a modem, just execute another CALL with the proper device name as the argument, as in CALL "MOD".

You may want to write a driver that accepts input from a modem or another terminal. Then when you want to pass control to that device, just initialize it with the CALL command.

PRINT USING may be used to manipulate string data. If you consider that A$ = "FRED", B$ = "SMITH", then the statement

```
PRINT USING "PAY TO !! %          %";
A$".";B$
```

will print PAY TO F. SMITH to the terminal. To dispatch this to an output device, use the statement:

```
PRINT! USING "PAY TO !! %          %";
A$".";B$
```

The exclamation point after the PRINT command directs the output to the external device that was initialized with the CALL command.

PRINT USING allows seven different types of format identifiers for dealing with numbers. The pound sign is used exclusively for defining the field width of a number. The PRINT USING command in conjunction with the pound sign causes number fields to be right-justified. For example, if you wish to print a column of numbers beginning at position 50 on the page, you could use the command:

```
PRINT TAB(50); USING "######, #"; N
```

The use of the comma in the field specifier will cause a comma to be output every three places in the number. Your printout might look like:

```
123,456
    232
  1,508
```

If decimal positions are to be defined, simply use the command

```
PRINT TAB(50); USING "#####, #. ##"; N
```

and the column will be right-justified, rounded to two decimal places, and zero-filled on the right.

```
123,456.25
    232.00
  1,508.07
```

File-Handling Techniques

Along with these unique commands, I want to introduce some of HDE Disk BASIC's file-handling techniques. There are three types of data files: SNAPSHOT, SEQUENTIAL, and RANDOM access. In addition, you have the ability to create an ASCII file of the program listing using the LIST# (list pound sign) command.

The main difference between the different types of data files is the way the data is stored on the disk and the techniques used to access it. First of all, the snapshot data file is, as its name implies, a snapshot of all the data in memory. If you can picture being able to grab the data in memory, compress it into one block, and then write it to the disk, you can understand the operation of this file. It is most useful when saving analytical data. For example, if you are accumulating data and monitoring the results of laboratory tests, but need something recorded quickly, the command SAVED "TEST1" (meaning "save data") will, in a matter of seconds, write the contents of every variable to the disk file TEST1. To reload the information for later analysis, simply execute the command LOAD "TEST1" and memory will be restored to its previous contents.

With the RANDOM access file you can randomly access records within the file *without* reading or writing any other part of the file. This provides quick access to any record in the file.

The SEQUENTIAL data files are useful for data such as tax tables, rate tables, etc. Sequential files are best used when data fields or records are of varying lengths. Normally this type of data is manipulated in memory then written to the disk file when the user has completed working with it. The disadvantage of sequential file use is that to read the last record in the file, you must read the entire file. The same is true when changing one record in the file: you must read the file, make the change, and rewrite the entire file. But sequential files are usually more compact than RANDOM files.

To use sequential files properly, you must understand the structure of the file. First, records within the file may be terminated by a carriage return character ($0D), a comma, or, when dealing with numerical fields, a space. The end of the file is signified by an end-of-file, or EOF mark. If you could look into the disk, you might find any of the following structures in a sequential file:

```
THIS IS RECORD 1$0D THIS IS
RECORD 2$0DEOF 22 33 44 $0D 11 66
55 $0D 13 $0D 99 21 $0DEOF 22, 33,
-44$0D 11, 66, 55$0D 13$0D
99,-21$0DEOF "THIS IS RECORD
1"$0D"THIS IS RECORD 2","THIS IS
RECORD 3"$0DEOF
```

The first two files were created using the PRINT# command, the second two using the WRITE# commands.

Please send all correspondence for Mr. Rinehart to 1508 Stanton St., York, PA 17404.

MICRO

Commodore and MICRO

by Loren Wright

Commodore has been a dominant force in the microcomputer world since the Personal Electronic Transactor (or PET) was introduced in 1977. The first PETs admittedly had their problems. The small keyboard, a holdover from Commodore's calculator background, gave the PET a toy-like appearance. There was no resident machine language monitor — a tape version had to be loaded into RAM each time. There were also a few problems, of varying annoyance, in the operating system. Documentation had to be obtained from users' groups, since Commodore would not, and perhaps could not, provide very much.

In the U.S., the PET came out in the face of stiff competition from Apple and Radio Shack. The Apple attracted many people with its high-resolution color and full-size keyboard, while Radio Shack, with its nationwide network of stores and well-organized marketing effort, drew even more attention. To compound their problems, Commodore attempted to sell the PET only directly or through its Mr. Calculator stores, and there were delivery delays of many months.

The PET, as many other computer shoppers recognized, offered a system complete with CRT display, cassette mass-storage, and fully implemented BASIC for a price less than comparable systems from either Apple or Radio Shack. In the rest of the world, where Commodore was organizationally better equipped to compete, the PET became (and still is) the number one microcomputer.

The company corrected most of the problems with a new operating system. Unfortunately, this was done without much consideration for those who had already invested a lot of time and money developing commercial software for the old operating system. Many people abandoned ship at this point, but most adjusted and are still loyal PET owners. Since then, there has been yet another operating system introduced, but this time the changes were far less radical, and Commodore cooperated considerably more in the transition.

To understate the situation, Commodore has been unpredictable in its approach to the market. When the 80-column business machine and the VIC were announced, there were widespread fears that the company would abandon PET owners in favor of the more lucrative entertainment and business markets. So far, those fears have not been justified. It is clear now, particularly with the announcement of several new computers, that Commodore wants to compete in all microcomputer markets. The new line-up will apparently include the Ultimax, the VIC, the SuperVIC, the PET, the 8016/8032, the color 8032, the Commodore-64, the SuperPET, and the 8096. Each of these is aimed at a particular segment of the market.

If its new advertising campaign is any indication, Commodore intends to provide the best value for any microcomputing need. The company plans to accomplish this not by inventing radically new computers, but rather by producing variations on its PET and VIC themes to compete over the full range of the market. To quote Jack Tramiel, the man behind Commodore, "We will become the Japanese!" — meaning that they will offer a lower-priced alternative. Whether Commodore can actually accomplish its goals is still uncertain.

MICRO has been covering the PET since its inception. Much of our job in the early days was to provide the information not provided by Commodore, and to help PET owners get around the bugs in their systems. Things have progressed much further than that now. The PET system is virtually bug-free and good documentation is available not only from Commodore, but from a number of other sources. We will continue to publish articles of special interest to PET users, but you will find many of our other articles valuable as well. More articles written for other computers will be accessible to PET users, and we will continue to expand your horizons with material on new programming techniques, languages, and applications.

This issue's feature article "Growing Knowledge Trees," by David Heise, introduces artificial intelligence to MICRO readers. While it is written especially for the PET, I recommend that all MICRO readers try to see this program in operation. It should provide some ideas for your own artificial intelligence programs.

"Menu and Tape Timer," by Dale DePriest is a sequel to last month's "A Real Tape Operating System." In that article he discussed the good and bad features of the PET's tape system and presented some techniques to get the most from that system. This month's programs will help you turn your cassette collection into a well-organized file retrieval system. Although a disk drive is faster and more convenient, the PET's cassette system, with a few refinements, can offer a considerably less expensive alternative, which is still very satisfactory.

Louis Sander's "PET Memory Protector" is a simple circuit that is inserted between one of your PET's static RAM chips and its socket. Depending on where it is installed, PMP protects 1K or more of RAM from BASIC, LOADs and resets. The reset button, which is part of the circuit, can be used for either a cold or a warm reset.

MICRO

PET Memory Protector

by Louis F. Sander

This easy-to-build add-on for 8K PETs selectively isolates 1K or more of memory from BASIC and from resets. The protected area is an ideal repository for monitors or other machine-language programs.

PMP
requires:

PET with socketed 6550 or 2114 RAM chips (all small-keyboard PETs, except the most recent release) and a number of electronic parts.

PET users have a shortage of protected memory for machine-language programs. The PET Memory Protector is a simple add-on device that eliminates this shortage. In a typical PET, only the second cassette buffer, with its meager 192 bytes, is out of reach of BASIC, resets, and LOADs. The only way to protect an area in high memory is to do several zero-page POKEs, an annoying task. The PET Memory Protector, or PMP, provides a simpler and more reliable way to reserve 1K blocks of high memory for machine-language programs or any other use.

PMP is activated by momentarily depressing one switch, eliminating the need for memory-reserving POKEs. BASIC cannot write into the PMP-protected area unless specifically directed, and LOADing a tape from either cassette does not affect it. The PMP includes a reset function that allows selection or deselection of memory protection while the reset is performed, all with one simple control.

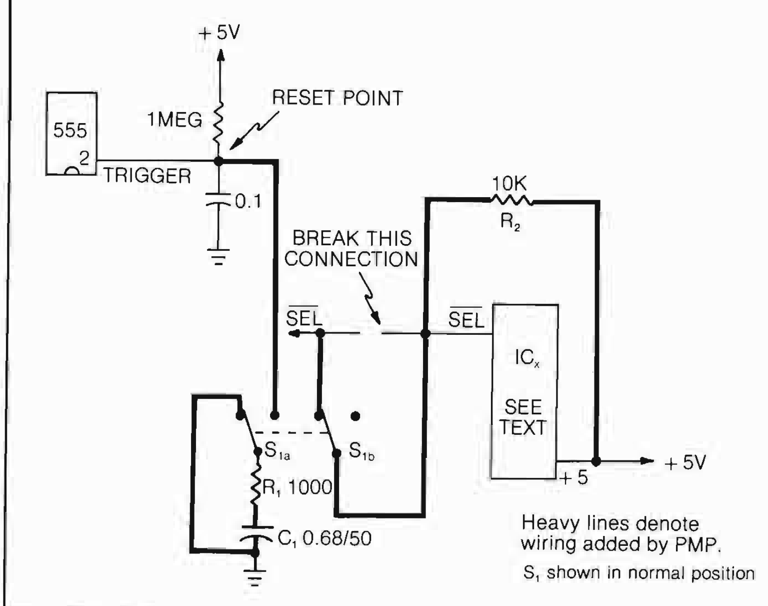
Installation of the PMP requires no drilling or cutting of the PET, and no soldering to any PET component. You simply insert the PMP connector between one memory chip and its socket, and mount the PMP switch in existing holes in the PET. The typical installation protects 1K of memory, but larger 1K multiples can be protected simply by moving the PMP connector. The present version of the PMP will work only with PETs using the type 6550 or 2114 RAMs in plug-in sockets (basically, all the small-keyboard machines). Work is in progress on a version for the large-keyboard machines. Construction of a PMP is not difficult for an experienced electronic builder; non-builders can purchase a fully assembled and tested version from the authors.

Theory of Operation

Figure 1 is the schematic diagram of the PMP. When S_1 is closed, it connects R_1 and C_1 to the trigger pin of PET's power-up timer, and opens the chip-select line to IC_x. IC_x can be any one of PET's RAM chips; typically it will be one of the two that constitute the top 1K of user RAM. The charging effect of C_1 momentarily lowers the voltage on the timer trigger pin, which activates the timer and a power-on reset. At the end of the timer's one-second cycle, PET writes a character into the lowest memory location of the user program area, then reads the contents of that location. If the read and write are identical, PET repeats the process at the next higher memory location. The first time the read and write do not match, PET concludes that it has passed the top of available RAM. It then sets its zero-page BASIC pointers accordingly, and puts the appropriate BYTES FREE message onto the CRT.

If S_1 is still actuated when the reset routine tries to write into IC_x, the $\overline{SEL}$ line will still be broken by S_{1b} and the

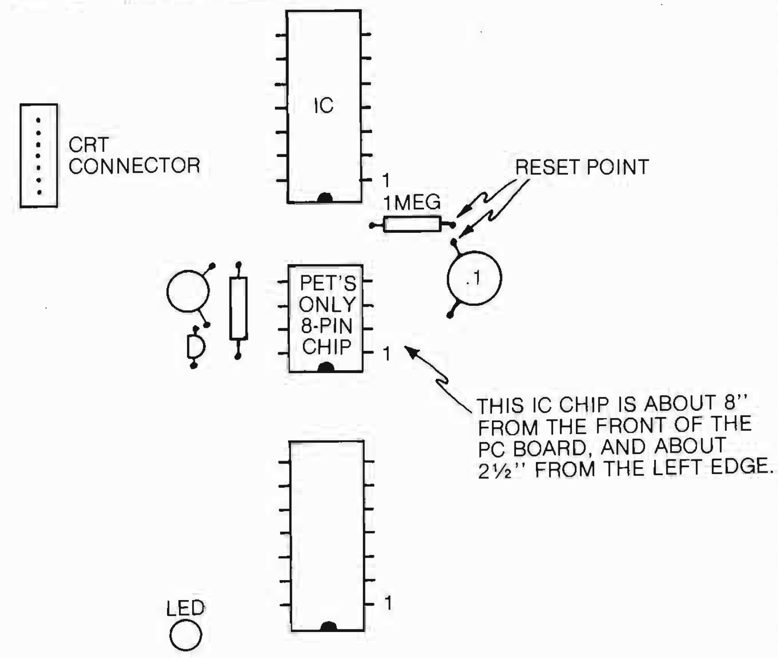
Figure 1: **PMP Schematic**



$\overline{\text{SEL}}$ pin will be held high by the voltage from R_2. So the read/write process into IC_x will fail, and PET will conclude that IC_x and all memory above it does not exist. Because IC_x is paired with another chip that is not disabled by S_1, the reset will have modified the lowest single byte in IC_x, but will not have affected any higher memory locations. The BYTES FREE message will include only those memory locations below IC_x. S_1 can be released as soon as the BYTES FREE message appears, and at that time IC_x will be fully functional, but BASIC will not know that it is there. In other words, IC_x and above will constitute a fully protected area of memory.

If S_1 is released before the timer finishes its cycle, IC_x will be properly connected when PET attempts to access it. The reset process will proceed normally, writing over any information in IC_x and above, and including those locations in the BYTES FREE message. IC_x and above will *not* be protected memory. In either case, as soon as S_1 is released, C_1 will discharge through R_1, to be ready for the next reset.

Figure 2: Locating the Reset Point



Construction

If you are not an experienced electronics builder, you shouldn't try this project on your own, since a miswired PMP could mean disaster to your PET. For the builder with any experience, PMP construction is straightforward, except for breaking the $\overline{\text{SEL}}$ line to IC_x. For the 6550, $\overline{\text{SEL}}$ is pin 18, and +5 is pin 17. For the 2114, $\overline{\text{SEL}}$ is pin 8 and +5 is pin 18. Make up a "PMP connector" from two wire-wrap IC sockets. Plug the sockets together, piggyback-style, and cut the pin carrying the $\overline{\text{SEL}}$ lead from one socket to the other. If you want to do a more professional job, use one wire-wrap socket and the plastic base from another, cutting one of the pins. Either way, solder your S_{1b} leads to the severed ends, and glue everything together so it can't move. Solder R_2 to the appropriate pins of the upper socket, and you're in business. (You need wire-wrap-type sockets for this work, because the solder-tab-type pins are too short to work with, unless you're used to microsurgery.)

We advise the prospective builder to be persistent in his search for parts, since S_1 and the IC sockets are not common items. They are manufactured by the thousands for industrial use, but your local Radio Shack doesn't carry them.

Installation

The first step in installation is to unplug your PET. Then find a way to mount S_1 permanently. You can either drill a hole in your PET, or drill a ½" × 3" strip of heavy sheet metal to accept S_1. Then mount it to the cover hold-down bracket on PET's right side, using two additional holes drilled through your piece of sheet metal. If S_1 is properly chosen, it will easily fit in the ½" space between PET's cover and base, making a very attractive and unobtrusive installation.

Next, connect the wire from S_{1a} to the reset point. Here you can solder a wire directly to PET's circuit board, or you can use a tiny test clip to connect it to a component lead. The reset point is easy to find by locating the 555 timer chip, which is the only eight-pin IC in the PET. It's on the far left side of the printed circuit board, about eight inches from the front edge. The reset point is accessible either at pin 2 of the 555, or at the resistor or capacitor lead wire shown in figure 2. (By the way, this is the same point used by the reset buttons on old ROM PETs.)

Finally, locate IC_x and put the PMP connector between it and its socket. At the very front of the main printed circuit board are two identical rows of eight IC chips in sockets; this is PET's RAM. Each 1K of memory is made up of one IC in the front row, plus its partner in the second row. Half a byte is stored in each chip, for 1024 memory locations in each pair.

If your memory chips have 18 pins each, they are 2114's, and the IC's making up the lowest memory locations are to the far left. The highest memory locations are to the right. To protect 1K of memory, the PMP plugs into either one of the rightmost chips. With 2114's the PMP can be plugged into any RAM socket, protecting any number of 1K memory blocks.

If your memory chips have 22 pins, they are 6550's, and things work differently. The low memory locations are to the *right* in this case, and the high ones are to the left. Your PMP will only work properly in the highest 1K (the leftmost socket), or the highest 4K (number 4 from the left).

To locate IC_x, first determine how much memory you want PMP to protect. If it's 1K, then IC_x is the rightmost or leftmost IC in the front row. If you want to protect 2K of memory, IC_x is the chip just next to that one, and so on, at the rate of 1K per chip. For test purposes, you will need to protect 1K, so initially use the end chip in the front row. Use the left chip for 6550's, or the right chip for 2114's. (In every case, the corresponding chip in the second row could be used, with identical effect.

We've arbitrarily chosen the front row chips because they're easier to get to.)

Before removing IC_x, note the U-shaped depression on its top at one end. That is an orientation mark, and when it faces you, with the IC pins pointed downward, pin 1 is the closest pin to you on the right. See figure 2 for examples. Take careful note of IC_x's orientation, so that you'll be able to insert the PMP connector and IC_x in the proper direction.

When you've done this, gently pry IC_x from its socket, using a small screwdriver inserted from the front. Use standard static protection techniques: keep yourself grounded, and lay the naked IC, pins downward, on a piece of foil or conductive foam. Now insert the PMP connector into the vacant socket, being extremely careful to preserve proper orientation. Using static protection techniques, and once again paying careful attention to orientation, insert IC_x into the PMP connector, and you're ready to test your PMP.

Test and Operation

Visually inspect the installation to make sure there are no broken wires or short circuits. Make sure that the PMP is plugged into the correct socket, and that all its pins are making contact. (Look closely, as bent pins are common, and easy to miss.) Make the same check on the IC chip, where it plugs into the PMP. Finally, double check the orientation of the IC and the PMP; if either one is in backwards, correct it immediately.

When you're certain that everything is as it should be, turn on the power to your PET. You should get the normal BYTES FREE message (7167 bytes on the 8K PET). Now load a machine-language program (MLP) of some sort into the top part of the top 1K of memory. Ideally, it should extend to the very last free byte. Be sure your program doesn't use the very first byte of the top 1K, since that byte will be modified by the reset routine. Run your MLP to make sure that it works.

Now activate S_1, and keep it activated until the BYTES FREE message appears once again. If all has gone well, that message will have appeared about one second after you first activated S_1, and will indicate 1024 fewer bytes than normal. Next, LOAD and RUN a BASIC program that uses several string variables. Run your MLP once again. If both programs work properly, PMP has protected upper memory from being written into by BASIC.

Parts List

C_1 — 0.68 F, 50 wv

R_1 — 1000 ohm, ¼ watt

R_2 — 10K ohm, ¼ watt

S_1 — DPDT momentary toggle or pushbutton switch

Two 18-pin or 22-pin wire-wrap-type IC sockets (pin configuration depends on your RAM type)

Hookup wire

Glue (Devcon clear epoxy or similar)

Optional ½" × 3" piece of heavy sheet metal (for switch mounting bracket)

Total parts cost should be $10-$12 for top-quality, name-brand parts.

For the final test, momentarily activate S_1, this time being sure to release it before the BYTES FREE message appears. If you get a normal BYTES FREE message, and if both programs are gone from memory, your PMP is working correctly. Congratulations on a job well done!

Now here's the full story on clearing and protecting memory in your PMP-equipped machine.

1. POWER ON clears all memory, overwriting it with characters dictated by your ROM set.

2. Using either cassette drive to LOAD, SAVE, or VERIFY clears the associated cassette buffer, replacing what was there with data from the tape header. The unused cassette buffer is not affected.

3. Momentarily depressing S_1 and releasing it before the BYTES FREE message appears clears all memory *except* the cassette buffers, and gives a normal BYTES FREE message. The cleared memory is overwritten with characters dictated by your ROM set.

4. Holding S_1 until the reduced BYTES FREE message appears clears all memory *except* the two cassette buffers *and* the memory above the first protected byte. That first byte will be altered by the reset process, but is protected afterward. Anything previously existing above that byte will be unaffected by the reset, and will be protected from being written into by BASIC. It can be PEEKed and POKEd, but that is all.

That's the full story on the PET Memory Protector. We've found it to be a very handy device for protecting high memory, and we hope that you will, too. If you'd like to have a fully assembled and tested PMP, we've made some up that we'll send you for $20 each. Just send your name, address, and RAM type to: Louis F. Sander at 153 Mayer Drive, Pittsburgh, PA 15237.

Louis F. Sander designs and markets electronic systems for hospitals and other health care providers. He is the originator of COMPTUER KINDERGARTEN™, a computer familiarization course for adults. Victor H. Pitre installs and services medical electronic systems. Both have worked in electronics since pre-transistor days, and they have collaborated on several hardware and software innovations for small computers.

MICRO

Growing Knowledge Trees

by David R. Heise

Knowledge can be represented in tree diagrams that are stored and analyzed by computers. This PET program finds out what people know about a topic, analyzes answers, and shows users the organized results.

"Knowledge Trees" requires:

16K or 32K PET
3.0 Operating System
Printer and disk drive are supported, but optional. Notes are provided for 1.0 and 4.0 conversion.

A computer needs to be knowledgeable if it is to help you classify plants, diagnose illnesses, or identify beliefs that hold down productivity. But how do you make a computer knowledgeable? How do you teach a computer what an expert knows? How do you represent knowledge?

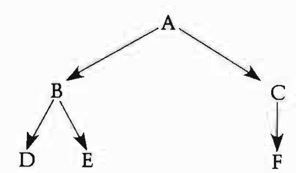
Research on these questions has been performed in computer science specialties like data base management and artificial intelligence, and also in social sciences like anthropology, sociology, and cognitive psychology. Data base management systems and psychological research demonstrate that some kinds of knowledge can be stored as associative networks with pointers linking items to related items. As a sociologist, I've shown that measurements about actors and behaviors can be used mathematically to generate information about social events. Artificial intelligence research and contemporary anthropology often represent knowledge as hierarchical trees of relationships.

This article discusses the tree concept for knowledge representation and presents a program that turns a Commodore microcomputer into a machine for gathering hierarchical knowledge from people and reducing it to its simplest form. The program illustrates how to pass multiple parameters to an assembly-language subroutine *via* the USR function in BASIC. It shows you how to create a BASIC subroutine that can call itself recursively, and demonstrates a method for automatically loading and protecting assembled code, along with a BASIC program. Assembled routines are provided for carrying out operations on trees.

Data Structures

Hierarchical knowledge can be represented in a tree diagram, which is a type of directed graph. See figure 1.

Figure 1



Letters A through F represent *nodes* in the graph. These are the entities involved in hierarchical relationships. For example, the nodes could be A = living creature, B = mammal, C = bird, D = dog, E = cat, and F = robin.

The arrows are called the *edges* of the graph. All of the arrows on a graph represent a certain relationship. For example, in figure 1, each arrow could represent the relation "is a kind of." Thus, reading backward along an arrow, we see that B is a kind of A. Different arrows show which entities are related to other entities. In general, the node at the top of an arrow is a *superordinate* of the node at the bottom of the arrow. The node at the bottom of an arrow is a *subordinate* of the node at the top.

Relations in a tree diagram are *transitive*. This means that a relationship between two entitites can be inferred when they are connected by a chain of arrows rather than by a direct arrow. For example, in figure 1, D (a dog), is a kind of B (mammal), and B (a mammal) is a kind of A (living creature). Therefore we can infer "a dog is a kind of living creature" even though there is no direct arrow from A to D. Many kinds of relationships are transitive. For example, time ordering is transitive: if event B occurs after A, and C occurs after B, then C occurs after A. "Is part of" also is transitive: if B is part of A, and C is part of B, then C is part of A. Some relations seem transitive but are not, like the relation "has." Jane may have a husband, and her husband may have a bald head. We would not want to conclude that Jane has a bald head.

Tree diagrams can be represented in matrix form. Figure 2 shows the matrix for the diagram in figure 1. Letters along the top of the matrix show nodes as originating sources for arrows. Letters along the side of the matrix show nodes as destination points for arrows. A zero in a cell indicates that no arrow connects the column node to the row node. A one indicates that an arrow goes from the originating node (column label) to the destination node (row label).

Figure 2

	A	B	C	D	E	F
A	<u>0</u>	0	0	0	0	0
B	1	<u>0</u>	0	0	0	0
C	1	0	<u>0</u>	0	0	0
D	0	1	0	<u>0</u>	0	0
E	0	1	0	0	<u>0</u>	0
F	0	0	1	0	0	<u>0</u>

The major diagonal is emphasized with underlining in figure 2. In a matrix that represents a hierarchical tree, the major diagonal is always filled with zeros, and nodes can be ordered such that all entries above the major diagonal are zero. A *topographical ordering* of nodes has the node at the top of the graph first (A). Nodes that are directly connected to this node come next (e.g., C and B), then nodes directly connected to these follow (e.g., F, E, D), and so on, until all nodes are listed. A topographical ordering orders the nodes as they are encountered when going from the top of the tree diagram to the bottom. If nodes are listed in topographical order for the matrix representation, then all cells above the major diagonal contain zeros.

A tree diagram can be stored inside a computer in various ways. We could store the matrix representation, but this wastes space on zeros above the main diagonal. Instead we will use the *edge list* representation. This approach stores a tree, including verbal labels for the nodes, in two lists — name and edge — as shown in figure 3.

Figure 3

Name List

(1) a living creature
(2) a mammal
(3) a bird
(4) a dog
(5) a cat
(6) a robin

Edge List

1,2
1,3
2,4
2,5
3,6

The name list is simply a list of node labels with index numbers implied. Each entry in the edge list corresponds to an arrow on the tree diagram, and the entry consists of two numbers. The first is the index number for the arrow's originating node. The second is the index number for the destination node. The edge list has as many entries as there are arrows on the diagram. Their order is not important.

In the program presented here, index numbers are interpreted as ASCII values and converted to characters. Thereby the edge list can be maintained as a character string, taking advantage of dynamic string allocation in CBM BASIC; we do not have to set aside space for the edge list, whose size is unpredictable beforehand. By adding 64 to each index number before converting to a character, we get ASCII values for letters of the alphabet. For example, the edge list in figure 3 could be represented as the following string:

'ABACBDBECF'

This string contains all the information represented in figures 1 and 2. Marking the string off into pairs of letters precisely defines each arrow in the tree diagram.

We now have a neat, concise way of representing hierarchical knowledge in a CBM microcomputer. Artificial intelligence research usually employs a list representation that calls for a LISP language interpreter, but that would not be as convenient as this approach that works in BASIC.

Elicitation

The next step is to input knowledge from humans into computers. Storing knowledge within programs is not an efficient approach because too many people do not know how to program. Rather, the computer should elicit and store people's knowledge:

— accommodating to a user's interest in a certain kind of relation,

— talking with the user about a given topic,

— helping the user recall topical elements (nodes),

— helping the user to define relations among elements (edges).

Ideally the computer would deal with any kind of relation for making trees and would talk in accordance with rules of grammar and discourse. These kinds of flexibilities are costly in terms of program space, so we compromise. The program here offers three kinds of relationships for analyses, and presents only a limited number of queries.

Requirements for eliciting nodes and edges are more critical. To represent a person's knowledge about a topic, we must get as close as possible to an exhaustive listing of concepts (nodes). Furthermore, we must meticulously examine every possible relationship to assure that all real ones are included.

This program uses several tactics to help a person recall entitites in the domain being considered. At the beginning of a session, and periodically thereafter, a general elicitation question is presented, in the form: "What is an entity in the domain being considered?" The user is reminded of all the entities that have been entered already. Once some entities have been specified, these are used to stimulate recall of more entities, using questions like: "What other entities are related to entity X in the domain being considered?" Ultimately, every recorded entity is used as a stimulus for obtaining more entities. Additionally, the user occasionally is asked to name an entity that differentiates some entities from others, in a query like: "X, Y, and Z are entities in the domain of interest; what entity might be implied by two of these?" While no methodology guarantees exhaustive recall of all entities, these techniques do promote extensive recall.

Definition of a new entity's relations is complex because a number of conditions have to be fulfilled in building a tree. These conditions are:

1. As we consider a new entity, we have to allow that it could be subordinate to any existing entity, and/or superordinate to any entity except the top one (which defines the domain of interest). That is, a new node might be positioned anywhere in the tree diagram except above the top node. In principle, this means that for every existing entity except the topmost one, the computer has to ask the user whether the new entity is a superordinate and/or subordinate. Fortunately, principles of logic and transitivity permit economies.

2. The complexity of hierarchical knowledge requires allowing any entity to have multiple subordinates or multiple superordinates.

3. Finally, proper tree structure calls for deleting any relations that can be inferred from transitivity.

The program assumes that a new entity is subordinate to the top node — it is in the domain of interest. Then a series of yes-no questions is asked to determine which existing entities are superordinates or subordinates of the new entity. Each query is of the form: "Does X (an existing entity) imply Y (the new entity)?" "Does Y imply X?"

The entities that are superordinate to the new entity are determined first. Querying works from the top of the tree downward, the procedure considers existing entities in topological order.

Let's say an existing entity is not superordinate to the new entity. Then subordinates of that existing entity are not superordinate to the new entity because of the transitivity principle.

Once a new entity's superordinates are known, more questions are asked to find its subordinates. The procedure employs two logical principles that follow from transitivity. First, if a new entity, Y, is not subordinate to an existing entity, X, then the new entity cannot be superordinate to any of X's subordinates. For example, since a sparrow is not a kind of mammal, various kinds of mammals cannot possibly be kinds of sparrows. Thus, no queries need to be presented regarding subordinates of entities which are not superordinate to the new entity. To find all of a new entity's subordinates, we only need to ask about the subordinates of entities for which the new entity is itself a subordinate.

Second, once we discover that the new entity is superordinate to an existing entity, we know that it is superordinate to all of that entity's subordinates. We do not want to represent those relations directly since they are derivable.

"Knowledge Trees" is written for the PET's 'upgrade' or 3.0 operating system. Notes have been provided to modify it to run on 1.0 and 4.0 systems. With the changes indicated, the program should work on 1.0 PETs. However, to run properly on 4.0 PETs, changes to the machine language portion of the program, beyond those indicated for PET ROM routines, will be necessary.

The problem is in the infamous 'garbage collection' routine. To speed up garbage collection (the process of removing old copies of dynamic strings), the 4.0 ROMs store a back pointer after the actual string characters in high memory. The garbage collection routine checks these back pointers to be sure they point to valid string descriptors in low memory. Bad strings are wiped out. The "Knowledge Trees" machine language routines do not accommodate these back pointers, and as a result, the system is likely to crash when a garbage collection takes place. MICRO will publish a fix for this problem in a future issue.

The transitivity principle eliminates queries about many possible relations, thereby substantially reducing the labor in establishing a new entity's position in the tree diagram. However, the potential for multiple subordinates and multiple superordinates requires that we ask many questions that might seem unnecessary.

To illustrate, suppose that "house pet" is added to the tree represented in figure 3 (abbreviated in figure 1). A house pet is a living creature, but queries establish that a house pet is not necessarily either a mammal or a bird. We then search for subordinates of house pet among the subordinates of living creature. Mammals are not all house pets, so mammal is not subordinate to house pet. Nevertheless, to avoid error, we must continue searching among the subordinates of mammal. It happens that dogs are a kind of house pet. Thus, in the new tree diagram, dog will be subordinate to both mammal and house pet; dog has two superordinates. Similarly, cat also is a subordinate of house pet. Having found these subordinates of house pet, we still must continue searching for other subordinates among the other subordinates of living creature. Searching through the bird subgraph yields no more subordinates in this example, but it would if we had canary as an entity.

Once all required queries have been answered, the program positions the new node by linking it to all of its immediate superordinates and all of its immediate subordinates. This is done by adding appropriate entries to the edge-list. If a superordinate and a subordinate of the new entity were originally connected by an arrow in the tree diagram, that relationship is deleted since it now can be derived using the transitivity principle.

Illustrative Analyses

One program option permits the analysis of the kinds of people at a scene. This elicitation provides a way to study people's subjective notions of social structure. The results can be related to issues of role identification, role conflict, and definitions of situations.

When the program has been loaded, the user is instructed on how to use the computer's keyboard. The user is given the instructions in exhibit 1 to read, and a sheet like exhibit 2 showing available commands.

The progression of questions and answers during the first part of a session might be as shown in exhibit 3. You might find a different structure if you did it yourself.

Choosing option 2 in answer to the first question would lead to the elicitation of happenings in an event. For example, analyzing what happens when we dine out might give the final results shown in exhibit 4. This elicitation could precede critical path analyses in order to define all the events that have to be considered. It also is an empirical approach to constructing "frames" for artificial intelligence research.

The BASIC Program

Input

The STOP key is disabled while the program is running (lines 340-350). A special input routine (lines 640-780) uses a shaded cursor to keep the program from ending if the user hits RETURN without making an entry. However, you can still break from a run by typing SPACE, then RETURN. The STOP key's function is turned on and off again during the input routine, so STOP is available if you break out of the program for programming purposes.

In the introductory part of the program (lines 370-510), a menu allows the user to choose one of three kinds of analysis.

1. *Analysis of a social scene.* This option elicits social roles and orders them by the "kind of" relation. Specifically, relations among roles are defined by asking "Is an X always a Y?" The name of the scene (the top node) and one role are elicited as part of the opening procedure (lines 880-1010).

2. *Analysis of an event.* Particular happenings are elicited, and then ordered in terms of their temporal priority: "Is X over before Y?" The definitional subroutine (lines 1030-1160) also gets the name of the general event (top node) and one specific happening.

Exhibit 1: Instructions

You can look at people in different ways. For example, the President of the United States is the President, a politician, a man, an adult, a husband, a father, and other things, too. Everyone can be seen in several ways.

How you see people depends a lot on where you see them. You might think of someone on the street as a pedestrian. The same person inside a store would be a customer or a browser or an employee. One idea about a person can lead to another. For example, teachers always are adults, so if people are talking about a teacher, you automatically know they are talking about an adult.

In this session with the microcomputer, you will name the scene you want to analyze. Thereafter the computer will ask you what kinds of people are found there, and how one type relates to another.

The machine begins by asking questions like, "Is a child always an adult?" But the computer learns what you tell it, so later questions are more interesting. In fact, you have to think carefully to answer some of them.

After you've named different types of people, the computer will analyze your answers and present the results in a chart. The chart has a row and a column for each type. Dots show which types are special cases of other types. For example, if professor and teacher were on the chart, a dot might appear in the professor row and the teacher column indicating that a professor is a kind of teacher.

Two dots in one column show what people have in common. For example, teacher and secretary might both have dots in an adult column, indicating that teachers and secretaries have adulthood as a bond between them. Two dots on one row suggest how a person might get confused in the situation. For example, professor might have a dot under teacher and another under researcher, indicating that sometimes a professor might feel torn between acting as a teacher and acting as a researcher.

Answer the first question by typing number 1 and pressing the RETURN key to analyze types of people.

The second question asks what kind of scene you want to analyze. Answer by typing a singular noun, like FAMILY, SCHOOL, or PARTY. Then press RETURN.

The next questions ask for types of persons you might find at the scene. Use just one or two words to name each type. Use nouns rather than adjectives — e.g., DUMMY rather than DUMB. Use singular rather than plural — e.g., TEACHER rather than TEACHERS. Type each name and press RETURN. If you misspell a word, you can correct it with the change command the next time you are asked to enter something.

Pretty soon the computer starts asking yes-no questions like, "Is a teacher always an adult?" Answer by typing Y or N. Answer yes-no questions for the ideal or general case. If you make a mistake while answering a yes-no question, continue answering Y until the yes-no questions are over. Then delete the entry with the change command so you can re-enter it correctly.

After the first few questions, the chart may appear automatically just before you name another type. When you are through looking at the chart, press the RETURN key, and the questions will start again. You also can get the chart by entering # instead of a name.

Enter $ to skip a question, if you cannot think of a good answer. (The $ option does not apply when answering yes-no questions.)

3. *Analysis of an entity.* Components of an entity and their incorporation relationships are elicited, using the relational question "Does X definitely have Y?" The name of the entity and of one of its parts are acquired while defining this option (lines 1180-1310).

Wordings of the various elicitation questions are adjusted for each option. You can change the wordings by changing the definitions of the string variables W3$, W4$, and W5$ (e.g., in lines 980-1000).

After these preliminaries, the program drops into a loop which continually cycles through all of the existing nodes in the graph (lines 530-620). The loop has no termination: the program ends when you press SPACE RETURN or turn off the machine.

The subroutine eliciting new entries (lines 1430-1840) ordinarily presents one of the existing entries as a stimulus, asking a question like: "What else might X relate to in the entity?" Existing entries that relate are listed as a reminder: "Aside from Y, Z?" The new entry is accepted, preceded by "a" or "an" (if that is appropriate), added to the name list, and another subroutine is called to place the new node in the tree.

Instead of entering a word in response to the elicitation question, the user can enter any of the program commands to correct, display, analyze, or save the data.

The main program loop calls a second elicitation subroutine (lines 2730-3010) that checks whether a node has more than two immediate subordinates. If so, another elicitation question is presented, such as: "Here are some parts of a W: X, Y, Z. What is a more general term for some of these?" The user may skip the question. If an answer is provided, it is treated in the same way that new entries ordinarily are handled.

The subroutine to establish a new node's position in the tree structure is the longest in the program (lines 1860-2710). First it clears some short-term memory for storing yes-no answers. Next it determines which existing nodes are superordinate to the new node. Then, working among the subordinates of these superordinates, it searches for existing nodes that are subordinate to the new node. The edge list is modified to reflect the new node's position.

An error-correction subroutine (lines 3680-4570) allows two kinds of changes to be made in the data. An entry in the name list can be changed to a different spelling (lines 3800-3850). Or a node can be deleted entirely from the graph (lines 3870-4570). In the latter case, the program checks whether subordinates are also to be deleted and, if so, deletes them first using the subroutine recursively. Recursion is achieved by defining a push-down 'stack' containing the nodes to be deleted, and letting the procedure work until the stack is empty. Deletion involves removing edges from the edge list and changing other edges in order to close up the graph over the deleted node. Deletion also involves removing the node from the name list and adjusting index numbers.

Output

A subroutine to display the matrix representation of the tree (lines 3130-3660) writes to screen or to printer. The lower triangle of the matrix is presented. Entry names, truncated to 13 characters, are listed in topological

Exhibit 2: Commands

$	Skip to next question without answering this one.
#	Look at the chart.
#H	Print the chart on paper (requires a printer hooked up to the computer).
!NAME	Change an entry that was entered as NAME. The program will instruct you on how to change the name, or delete the entry entirely.
@	Analyze commonalities of two or more entries. The program will tell you what more general term covers all of the chosen entries, and in what way each chosen entry is a special case of the more general term.
".FILE NAME	Save all of the data on tape so that they can be read in again later.
"\FILE NAME	Save all data on disk.
T	As an answer to program's first question, this causes the program to read data from tape.
D	As an answer to program's first question, this causes the program to read data from disk.

Exhibit 3: Sample Run

```
This program allows you to analyze the
organization of:
   1. people at a scene
   2. actions in an event
   3. the parts of an entity
Which analysis do you want to do? 1
What's a word naming the scene you're
going to analyze? FAMILY
What's one kind of actor you might find
at a family scene? FATHER
Who is an actor at a family scene — aside
from a father? MOTHER
Is a mother always a father? N
Is a father always a mother? N
Answer $ to skip:
What else might a father be at a family
scene? HUSBAND
```

Exhibit 3 (continued)

```
Is a husband always a father? N
Is a husband always a mother? N
Is a father always a husband? Y
Is a mother always a husband? N
```

The final results of this analysis might look as follows when the matrix representation is printed.

```
              AT A FAMILY SCENE

ADULT      X<
CHILD      X+<
MALE       X++<
FEMALE     X+++<
HUSBAND    +X+X+<
WIFE       +X++X+<
SON        ++XX+++<
DAUGHTER   ++X+X+++<
FATHER     +++++X+++<
MOTHER     ++++++X+++<
BROTHER    +++++++X+++<
SISTER     ++++++++X+++<
```

Exhibit 4: Sample Printout

```
               WHILE DINING OUT

LEAVING RESTA  X<
PAYING BILL    +X<
LEAVING TABLE  ++X<
LEAVING TIP    +++X<
WAITING FOR B  ++++X<
EATING FOOD    +++++X<
GETTING FOOD   ++++++X<
SITTING AT TA  +++++++X<
WAITING FOR F  +++++++X+<
GIVING ORDER   +++++++++X<
READING MENU   ++++++++++X<
WAITING FOR M  +++++++++++X<
FINDING TABLE  ++++++++X++X+<
ENTERING REST  ++++++++++++XX<

     WHILE DINING OUT
     LEAVING RESTAURANT
     PAYING BILL
     LEAVING TABLE
     LEAVING TIP
     WAITING FOR BILL
     EATING FOOD
     GETTING FOOD
     SITTING AT TABLE
     WAITING FOR FOOD
     GIVING ORDER
     READING MENU
     WAITING FOR MENU
     FINDING TABLE
     ENTERING RESTAURANT
```

order along the lefthand border (the full names are listed separately at the bottom when a hard copy is printed). Cells with zeros are left blank; cells with ones are marked graphically. Screen displays are limited to trees with less than 23 nodes; printed output covers the maximum size tree that the program handles — 63 nodes. The screen display is invoked randomly about one-third of the time after new entries are made (line 1830).

Common superordinates of two or more entries are found by another subroutine (lines 5070-5600). This procedure also indicates the immediate subordinates of the common superordinate. These subordinates lead down to each of the originally specified nodes — an analysis not easily done by inspection of the matrix. The algorithm involves concatenating all the up-graphs of the focus nodes, creating a dummy subordinate node of the focus nodes, and subtracting the up-graph of the dummy from the concatenated graphs (repeatedly if there are more than two focus nodes). The topologically lowest node of the remainder represents a commonality of the focus nodes. A search is made to find the immediate subordinates of the common node that are linked to focus nodes. When multiple common nodes exist, they all are presented in turn. The dummy node is deleted before the subroutine ends.

The routine to save a knowledge base on tape or disk (lines 4590-4810) creates a file name from the label given as part of the save command, plus the name of the top node in the graph, as follows: LABEL.NODE1. The file contains the type of analysis as specified for the program's first question, the number of nodes, the node names, the number of edges, and the list of edges. The file can be read later by another routine (lines 4830-5050), called by typing T or D in answer to the first program question (data are listed as they are read into the program). Writing and reading are done with tape unit #1 or disk unit #0. A disk must be properly initialized before the program is run.

Utilities

Articles are appended to the front of entries for readability (lines 1330-1410). This routine inserts a nonprinting character with 'a' to make the appendage uniform in length, simplifying removal whenever necessary. Articles are not added to event entries (option 2 in the program).

A subroutine (lines 800-860) presents the frequent query about relationships (e.g., "Is an X always a Y?"). The routine gets the answer and returns. A separate routine (lines 3030-3110) is used to add a name to the name list.

Loader Program

A separate loading program is shown in listing 2. This should be saved as the first program on a tape or disk. When it is loaded and RUN, it automatically loads the assembly language subroutines, guards the memory allocated to them, and then loads the main BASIC program and starts it running. On tape, the programs must be saved in the following order: loader (named anything); the file of assembly language subroutines, named CODE; main BASIC program, named TREES. These same file names must be implemented in order to use the loader program with a disk.

Entering the BASIC Code

A Glossary of special symbols used in the program listings is given at the

(Continued on page 74)

(Continued from page 72)
end of the main program (lines 5620-5700).

REMarks must be ignored when entering the main BASIC program into a 16K PET because all of the lines in listing 1 take up 18K. REMarks could be included when using a 32K PET, but then the assembly language subroutine would have to be relocated.

6502 Assembly-Language Routine

A number of analytic functions are coded in 6502 assembly language for the sake of speed. The code is presented in listing 3, which begins with definitions of zero-page cells and the ROM subroutines used. These locations refer to Operating System 3.0 on the PET. Notes are included, where appropriate, for conversion to O.S. 1.0 and 4.0.

The short routine at lines 370-460 (called before the BASIC program is loaded) points the USR function to the code and protects the code from BASIC. The overall logic of the procedure can be seen in the executive program (lines 510-560).

A number of parameters have to be established every time the routine is called (lines 610-1630). The number of nodes is determined by finding the name list and counting the number of entries in it. The routine looks for a specific array, so the name list *must* be NM$( ). Next the routine finds the location and the length of the edge list, which *must* be named L$(0). The name list and the edge list — and later two graphs named GR$(0) and BR$(0) — are found using a separate subroutine (lines 1100-1630). If an array is not found, a pointer is set to the word "array" among the PET's canned messages, and control is transfered to the standard error routine in ROM.

Three parameters are passed from BASIC when the USR function is invoked: the starting node from which to begin tracing a subgraph, a code indicating whether to trace upwards (0) or downwards (2), and a code indicating whether the list of encountered nodes should (1) or should not (0) contain the same element if the trace goes through it repeatedly. The no-repeat option usually is used in this application. The three numbers are combined into a single argument for USR; the starting node is added to the two code values, each multiplied by 256. The USR function transfers this number to FACC — the BASIC accumulator — in five-byte floating point format. After the number has been converted to two-byte integer format (using the ROM routine, FLPINT), the parameters are recovered. The low byte of the argument equals the specified starting node. The high byte of the argument is ANDed with one to recover the Repeat code. The low byte is ANDed with two to recover the Up-Down code.

Lines 1670-2280 create a topological list of node index numbers, using the second tape cassette buffer as a work space. The list is not returned directly; it is used to order nodes encountered while tracing a subgraph. The procedure follows an algorithm presented by Gotlieb and Gotlieb (1978). If a loop is encountered in the graph, then the routine aborts and prints "complex error."

The routine in lines 2330-3050 makes a list of all the nodes in a graph that are reachable, either in an upward direction or a downward direction, from the specified starting node. The index numbers are converted to characters and stored in a string named GR$(0), which must be dimensioned in the BASIC program. The length of GR$(0) is returned as the value of the USR function.

The GR$(0) list is put into topological order by lines 3090-3350. If the starting node is specified as one, with direction down, then GR$(0) will return with a topological listing of all nodes in the graph.

The final part of the procedure, lines 3390-3760, removes elements in the GR$(0) list from another list of elements, BR$(0). This array must be dimensioned in the BASIC program. If none of the members of GR$(0) is in BR$(0), then BR$(0) remains unchanged. If elements of GR$(0) are in BR$(0), then they are removed, making BR$(0) shorter. BR$(0) must always be defined with a space following it (e.g., see line 2070 of the BASIC program).

Every USR call performs all of these functions, even if some of them are not used.

Entering the Code

The procedure has been assembled for placement at the top of RAM in a 16K computer, and the same positioning can be used with a 32K machine. The positioning can be changed to the top of 32K RAM by changing every hex memory address in the $3000 range so that it begins with seven instead of three. Enter the code with the CBM monitor, display the relevant cells with the M command, and overwrite the contents following listing 3. Use the following monitor command to save to disk.

```
.S "0:CODE",08,3DA8,3FF0
```

Use the following command to save on tape.

```
.S "CODE",01,3DA8,3FF0
```

Bibliography

1. C.C. Gotlieb and L.R. Gotlieb, *Data Types and Structures*, (Prentice-Hall, 1978).
2. J.P. Spradley and D.W. McCurdy, *The Cultural Experience*, (Science Research Associates, 1971).
3. P.H. Winston, *Artifical Intelligence*, (Addison-Wesley, 1979).

David Heise is Professor of Sociology at Indiana University. He recently edited "Microcomputers in Social Research" (Sage Publications, 1981). His books include *Causal Analysis* and *Understanding Events*. He has created commercial microcomputer programs for word processing and statistical analysis, and he has published a number of programs for PET computers. Contact Mr. Heise at the Department of Sociology, Indiana University, Bloomington, IN 47405.

***Listing 2:* Loader Program**

```
100 : REM: LOADER PROGRAM
110 :
120 : REM: ROUTINE TO LOAD THE ASSEMBLED CODE, THEN THE MAIN PROGRAM.
130 DV$=",8":CU$="■■■■■"
140 REM DV$="":CU$=CU$+"■"   :REM ADD THIS LINE TO LOAD FROM TAPE.
150 : REM: SET GRAPHICS MODE.
160 POKE 59468,12
170 : REM: STORE 4 CARR. RETURNS IN INPUT BUFFER,
    (USE 525 AND 527-530 FOR 1.0)
180 POKE 158,4:POKE 623,13:POKE 624,13:POKE 625,13:POKE 626,13
190 : REM: SET UP SCREEN TO INVOKE THE SUBROUTINE FILE -- 'CODE',
200 PRINT "■■■LOAD" CHR$(34) "CODE" CHR$(34)DV$:PRINTCU$"SYS 15784"
210 : REM: THEN LOAD PROGRAM 'MAIN'.
220 PRINT "■■LOAD" CHR$(34) "MAIN" CHR$(34)DV$
230 PRINT CU$ "RUN■";:END
```

Listing 1: Tree Analysis

```
100 : REM: ***TREE ANALYSIS***
110 :
120 : REM: THE ASSEMBLER ROUTINES ARE CALLED BY THE USR FUNCTION, AS FOLLOWS:
130 : REM: LL=USR(START+ DIRECTION + REPEATS)
140 : REM: WHERE: 'START' IS THE INDEX NUMBER OF THE REFERENCE NODE;
150 : REM: 'DIRECTION' INDICATES DIRECTION OF SEARCH;
160 UP=0:DOWN=512
170 : REM: REPEATS=256 ALLOWS REPETITION OF NODES.
180 : REM: ANALYSES REQUIRE A LIST OF EDGES STORED IN L$(0).
190 : REM: NODES REACHABLE FROM 'START' ARE RETURNED IN GR$(0), LENGTH LL, IN
200 : REM: TOPOLOGICAL ORDER. THESE NODES ARE DELETED FROM BR$(0).
210 :
220 : REM: MAKE STRING ARRAYS FOR NAMES (NM$), EDGES (L$), AND SEARCHES (GR$).
230 DIM NM$(65), L$(0), GR$(0)
240 : REM: DEFINE WORK ARRAYS.
250 DIM OD%(65), BR$(0):BR$(0)=" "
260 : REM: DEFINE A SPACING STRING WITH 16 BLANKS.
270 SP$="                "
280 :
290 : REM: MAKE A FUNCTION TO CONVERT A CHARACTER IN L$ TO INDEX NUMBER.
300 DEF FNL(Z)=ASC(MID$(L$(0),Z,1))-64
310 : REM: MAKE A FUNCTION TO CONVERT A CHARACTER IN GR$ TO INDEX NUMBER.
320 DEF FNG(Z)=ASC(MID$(GR$(0),Z,1))-64
330 :
340 : REM: DISABLE STOP KEY (OLD ROM: POKE 537,136; OS 4.0: POKE 144,88)
350 POKE 144,49
360 :
370 : REM PRESENT BEGINNING MENU.
380 PRINT "<s>THIS PROGRAM ALLOWS YOU TO ANALYZE
390 PRINT "THE ORGANIZATION OF:":PRINT
400 PRINT "  1 PEOPLE AT A SCENE
410 PRINT "  2 ACTIONS IN AN EVENT
420 PRINT "  3 THE PARTS OF AN ENTITY
430 PRINT "<Q>WHICH ANALYSIS DO YOU WANT TO DO";:GOSUB 670
440 : REM: INPUT A FILE?
450 IF WD$="T" OR WD$="D" THEN GOSUB 4860:GOTO 540
460 : REM: DO INITIAL SETUPS.
470 PRINT "<s>";:TA=VAL(WD$):ON TA GOSUB 890,1040,1190
480 : REM: NOW HAVE DEFINED THE GRAPH ORIGIN AND ONE OTHER NODE.
490 NM$(2)=WD$
500 : REM: THUS THERE ARE 2 NODES AND ONE EDGE FROM ORIGIN TO OTHER NODE.
510 IL=2:L$(0)="AB"
520 :
530 : REM: USE EXISTING NODES TO ELICIT MORE NODES.
540 FOR IP=1 TO IL
550 : REM: USE ONLY A GENERAL ELICITATION QUESTION FOR EVENTS
560 IF TA=2 THEN IP=1
570 : REM: FIND RELATED WORDS.
580 GOSUB 1450
590 : REM: ELICT GROUPING WORDS WHEN THERE ARE MORE THAN 7 NODES.
600 IF IL>7 THEN GOSUB 2750
610 NEXT IP
620 GOTO 540
630 :
640 : REM: INPUT WORDS, GUARDING AGAINST BREAKOUT
650 : REM: RESTORE STOP KEY IN CASE PROGRAMMER BREAKS OUT BY ENTERING SPACE
660 : REM: (OLD ROM: POKE 537,133; OS 4.0: POKE 144,85)
670 POKE 144,46
680 : REM: FIND CURSOR POSITION
690 : REM: (A=226 FOR OLD ROM)
700 A=198:X=PEEK(A)
710 : REM: GET INPUT STRING
720 INPUT "<]><]>[&]<}><}><}>";WD$
730 : REM: IGNORE IF USER PRESSES RETURN WITHOUT PRIOR KEYSTROKE
740 IF WD$="[&]" THEN PRINT"<q>";:POKE A,X:GOTO 720
750 : REM: DISABLE STOP KEY
760 : REM: (OLD ROM: POKE 537,136; OS 4.0: POKE 144,88)
770 POKE 144,49
780 RETURN
```

Tree Analysis *(Continued)*

```
790 :
800 : REM: SUBROUTINE TO PRINT LINKAGE QUESTION AND GET ANSWER.
810 IF TA=3 THEN PRINT "DOES"W2$W5$W1$"? ";:GOTO 830
820 PRINT "IS"W1$W5$W2$"? ";
830 GET A$:IF A$="" THEN 830
840 IF A$<>"Y" AND A$<>"N" THEN 830
850 PRINT A$
860 RETURN
870 :
880 : REM: SET UP PEOPLE-AT-A-SCENE.
890 PRINT "WHAT'S A WORD NAMING THE SCENE
900 PRINT "YOU'RE GOING TO ANALYZE";
910 : REM: GET FIRST NODE, X; CONVERT TO 'AT AN X SCENE'.
920 GOSUB 670:GOSUB 1340
930 NM$(1)=" AT"+WD$+" SCENE"
940 : REM: GET 2ND NODE.
950 PRINT:PRINT "WHAT'S ONE KIND OF ACTOR YOU MIGHT": PRINT "FIND" NM$(1);
960 GOSUB 670:GOSUB 1340
970 : REM: SET UP ELICITATION QUESTIONS.
980 W3$="WHO IS AN ACTOR": REM (...) AT THE SCENE?
990 W4$=" BE": REM  WHAT ELSE MIGHT Y (BE) AT THE SCENE?
1000 W5$=" ALWAYS": REM  IS Y (ALWAYS) Z?
1010 RETURN
1020 :
1030 : REM: SET UP ACTIONS-IN-AN-EVENT.
1040 PRINT "WHAT'S A PHRASE NAMING THE EVENT YOU'RE
1050 PRINT "GOING TO ANALYZE
1060 PRINT "(E.G., 'DINING OUT')";:GOSUB 670
1070 : REM: CREATE FIRST NODE: 'WHILE X'.
1080 NM$(1)=" WHILE "+WD$
1090 : REM: GET 2ND NODE.
1100 PRINT:PRINT "HOW MIGHT "WD$
1110 PRINT "BEGIN";:GOSUB 670:GOSUB 1340
1120 : REM: SET UP ELICITATION QUESTIONS.
1130 W3$="WHAT IS AN OCCURANCE"+CHR$(13): REM (...) WHILE DOING?
1140 W4$=" RELATE TO": REM  WHAT ELSE MIGHT X (RELATE TO) WHILE DOING?
1150 W5$=CHR$(13)+" OVER BEFORE"+CHR$(13): REM  IS X (OVER BEFORE) Y?
1160 RETURN
1170 :
1180 : REM: SET UP PARTS-OF-AN-ENTITY.
1190 PRINT "WHAT'S THE NAME OF THE ENTITY (E.G.,
1200 PRINT "'A CAR', 'MY BIKE')";:GOSUB 670
1210 : REM: CREATE 1ST NODE: 'IN X'.
1220 NM$(1)=" IN "+WD$
1230 : REM: GET 2ND NODE.
1240 PRINT:PRINT "WHAT'S ONE THING THAT MIGHT BE PART
1250 PRINT "OF " WD$;
1260 GOSUB 670:GOSUB 1340
1270 : REM: SET UP ELICITATION QUESTIONS.
1280 W3$="WHAT IS TO BE FOUND": REM (...) IN THING?
1290 W4$=" RELATE TO": REM  WHAT ELSE MIGHT X (RELATE TO) IN THING?
1300 W5$=" DEFINITELY HAVE"+CHR$(13): REM  DOES X (DEFINITELY HAVE) Y?
1310 RETURN
1320 :
1330 : REM: ADD AN ARTICLE TO WORD.
1340 IF TA=2 THEN WD$=" "+CHR$(6)+CHR$(6)+CHR$(6)+WD$:RETURN
1350 : REM: (CHR$(6) IS NONPRINTING, USED TO MAKE AR$ 4 CHARACTERS LONG.)
1360 AR$=" A "+CHR$(6)
1370 FOR H=1 TO 5
1380 IF LEFT$(WD$,1)=MID$("AEIOU",H,1) THEN AR$=" AN "
1390 NEXT H
1400 WD$=AR$+WD$
1410 RETURN
1420 :
1430 : REM: SUBROUTINE TO ELICIT A NEW WORD.
1440 : REM: PRINT INSTRUCTIONAL MESSAGES.
1450 IF IL>3 THEN PRINT "<Q>TYPE <R>!<r> FOLLOWED BY FIRST PART OF ENTRY
1455 IF IL>3 THEN PRINT "TO REVISE
1460 IF IL>2 THEN PRINT "<Q>ANSWER <R>$<r> TO SKIP:<Q>
```

Tree Analysis *(Continued)*

```
1470 PRINT
1480 : REM: USE A SPECIAL QUESTION IF USING THE TOP NODE FOR ELICITATION.
1490 IF IP=1 THEN PRINT W3$NM$(1):GOTO 1540
1500 : REM: OTHERWISE PRINT THE STANDARD STEM
1510 PRINT "WHAT ELSE MIGHT
1520 PRINT RIGHT$(NM$(IP),LEN(NM$(IP))-1)W4$NM$(1)
1530 : REM: SHOWING PRESENT LINKS, IF ANY.
1540 IO=LEN(L$(0)):FOR I=1 TO IO STEP 2
1550 IF FNL(I)=IP OR (FNL(I+1)=IP AND FNL(I)<>1) THEN I=IO:NEXT I:GOTO 1570
1560 NEXT I:GOTO 1640
1570 PRINT "--ASIDE FROM
1580 BR$(0)="  ":DI=2:LL=USR(IP+DOWN)
1590 FOR J=1 TO LL
1600 IF FNG(J)>1 AND FNG(J)<>IP THEN PRINT "  "NM$(FNG(J))
1610 NEXT J
1620 IF DI THEN DI=0:LL=USR(IP+UP):GOTO 1590
1630 : REM: GET USER'S INPUT.
1640 GOSUB 670:A$=LEFT$(WD$,1)
1650 : REM: ALLOW SKIPPING.
1660 IF A$="$" THEN RETURN
1670 : REM: ACCEPT A PRINT-GRID COMMAND.
1680 IF A$="#" THEN GOSUB 3160:GOTO 1450
1690 : REM: ACCEPT REVISE COMMAND.
1700 IF A$="!" THEN GOSUB 3700:GOTO 1450
1710 : REM: ACCEPT SAVE COMMAND.
1720 IF A$="." OR A$="\" THEN GOSUB 4620:GOTO 1450
1730 : REM: ACCEPT ANALYZE COMMAND.
1740 IF A$="@" THEN 5090:GOTO 1450
1750 : REM: THIS IS AN ENTRY!
1760 : REM: ADD AN ARTICLE TO THE WORD.
1770 GOSUB 1340
1780 : REM: ADD THE NODE TO NM$.
1790 H=IL:GOSUB 3060:IF H=IL THEN PRINT "<R>ALREADY EXISTS":GOTO 1450
1800 : REM: FIND THE NODE'S LINKAGES.
1810 GOSUB 1880
1820 : REM: GIVE AUTOMATIC PRESENTATION OF THE GRID OCCASIONALLY.
1830 IF RND(1)<.3 THEN WD$="#":GOSUB 3160
1840 RETURN
1850 :
1860 : REM: SUBROUTINE TO POSITION NEW NODE IN TREE.
1870 : REM: CLEAR SPACE FOR REMEMBERING YES-NO ANSWERS.
1880 FOR I=1 TO 65:OD%(I)=0:NEXT I
1890 : REM: GET A DOWN GRAPH FROM THE ORIGIN, WITHOUT REPEATS.
1900 BR$(0)="  ":LQ=USR(1+DOWN)
1910 : REM: IS NEW NODE SUBORDINATE TO AN EXISTING NODE?
1920 FOR IB=1 TO LQ
1930 : REM: STOP IF BEYOND THE LENGTH OF CURRENT GR$.
1940 IF IB>LEN(GR$(0)) THEN IB=LQ:GOTO 2140
1950 : REM: AND SKIP THE QUESTION FOR THE ORIGIN.
1960 IF FNG(IB)=1 THEN NEXT IB
1970 : REM: IF QUESTION WAS ASKED FOR THIS NODE, RECALL THE ANSWER.
1980 IF OD%(FNG(IB))<>0 THEN A$=CHR$(OD%(FNG(IB))):GOTO 2040
1990 : REM: OTHERWISE PRESENT QUESTION AND GET YES-NO ANSWER.
2000 W1$=NM$(ZZ):W2$=NM$(FNG(IB)):GOSUB 810
2010 : REM: (REMEMBER THE ANSWER.)
2020 OD%(FNG(IB))=ASC(A$)
2030 : REM: IF NEW NODE IS NOT SUBORDINATE TO A NODE,
2040 IF A$<>"N" THEN NEXT IB:GOTO 2160
2050 : REM: THEN REMOVE THAT NODE AND ITS SUBORDINATES FROM THE DOWN GRAPH.
2060 : REM: BY SETTING BR$ EQUAL TO A DUPLICATE OF GR$,
2070 BR$(0)=LEFT$(GR$(0)+" ",LEN(GR$(0)))
2080 : REM: REMOVING UNWANTED NODES FROM BR$,
2090 LL=USR(FNG(IB)+DOWN)
2100 : REM: THEN SETTING GR$ EQUAL TO BR$.
2110 GR$(0)=LEFT$(BR$(0),LEN(BR$(0)))
2120 : REM: RELOOK AT THE REVISED GR$.
2130 IB=1
2140 NEXT IB
2150 : REM: CLEAR SPACE FOR REMEMBERING YES-NO ANSWERS.
```

Tree Analysis *(Continued)*

```
2160 FOR I=1 TO 65:OD%(I)=0:NEXT I
2170 : REM: CREATE A DUPLICATE OF THE CURRENT PRIMARY GRAPH.
2180 GG$=LEFT$(GR$(0),LEN(GR$(0)))
2190 : REM: WORK BACKWARD IN THE PRIMARY GRAPH UNTIL ALL NODES ARE DELETED.
2200 IB=LEN(GR$(0)):IF IB=0 THEN RETURN
2210 :REM: MAKE A LINK IN EDGE LIST FROM THIS NODE TO NEW NODE.
2220 L$(0)=L$(0)+RIGHT$(GR$(0),1)+CHR$(ZZ+64)
2230 : REM: GET THE DOWN-GRAPH FROM NODE IB.
2240 BI=FNG(IB)
2250 BR$(0)="  ":LL=USR(BI+DOWN)
2260 : REM: LOOK AT NEXT NODE IN THE DOWN GRAPH,
2270 FOR BB=1 TO LL
2280 : REM: UNTIL ALL ARE CONSIDERED.
2290 IF BB>LEN(GR$(0)) THEN NEXT BB:GOTO 2690
2300 : REM: SKIP THE NEW NODE OR NODE IB.
2310 IF FNG(BB)=ZZ OR FNG(BB)=BI THEN NEXT BB: GOTO 2690
2320 : REM: IF QUESTION WAS ASKED FOR THIS NODE, RECALL ANSWER.
2330 K=OD%(FNG(BB)):IF K<>0 THEN A$=CHR$(K):GOTO 2390
2340 : REM: OTHERWISE ASK IF THIS NODE IS SUBORDINATE TO THE NEW NODE.
2350 W1$=NM$(FNG(BB)):W2$=NM$(ZZ):GOSUB 810
2360 : REM: (REMEMBER THE ANSWER.)
2370 OD%(FNG(BB))=ASC(A$)
2380 : REM: WHEN THIS NODE, BB, IS SUBORDINATE TO NEW NODE,ZZ,
2390 IF A$="N" THEN NEXT BB:GOTO 2690
2400 : REM: SEE IF BB NODE IS DIRECTLY CONNECTED TO IB NODE.
2410 A$=LEFT$(GR$(0),1)+MID$(GR$(0),BB,1)
2420 FOR BC=1 TO LEN(L$(0)) STEP 2
2430 B$=MID$(L$(0),BC,2)
2440 : REM: IF IT IS NOT, THEN ADD EDGE FROM NEW NODE TO BB NODE
2450 IF A$=B$ THEN 2510
2460 NEXT BC
2470 : REM: (PROVIDING IT'S NOT ALREADY PRESENT).
2480 IF K<>ASC("Y") THEN L$(0)=L$(0)+CHR$(ZZ+64)+MID$(GR$(0),BB,1)
2490 GOTO 2610
2500 : REM: IF IT IS, SUBSTITUTE EDGE ZZ-BB FOR OLD EDGE IB-BB
2510 C$=CHR$(ZZ+64)+MID$(L$(0),BC+1,1)
2520 : REM: (PROVIDING IT'S NOT ALREADY THERE).
2530 FOR IO=1 TO LEN(L$(0)) STEP 2:IF MID$(L$(0),IO,2)=C$ THEN C$=""
2540 NEXT IO
2550 A$="":IF BC>1 THEN A$=LEFT$(L$(0),BC-1)
2560 B$="":IF BC<(LEN(L$(0))-2) THEN B$=RIGHT$(L$(0),LEN(L$(0))-BC-1)
2570 : REM: (CLOSE THE BC LOOP.)
2580 BC=LEN(L$(0)):NEXT BC
2590 L$(0)=A$+C$+B$
2600 : REM: REMOVE BB AND ITS SUBORDINATE NODES FROM THE IB DOWN-GRAPH,
2610 BR$(0)=LEFT$(GR$(0),LEN(GR$(0)))
2620 LL=USR(FNG(BB)+DOWN)
2630 GR$(0)=LEFT$(BR$(0),LEN(BR$(0)))
2640 :REM : AND RELOOK AT IT.
2650 BB=BB-1
2660 NEXT BB
2670 : REM: IF ALL SUBORDINATES OF IB ARE DONE. RETURN TO PRIMARY GRAPH.
2680 : REM: REMOVE NODES UPWARD FROM IB,
2690 BR$(0)=LEFT$(GG$+" ",LEN(GG$)):LL=USR(BI+UP)
2700 : REM: AND CONTINUE.
2710 GR$(0)=LEFT$(BR$(0),LEN(BR$(0))):GOTO 2180
2720 :
2730 : REM: SUBROUTINE TO ELICIT GROUPING WORD.
2740 : REM: WHEN CURRENT NODE HAS 3+ IMMEDIATE SUBORDINATES,
2750 J=0:FOR I=1 TO LEN(L$(0)) STEP 2:IF CHR$(IP+64)=MID$(L$(0),I,1) THEN J=J+1
2760 NEXT I:IF J<3 THEN RETURN
2770 : REM: PRINT ELICITATION STIMULUS, ADJUSTING FOR KIND OF ANALYSIS.
2780 I=3:W1$="":ON TA GOTO 2790,2810,2830
2790 PRINT:PRINT "HERE ARE KINDS OF ":IF IP=1 THEN W1$="PARTICIPANTS":I=0
2800 GOTO 2840
2810 PRINT:PRINT "HERE ARE ASPECTS OF ":IF IP=1 THEN W1$="WHAT HAPPENS":I=0
2820 GOTO 2840
2830 PRINT:PRINT "HERE ARE SOME PARTS":W1$=" OF":IF IP=1 THEN W1$="OF":I=0
```

Tree Analysis *(Continued)*

```
2840 PRINT W1$RIGHT$(NM$(IP),LEN(NM$(IP))-I)"."
2850 : REM: PRINT IMMEDIATE SUBORDINATES OF NODE IP.
2860 FOR I=1 TO LEN(L$(0)) STEP 2
2870 IF CHR$(IP+64)=MID$(L$(0),I,1) THEN PRINT " "NM$(FNL(I+1))
2880 NEXT I
2890 : REM: PRINT ELICITATION QUESTION AND GET ANSWER.
2900 PRINT:PRINT " WHAT IS A MORE GENERAL TERM FOR
2910 PRINT " SOME OF THESE (<R>$<r>: SKIP)":GOSUB 670
2920 : REM: PERMIT SKIP.
2930 IF WD$="$" THEN RETURN
2940 : REM: ACCEPT PRINT-GRID COMMAND.
2950 IF LEFT$(WD$,1)="#"THEN GOSUB 3160:GOTO 2750
2960 : REM: ADD AN ARTICLE TO NEW WORD, ENTER IN NM$, AND LINK IT INTO TREE.
2970 GOSUB 1340:H=IL:GOSUB 3060:IF H=IL THEN PRINT"<R>ALREADY EXISTS":GOTO2750
2980 GOSUB 1880
2990 : REM: SHOW GRID.
3000 GOSUB 3160
3010 RETURN
3020 :
3030 : REM: SUBROUTINE TO ADD A WORD TO THE LIST OF WORDS, NM$.
3040 : REM: IF WORD ALREADY IS IN NM$
3050 : REM: THEN CLOSE LOOP AND RETURN.
3060 FOR JJ=2 TO IL
3070 IF WD$=LEFT$(NM$(JJ),LEN(WD$)) THEN ZZ=JJ:JJ=IL:NEXT JJ:RETURN
3080 NEXT JJ
3090 : REM: OTHERWISE ADD THE WORD TO NM$.
3100 ZZ=JJ:IL=JJ:NM$(JJ)=WD$
3110 RETURN
3120 :
3130 : REM: SUBROUTINE TO DISPLAY RESULTS ON GRID.
3140 : REM: ANSWER '#H' GIVES HARD COPY.
3150 : REM: SET DEVICE # FOR SCREEN OR PRINTER.
3160 DV=3:IF MID$(WD$,2,1)="H" THEN DV=4
3170 : REM: SCREEN OUTPUT LIMITED TO 22 NODES.
3180 IF DV=3 AND MS>0 AND IL>22 THEN RETURN
3190 : REM: IF ON THE 23RD NODE, PRINT ONE-TIME MESSAGE.
3200 MS$="TOO MANY ENTRIES TO DISPLAY GRID ON      SCREEN.
3210 IF MS=0 AND IL>23 THEN PRINT MS$:MS=1:RETURN
3220 : REM: SET GRAPHIC CHARACTERS FOR SCREEN OR PRINTER.
3230 LN$=CHR$(219):MK$=CHR$(209):DG$=CHR$(95)
3240 IF DV=4 THEN LN$="+":MK$="X":DG$="<"
3250 : REM: OPEN PRINTING FILE.
3260 OPEN 1,DV
3270 : REM: CLEAR SCREEN AND PRINT HEADER.
3280 PRINT "<s>";
3290 PRINT#1,MID$(NM$(1),2):PRINT#1
3300 : REM: SEARCH DOWN TO GET ORDERED LIST OF NODES (GR$) WITHOUT REPEATS.
3310 BR$(0)="  ":LL=USR(1+DOWN)
3320 : REM: FOR EACH WORD IN TOPOLOGICAL ORDER,
3330 FOR IO=2 TO IL
3340 : REM: REMOVE ARTICLE,
3350 W1$=MID$(NM$(FNG(IO)),5)
3360 : REM: AND TRUNCATE IF TOO LONG.
3370 W1$=MID$(W1$,1,13)
3380 : REM: THEN PRINT ON NEXT ROW, FOLLOWED BY SPACES TO EDGE OF GRID.
3390 PRINT#1:PRINT#1,LEFT$(W1$+SP$,15);
3400 : REM: GENERATE GRID CHARACTERS.
3410 FOR JO=1 TO IO-1:OD%(JO)=ASC(LN$):NEXT JO:OD%(IO)=ASC(DG$)
3420 : REM: SET SPACING INDEX.
3430 JL=1
3440 : REM: GO THROUGH RECEIVER NODES IN EDGE-LIST TO FIND THIS NODE.
3450 FOR JO=2 TO LEN(L$(0)) STEP 2
3460 IF MID$(L$(0),JO,1)<>MID$(GR$(0),IO,1) THEN NEXT JO:GOTO 3550
3470 : REM: NOW FIND SENDER-NODE'S TOPOLOGICAL ORDER,
3480 FOR JJ=1 TO LL
3490 IF MID$(L$(0),JO-1,1)<>MID$(GR$(0),JJ,1) THEN NEXT JJ
3500 : REM: INSERT A LINK MARK IN THE GRID LINE,
3510 OD%(JJ)=ASC(MK$)
3520 : REM: AND CLOSE LOOPS.
```

Tree Analysis *(Continued)*

```
3530 JJ=LL:NEXT JJ,JO
3540 : REM: WHEN LINE IS FINISHED, PRINT IT.
3550 FOR JO=1 TO IO:PRINT#1,CHR$(OD%(JO));:NEXT JO
3560 : REM: CONTINUE THROUGH ALL LINES.
3570 NEXT IO
3580 : REM: IF DOING HARD COPY, PRINT OUT NAMES.
3590 IF DV=4 THEN PRINT#1:FOR IO=1 TO IL:PRINT#1,NM$(FNG(IO)):NEXT IO
3600 : REM: CLOSE THE PRINT FILE, AND PRINT A MESSAGE.
3610 CLOSE 1
3620 PRINT"<S>":PRINT" <R>TYPE RETURN":PRINT" <R>TO CONTINUE
3630 : REM: WAIT FOR USER RESPONSE.
3640 GET A$:IF A$="" THEN 3640
3650 : REM: CLEAR SCREEN AND RETURN TO ELICITATION.
3660 PRINT "<s>";:RETURN
3670 :
3680 : REM: SUBROUTINE TO REVISE A NODE.
3690 : REM: REMOVE ! FROM FRONT OF WORD, ADD ARTICLE, AND FIND IN NM$ LIST.
3700 WD$=RIGHT$(WD$,LEN(WD$)-1):K=IL:GOSUB 1340:GOSUB 3060
3710 : REM: ABORT IF WORD WAS NOT IN NM$ LIST.
3720 IF K<IL THEN NM$(IL)="":IL=IL-1:PRINT "<R>NOT FOUND":RETURN
3725 PRINT "<s>FOUND":PRINT "<R>"NM$(ZZ)
3730 : REM: PRINT MENU.
3740 PRINT "YOU CAN <R>C<r>HANGE THE NAME,
3750 PRINT "      OR <R>D<r>ELETE THE WORD ENTIRELY,
3755 PRINT "      OR <R>E<r>XIT.
3760 PRINT "<Q>ENTER <R>C<r>, <R>D<r>, OR <R>E<r>.
3770 GET A$:IF A$="" OR (A$<"C" AND A$>"E") THEN 3770
3780 : REM: GO TO SUBROUTINE FOR DELETION.
3785 IF A$="E" THEN RETURN
3790 IF A$="D" THEN GOSUB 3890:RETURN
3800 : REM: DO NAME CHANGE.
3810 PRINT "<Q>NEW NAME FOR <R>"RIGHT$(NM$(ZZ),LEN(NM$(ZZ))-3)
3820 PRINT "<r>";:GOSUB 670
3830 : REM: ADD ARTICLE AND REPLACE.
3840 GOSUB 1340:NM$(ZZ)=WD$
3850 RETURN
3860 :
3870 : REM: SUBROUTINE TO DELETE A NODE.
3880 : REM: PUT NODE TO BE REMOVED ON 'STACK'.
3890 OD%(0)=1:OD%(1)=ZZ
3900 : REM: GET LAST NODE, X, ON STACK.
3910 X=OD%(OD%(0))
3920 : REM: DEFINE ITS DOWN-GRAPH,
3930 BR$(0)="  ":LQ=USR(X+DOWN)
3940 : REM: WITHOUT THE X NODE.
3950 GR$(0)=RIGHT$(GR$(0),LQ-1)
3960 : REM: SAVE GR$.
3970 GG$=LEFT$(GR$(0),LEN(GR$(0)))
3972 : REM: WHEN X HAS SUBORDINATES,
3974 IF LEN(GR$(0))=0 THEN 4270
3980 : REM: CONSIDER THE FIRST OF THEM.
3990 J=FNG(1)
4000 : REM: ASK IF NODE J IS TO BE KEPT.
4010 PRINT:PRINT "RETAIN <R>"RIGHT$(NM$(J),LEN(NM$(J))-4)"<r>? ";:GOSUB 830
4020 : REM: IF NOT, DELETE J RECURSIVELY, THEN START OVER ON X.
4030 IF A$="N" THEN OD%(0)=OD%(0)+1:OD%(OD%(0))=J:GOSUB 3910:GOTO 3910
4040 : REM: IF YES, GET UP-GRAPH FROM J WITH REPEAT.
4043 BR$(0)="  ":LL=USR(J+UP+256)
4044 : REM: DELETE EDGE FROM X TO J.
4050 A$=CHR$(X+64)+CHR$(J+64)
4060 K=LEN(L$(0))
4070 FOR I=1 TO K STEP 2
4080 B$=MID$(L$(0),I,2)
4090 IF A$<>B$ THEN NEXT I:GOTO 4210
4100 A$="":IF I>1 THEN A$=LEFT$(L$(0),I-1)
4110 B$="":IF I<K-1 THEN B$=RIGHT$(L$(0),K-I-1)
4120 L$(0)=A$+B$
4130 I=K:NEXT I
4140 : REM: AND USE UP-LINKS OF X TO DEFINE UP-LINKS OF J.
```

Tree Analysis *(Continued)*

```
2840 PRINT W1$RIGHT$(NM$(IP),LEN(NM$(IP))-I)"."
2850 : REM: PRINT IMMEDIATE SUBORDINATES OF NODE IP.
2860 FOR I=1 TO LEN(L$(0)) STEP 2
2870 IF CHR$(IP+64)=MID$(L$(0),I,1) THEN PRINT " "NM$(FNL(I+1))
2880 NEXT I
2890 : REM: PRINT ELICITATION QUESTION AND GET ANSWER.
2900 PRINT:PRINT " WHAT IS A MORE GENERAL TERM FOR
2910 PRINT " SOME OF THESE (<R>$<r>: SKIP)":GOSUB 670
2920 : REM: PERMIT SKIP.
2930 IF WD$="$" THEN RETURN
2940 : REM: ACCEPT PRINT-GRID COMMAND.
2950 IF LEFT$(WD$,1)="#"THEN GOSUB 3160:GOTO 2750
2960 : REM: ADD AN ARTICLE TO NEW WORD, ENTER IN NM$, AND LINK IT INTO TREE.
2970 GOSUB 1340:H=IL:GOSUB 3060:IF H=IL THEN PRINT"<R>ALREADY EXISTS":GOTO2750
2980 GOSUB 1880
2990 : REM: SHOW GRID.
3000 GOSUB 3160
3010 RETURN
3020 :
3030 : REM: SUBROUTINE TO ADD A WORD TO THE LIST OF WORDS, NM$.
3040 : REM: IF WORD ALREADY IS IN NM$
3050 : REM: THEN CLOSE LOOP AND RETURN.
3060 FOR JJ=2 TO IL
3070 IF WD$=LEFT$(NM$(JJ),LEN(WD$)) THEN ZZ=JJ:JJ=IL:NEXT JJ:RETURN
3080 NEXT JJ
3090 : REM: OTHERWISE ADD THE WORD TO NM$.
3100 ZZ=JJ:IL=JJ:NM$(JJ)=WD$
3110 RETURN
3120 :
3130 : REM: SUBROUTINE TO DISPLAY RESULTS ON GRID.
3140 : REM: ANSWER '#H' GIVES HARD COPY.
3150 : REM: SET DEVICE # FOR SCREEN OR PRINTER.
3160 DV=3:IF MID$(WD$,2,1)="H" THEN DV=4
3170 : REM: SCREEN OUTPUT LIMITED TO 22 NODES.
3180 IF DV=3 AND MS>0 AND IL>22 THEN RETURN
3190 : REM: IF ON THE 23RD NODE, PRINT ONE-TIME MESSAGE.
3200 MS$="TOO MANY ENTRIES TO DISPLAY GRID ON      SCREEN.
3210 IF MS=0 AND IL>23 THEN PRINT MS$:MS=1:RETURN
3220 : REM: SET GRAPHIC CHARACTERS FOR SCREEN OR PRINTER.
3230 LN$=CHR$(219):MK$=CHR$(209):DG$=CHR$(95)
3240 IF DV=4 THEN LN$="+":MK$="X":DG$="<"
3250 : REM: OPEN PRINTING FILE.
3260 OPEN 1,DV
3270 : REM: CLEAR SCREEN AND PRINT HEADER.
3280 PRINT "<s>";
3290 PRINT#1,MID$(NM$(1),2):PRINT#1
3300 : REM: SEARCH DOWN TO GET ORDERED LIST OF NODES (GR$) WITHOUT REPEATS.
3310 BR$(0)="  ":LL=USR(1+DOWN)
3320 : REM: FOR EACH WORD IN TOPOLOGICAL ORDER,
3330 FOR IO=2 TO IL
3340 : REM: REMOVE ARTICLE,
3350 W1$=MID$(NM$(FNG(IO)),5)
3360 : REM: AND TRUNCATE IF TOO LONG.
3370 W1$=MID$(W1$,1,13)
3380 : REM: THEN PRINT ON NEXT ROW, FOLLOWED BY SPACES TO EDGE OF GRID.
3390 PRINT#1:PRINT#1,LEFT$(W1$+SP$,15);
3400 : REM: GENERATE GRID CHARACTERS.
3410 FOR JO=1 TO IO-1:OD%(JO)=ASC(LN$):NEXT JO:OD%(IO)=ASC(DG$)
3420 : REM: SET SPACING INDEX.
3430 JL=1
3440 : REM: GO THROUGH RECEIVER NODES IN EDGE-LIST TO FIND THIS NODE.
3450 FOR JO=2 TO LEN(L$(0)) STEP 2
3460 IF MID$(L$(0),JO,1)<>MID$(GR$(0),IO,1) THEN NEXT JO:GOTO 3550
3470 : REM: NOW FIND SENDER-NODE'S TOPOLOGICAL ORDER,
3480 FOR JJ=1 TO LL
3490 IF MID$(L$(0),JO-1,1)<>MID$(GR$(0),JJ,1) THEN NEXT JJ
3500 : REM: INSERT A LINK MARK IN THE GRID LINE,
3510 OD%(JJ)=ASC(MK$)
3520 : REM: AND CLOSE LOOPS.
```

Tree Analysis *(Continued)*

```
3530 JJ=LL:NEXT JJ,JO
3540 : REM: WHEN LINE IS FINISHED, PRINT IT.
3550 FOR JO=1 TO IO:PRINT#1,CHR$(OD%(JO));:NEXT JO
3560 : REM: CONTINUE THROUGH ALL LINES.
3570 NEXT IO
3580 : REM: IF DOING HARD COPY, PRINT OUT NAMES.
3590 IF DV=4 THEN PRINT#1:FOR IO=1 TO IL:PRINT#1,NM$(FNG(IO)):NEXT IO
3600 : REM: CLOSE THE PRINT FILE, AND PRINT A MESSAGE.
3610 CLOSE 1
3620 PRINT"<S>":PRINT" <R>TYPE RETURN":PRINT" <R>TO CONTINUE
3630 : REM: WAIT FOR USER RESPONSE.
3640 GET A$:IF A$="" THEN 3640
3650 : REM: CLEAR SCREEN AND RETURN TO ELICITATION.
3660 PRINT "<s>";:RETURN
3670 :
3680 : REM: SUBROUTINE TO REVISE A NODE.
3690 : REM: REMOVE ! FROM FRONT OF WORD, ADD ARTICLE, AND FIND IN NM$ LIST.
3700 WD$=RIGHT$(WD$,LEN(WD$)-1):K=IL:GOSUB 1340:GOSUB 3060
3710 : REM: ABORT IF WORD WAS NOT IN NM$ LIST.
3720 IF K<IL THEN NM$(IL)="":IL=IL-1:PRINT "<R>NOT FOUND":RETURN
3725 PRINT "<s>FOUND":PRINT "<R>"NM$(ZZ)
3730 : REM: PRINT MENU.
3740 PRINT "YOU CAN <R>C<r>HANGE THE NAME,
3750 PRINT "       OR <R>D<r>ELETE THE WORD ENTIRELY,
3755 PRINT "       OR <R>E<r>XIT.
3760 PRINT "<Q>ENTER <R>C<r>, <R>D<r>, OR <R>E<r>.
3770 GET A$:IF A$="" OR (A$<"C" AND A$>"E") THEN 3770
3780 : REM: GO TO SUBROUTINE FOR DELETION.
3785 IF A$="E" THEN RETURN
3790 IF A$="D" THEN GOSUB 3890:RETURN
3800 : REM: DO NAME CHANGE.
3810 PRINT "<Q>NEW NAME FOR <R>"RIGHT$(NM$(ZZ),LEN(NM$(ZZ))-3)
3820 PRINT "<r>";:GOSUB 670
3830 : REM: ADD ARTICLE AND REPLACE.
3840 GOSUB 1340:NM$(ZZ)=WD$
3850 RETURN
3860 :
3870 : REM: SUBROUTINE TO DELETE A NODE.
3880 : REM: PUT NODE TO BE REMOVED ON 'STACK'.
3890 OD%(0)=1:OD%(1)=ZZ
3900 : REM: GET LAST NODE, X, ON STACK.
3910 X=OD%(OD%(0))
3920 : REM: DEFINE ITS DOWN-GRAPH,
3930 BR$(0)="  ":LQ=USR(X+DOWN)
3940 : REM: WITHOUT THE X NODE.
3950 GR$(0)=RIGHT$(GR$(0),LQ-1)
3960 : REM: SAVE GR$.
3970 GG$=LEFT$(GR$(0),LEN(GR$(0)))
3972 : REM: WHEN X HAS SUBORDINATES,
3974 IF LEN(GR$(0))=0 THEN 4270
3980 : REM: CONSIDER THE FIRST OF THEM.
3990 J=FNG(1)
4000 : REM: ASK IF NODE J IS TO BE KEPT.
4010 PRINT:PRINT "RETAIN <R>"RIGHT$(NM$(J),LEN(NM$(J))-4)"<r>? ";:GOSUB 830
4020 : REM: IF NOT, DELETE J RECURSIVELY, THEN START OVER ON X.
4030 IF A$="N" THEN OD%(0)=OD%(0)+1:OD%(OD%(0))=J:GOSUB 3910:GOTO 3910
4040 : REM: IF YES, GET UP-GRAPH FROM J WITH REPEAT.
4043 BR$(0)="  ":LL=USR(J+UP+256)
4044 : REM: DELETE EDGE FROM X TO J.
4050 A$=CHR$(X+64)+CHR$(J+64)
4060 K=LEN(L$(0))
4070 FOR I=1 TO K STEP 2
4080 B$=MID$(L$(0),I,2)
4090 IF A$<>B$ THEN NEXT I:GOTO 4210
4100 A$="":IF I>1 THEN A$=LEFT$(L$(0),I-1)
4110 B$="":IF I<K-1 THEN B$=RIGHT$(L$(0),K-I-1)
4120 L$(0)=A$+B$
4130 I=K:NEXT I
4140 : REM: AND USE UP-LINKS OF X TO DEFINE UP-LINKS OF J.
```

Tree Analysis *(continued)*

```
4150 K=K-2
4160 FOR I=2 TO K STEP 2
4170 IF MID$(L$(Ø),I,1)<>CHR$(X+64) THEN NEXT I:GOTO 421Ø
4171 : REM: (DON'T LINK IF PATH ALREADY EXISTS.)
4172 KK=Ø
4173 FOR II=1 TO LL
4174 IF MID$(L$(Ø),I-1,1)=MID$(GR$(Ø),II,1) THEN KK=KK+1
4175 NEXT II
4176 IF KK>1 THEN 419Ø
418Ø L$(Ø)=L$(Ø)+MID$(L$(Ø),I-1,1)+CHR$(J+64)
419Ø NEXT I
42ØØ : REM: TAKE DOWN-GRAPH OF J OUT OF GR$.
421Ø BR$(Ø)=LEFT$(GG$,LEN(GG$))
422Ø LQ=USR(J+DOWN)
423Ø GR$(Ø)=LEFT$(BR$(Ø),LEN(BR$(Ø)))
424Ø : REM: AND CONTINUE WITH OTHER SUBORDINATES OF X.
425Ø GOTO 397Ø
426Ø : REM: WHEN X HAS NO SUBORDINATES, REMOVE ITS UP LINKS FROM L$.
427Ø K=LEN(L$(Ø))
428Ø FOR I=2 TO K STEP 2
429Ø IF MID$(L$(Ø),I,1)<>CHR$(X+64) THEN NEXT I:GOTO 435Ø
43ØØ A$="":IF I>2 THEN A$=LEFT$(L$(Ø),I-2)
431Ø B$="":IF I<K THEN B$=RIGHT$(L$(Ø),K-I)
432Ø L$(Ø)=A$+B$
433Ø I=K:NEXT I:GOTO 427Ø
434Ø : REM: THEN REMOVE THE WORD FROM THE NM$ LIST.
435Ø FOR I=X TO 64
436Ø NM$(I)=NM$(I+1)
437Ø NEXT I:NM$(65)=""
438Ø IL=IL-1
439Ø : REM: LOOK THROUGH L$, AND
44ØØ FOR I=1 TO K
441Ø : REM: IF AN INDEX VALUE IS ABOVE X,
442Ø IF FNL(I)<X THEN NEXT I: GOTO 449Ø
443Ø : REM: DECREMENT IT TO ADJUST FOR THE DELETED NODE.
444Ø A$="":IF I>1 THEN A$=LEFT$(L$(Ø),I-1)
445Ø B$="":IF I<K THEN B$=RIGHT$(L$(Ø),K-I)
446Ø L$(Ø)=A$+CHR$(ASC(MID$(L$(Ø),I,1))-1)+B$
447Ø NEXT I
448Ø : REM: SIMILARLY, ADJUST INDEX VALUES IN THE 'STACK'.
449Ø FOR I=1 TO OD%(Ø)
45ØØ IF OD%(I)>X THEN OD%(I)=OD%(I)-1
451Ø NEXT I
452Ø : REM: REMOVE THE X NODE FROM THE 'STACK'.
453Ø OD%(Ø)=OD%(Ø)-1
454Ø : REM: WHEREUPON REMOVAL OF NODE X IS FINISHED.
455Ø : REM: RESET SCANNING INDEX.
456Ø IP=2
457Ø RETURN
458Ø :
459Ø : REM: SUBROUTINE TO SAVE DATA,
46ØØ : REM: CALLED BY ENTERING .LABEL OR \LABEL AS A WORD.
461Ø : REM: MAKE THE FILE NAME FROM NM$(1) AND THE LABEL.
462Ø K=9-TA:IF TA=3 THEN K=4
463Ø FI$=MID$(WD$,2,7)+"."+MID$(NM$(1),K,8)
464Ø : REM: SET UP TAPE SAVE,
465Ø DV=1:CH=1
466Ø : REM: BUT CHANGE TO DISK SAVE WHEN SLASH PRECEDES LABEL.
467Ø IF LEFT$(WD$,1)="\" THEN DV=8:CH=5:FI$="Ø:"+FI$+",S,W"
468Ø : REM: OPEN THE WRITE FILE.
469Ø OPEN 1,DV,CH,FI$
47ØØ : REM: SAVE THE TYPE OF ANALYSIS (FOLLOWED BY 'RETURN'),
471Ø PRINT#1,TA;CHR$(13);
472Ø : REM: THE NUMBER OF NODES,
473Ø PRINT#1,IL;CHR$(13);
474Ø : REM: THE NODE NAMES,
475Ø FOR I=1 TO IL:PRINT#1,CHR$(34);NM$(I);CHR$(34);CHR$(13);:NEXT I
476Ø : REM: THE NUMBER OF EDGES,
477Ø PRINT#1,LEN(L$(Ø))/2;CHR$(13);
478Ø : REM: AND THE LIST OF EDGES.
479Ø FOR I=1 TO LEN(L$(Ø)) STEP 2:PRINT#1,MID$(L$(Ø),I,2);CHR$(13);:NEXT I
48ØØ : REM: FINISH.
481Ø CLOSE 1:RETURN
482Ø :
483Ø : REM: SUBROUTINE TO READ DATA,
484Ø : REM: CALLED BY ENTERING T OR D AT FIRST QUESTION.
485Ø : REM: GET THE FILE NAME.
486Ø INPUT "<R>FILE NAME<r>"; FI$
487Ø : REM: SET UP TAPE READ,
488Ø DV=1:CH=Ø
489Ø : REM: BUT CHANGE TO DISK READ WHEN A$=D.
49ØØ IF WD$="D" THEN DV=8:CH=5:FI$="Ø:"+FI$+",S,R"
491Ø : REM: OPEN THE READ FILE.
492Ø OPEN 1,DV,CH,FI$
493Ø : REM: GET AND DISPLAY THE TYPE OF ANALYSIS.
494Ø INPUT#1,TA:PRINT "<Q><Q>TYPE OF ANALYSIS =";TA
495Ø : REM: THE NUMBER OF NODES,
496Ø INPUT#1,IL:PRINT IL;"NODES ARE:"
497Ø : REM: THE NODE NAMES,
498Ø FOR I=1 TO IL:INPUT#1,NM$(I):PRINT NM$(I),:NEXT I:PRINT
499Ø : REM: THE NUMBER OF EDGES,
5ØØØ INPUT#1,LL:PRINT LL;"EDGES:"
5Ø1Ø : REM: AND THE LIST OF EDGES.
5Ø2Ø FOR I=1 TO LL:INPUT#1,A$:PRINT A$,:L$(Ø)=L$(Ø)+A$:NEXT I:PRINT
5Ø3Ø : REM: FINISH.
5Ø4Ø CLOSE 1:ON TA GOSUB 98Ø,113Ø,128Ø
5Ø5Ø RETURN
5Ø6Ø :
5Ø7Ø : REM: SUBROUTINE TO DISPLAY COMMONALITIES.
5Ø8Ø : REM: GET GENERAL DOWN-GRAPH.
5Ø9Ø BR$(Ø)=" ":LL=USR(1+DOWN)
51ØØ : REM: SHOW NODES IN TYPOLOGICAL ORDER WITH REFERENCE NUMBERS.
511Ø FOR I=2 TO IL:PRINT "<R>"FNG(I)"<r>"MID$(NM$(FNG(I))+SP$,5,15),:NEXT I
512Ø : REM: PROMPT.
513Ø GG$="":K=1:OD%(Ø)=Ø:PRINT:PRINT"COMMONALITIES FOR WHICH ENTRIES"
514Ø INPUT "(ENTER Ø TO END)"; OD%(K):IF OD%(K)=Ø THEN 521Ø
515Ø OD%(Ø)=OD%(Ø)+1
516Ø : REM: CONCATENATE THE UP-GRAPHS.
517Ø LL=USR(OD%(K)+UP)
518Ø GG$=GG$+GR$(Ø)
519Ø K=K+1:GOTO 514Ø
52ØØ : REM: CREATE A DUMMY SUBORDINATE OF FOCUS NODES.
521Ø IL=IL+1:NM$(IL)="QQ"
522Ø FOR I=1 TO OD%(Ø):L$(Ø)=L$(Ø)+CHR$(OD%(I)+64)+CHR$(IL+64):NEXT I
523Ø : REM: ELIMINATE IRRELEVANT NODES FROM GG$ BY TAKING UP-GRAPH FROM QQ.
524Ø FOR I=1 TO OD%(Ø)-1
525Ø BR$(Ø)=LEFT$(GG$+" ",LEN(GG$)):LL=USR(IL+UP)
526Ø GG$=LEFT$(BR$(Ø),LEN(BR$(Ø)))
527Ø : REM: REPEAT IF MORE THAN TWO FOCUS NODES.
528Ø NEXT I
529Ø : REM: LOWEST NODE OF GG$ IS A COMMONALITY.
53ØØ K=ASC(RIGHT$(GG$,1))-64
531Ø PRINT:PRINT "COMMONALITY="NM$(K)
532Ø : REM: ELIMINATE ITS UP-GRAPH.
533Ø BR$(Ø)=LEFT$(GG$+" ",LEN(GG$)):LL=USR(K+UP)
534Ø GG$=LEFT$(BR$(Ø),LEN(BR$(Ø)))
535Ø : REM: FIND SUBORDINATES LEADING TO EACH FOCUS NODE.
536Ø FOR J=1 TO OD%(Ø)
537Ø : REM: PRINT RELEVANT FOCAL NODE AND ITS SUBCOMMONALITIES.
538Ø PRINT "FOR"NM$(OD%(J))
539Ø : REM: GET UP-GRAPH FROM FOCUS NODE.
54ØØ BR$(Ø)=" ":LL=USR(OD%(J)+UP):A$=LEFT$(GR$(Ø),LEN(GR$(Ø)))
541Ø : REM: GET DOWN-GRAPH FROM COMMON NODE.
542Ø LL=USR(K+DOWN):B$=LEFT$(GR$(Ø),LEN(GR$(Ø)))
543Ø : REM: SUBTRACT DOWN-GRAPH FROM SUBORDINATE OF COMMON NODE:
```

Tree Analysis *(continued)*

```
5440 IF LEN(B$)=1 THEN NEXT J:GOTO 5550
5450 : REM: FROM A$;
5460 BR$(0)=LEFT$(A$+" ",LEN(A$)):LQ=ASC(MID$(B$,2,1))-64:LL=USR(LQ+DOWN)
5470 I=ASC(RIGHT$(BR$(0),1))-64
5480 : REM: THEN FROM B$.
5490 BR$(0)=LEFT$(B$+" ",LEN(B$)):LL=USR(LQ+DOWN)
5500 B$=LEFT$(BR$(0),LEN(BR$(0)))
5510 : REM: FOUND A LINK IF FOCAL NODE WAS DELETED FROM A$.
5520 IF I<>OD%(J) THEN PRINT NM$(LQ)
5530 GOTO 5440
5540 : REM:  IF THERE ARE MORE COMMON NODES, THEN CONSIDER THEM.
5550 IF LEN(GG$)>0 THEN 5300
5560 : REM: OTHERWISE ELIMINATE THE DUMMY NODE
5570 WD$=NM$(IL):ZZ=IL:GOSUB 3890
5580 : REM: THEN WAIT FOR USER RESPONSE AND RETURN.
5590 PRINT "<R>PRESS RETURN TO CONTINUE.
5600 GOTO 3640
5610 :
5620 "<S>" : CURSOR HOME
5630 "<s>" : CLEAR SCREEN
5640 "<Q>" : CURSOR DOWN
5650 "<q>" : CURSOR UP
5660 "<]>" : CURSOR RIGHT
5670 "<}>" : CURSOR LEFT
5680 "<R>" : REVERSE ON
5690 "<r>" : REVERSE OFF
5700 "[&]" : SHADED SQUARE
READY.
```

Listing 3: **Trees**

```
              0010          ;              ***************************
              0020          ;              *                         *
              0030          ;              *          TREES          *
              0040          ;              *                         *
              0050          ;              ***************************
              0060          ;
              0090 START    .BA 15784     ;PLACE WHERE CODE BEGINS WHEN USED;
              0095          ;             ;USE 32168 FOR 32K [7592 FOR 8K].
              0100          .MC $7DA8     ;PLACE WHERE CODE ASSEMBLES.
              0110          .OS  ;        ;CREATE CODE.
              0120          ;
              0130          ;              BASIC SUBROUTINES
              0140 INTFLP   .DE $D26D     ;INTEGER TO FLOATING POINT [1.0 - $D278, 4.0 - $C4BC]
              0150 FLPINT   .DE $D09A     ;FLOATING POINT TO INTEGER [1.0 - $D0A7, 4.0 - $C2EA]
              0160 OUT      .DE $C369     ;ERROR ROUTINE [1.0 - $C36C, 4.0 - $B3E0]
              0170          ;
              0180          ;              ZERO-PAGE USAGE
              0190 PTR      .DE 44        ;BEGINNING OF ARRAYS [126]
              0200 LIM      .DE 46        ;END OF ARRAYS [128]
              0210 RAM      .DE 52        ;LOCATION FOR HIGHEST RAM ADDRESS [134]
              0220 NAME     .DE 66        ;VARIABLE NAME [148]
              0230 L\POINTER .DE 84       ;WORK SPACE [166]
              0240 FACC     .DE 94        ;FLOATING POINT ACCUMULATOR [176]
              0250          ;
              0260          ;              TAPE BUFFER USAGE
              0270 GRAPH    .DE 634
              0280 TLIST    .DE 1017-MAX
              0290 COUNT    .DE TLIST-MAX
              0300          ;
              0310 MAX      .DE 70        ;MAXIMUM # OF NODES IN GRAPH
              0320          ;
              0350          ;              ***************************
              0360          ;              BRING IN ASSEMBLED ROUTINES
3DA8- A9 B9   0370 INVOKE   LDA #L,TREE   ;SET USR POINTER
3DAA- 85 01   0380          STA *1
```

Trees *(Continued)*

```
3DAC- A9 3D   0390          LDA #H,TREE
3DAE- 85 02   0400          STA *2
              0410          ;             ;PROTECT SUBS FROM BASIC
3DB0- A9 A7   0420          LDA #L,INVOKE-1
3DB2- 85 34   0430          STA *RAM
3DB4- A9 3D   0440          LDA #H,INVOKE-1
3DB6- 85 35   0450          STA *RAM+1
3DB8- 60      0460          RTS
              0470          ;
              0480          ;
              0490          ;              ***************************
              0500          ;              EXECUTIVE PROGRAM
3DB9- 20 C9 3D 0510 TREE    JSR PRELIM    ;SET PARAMETERS FOR PROCEDURES
3DBC- 20 7A 3E 0520         JSR TOPOL     ;GET TOPOLOGICAL ORDERING OF NODES
3DBF- 20 E4 3E 0530         JSR TRACE     ;GET LIST OF REACHABLE NODES -- GR$
3DC2- 20 5D 3F 0540         JSR REORDER   ;PUT GR$ IN TOPOLOGICAL ORDER
3DC5- 20 96 3F 0550         JSR DELETE    ;REMOVE GR$ ENTRIES FROM BR$
3DC8- 60      0560          RTS
              0570          ;
              0580          ;              ***************************
              0590          ;              PRELIMINARY PROCEDURES
              0600          ;             ;COUNT NUMBER OF NODES IN TREE.
3DC9- A9 4E   0610 PRELIM   LDA #'N'      ;FIND THE NM$ ARRAY.
3DCB- 85 42   0620          STA *NAME
3DCD- A9 4D   0630          LDA #'M'
3DCF- 09 80   0640          ORA #128
3DD1- 85 43   0650          STA *NAME+1
3DD3- 20 1D 3E 0660         JSR FIND
3DD6- A0 00   0670          LDY #0        ;GET DIM OF NM$ ARRAY.
3DD8- B1 54   0680          LDA (L\POINTER),Y
3DDA- C9 48   0690          CMP #MAX+2
3DDC- 90 05   0700          BCC NM\SIZE
3DDE- A2 46   0710          LDX #$46      ;IF DIM>MAX,
3DE0- 4C 69 C3 0720         JMP OUT       ;ABORT WITH OVERFLOW ERROR.
3DE3- AA      0730 NM\SIZE  TAX  ;        ;SET X= DIM OF NM$ ARRAY.
3DE4- A9 FF   0740          LDA #255      ;INITIALIZE COUNTER TO GIVE
3DE6- 8D E0 3E 0750         STA IL+1      ;NUMBER OF NODES -1.
3DE9- A0 FE   0760          LDY #254      ;TO START AT 1ST STRING IN NM$ ARRAY,
3DEB- C8      0770 DOMORE   INY  ;        ;MOVE UP 3 BYTES (NOT FOR OS 4.0)
3DEC- C8      0780          INY
3DED- C8      0790          INY
              0800          ;             ;AND CHECK STRING LENGTH.
3DEE- B1 54   0810          LDA (L\POINTER),Y
3DF0- F0 03   0820          BEQ DIM       ;WHEN NONZERO
3DF2- EE E0 3E 0830         INC IL+1      ;INCREMENT THE COUNT.
3DF5- CA      0840 DIM      DEX
3DF6- D0 F3   0850          BNE DOMORE    ;CONTINUE IF MORE IN NM$.
              0860          ;
              0870          ;              FIND THE LIST OF EDGES--L$ ARRAY
3DF8- A9 4C   0880          LDA #'L'
3DFA- 85 42   0890          STA *NAME
3DFC- A9 80   0900          LDA #128
3DFE- 85 43   0910          STA *NAME+1
3E00- 20 1D 3E 0920         JSR FIND      ;GETS L$ POINTER AND LENGTH.
              0930          ;
              0940          ;             ;GET PARAMETERS PASSED BY USR.
3E03- 20 9A D0 0950         JSR FLPINT    ;CONVERT # IN FACC TO 2 BYTES.
3E06- A5 62   0960          LDA *FACC+4   ;GET START POINT FOR TREE SEARCH AND
3E08- 18      0970          CLC
3E09- 69 40   0980          ADC #64       ;CONVERT TO CHARACTER.
3E0B- 8D 01 3F 0990         STA SP+1
3E0E- A9 01   1000          LDA #1        ;ALLOW REPEATS IN TRACING GRAPH?
3E10- 25 61   1010          AND *FACC+3   ;(1=YES; 0=NO.)
3E12- 8D 12 3F 1020         STA RE+1
3E15- A9 02   1030          LDA #2        ;TRACE UP OR DOWN?
3E17- 25 61   1040          AND *FACC+3   ;(2=DOWN; 0=UP.)
```

Trees *(Continued)*

```
3E19- 8D F2 3E  1050              STA DI+1
3E1C- 60        1060              RTS
                1070              ;
                1080              ;            ***************************
                1090              ;            FIND ARRAY IN BASIC LIST
3E1D- A5 2C     1100 FIND         LDA *PTR    ;SAVE TABLE POINTER.
3E1F- 48        1110              PHA
3E20- A5 2D     1120              LDA *PTR+1
3E22- 48        1130              PHA
3E23- A0 00     1140 CHECK        LDY #0      ;COMPARE NAME IN ARRAY TABLE
3E25- B1 2C     1150              LDA (PTR),Y
3E27- C5 42     1160              CMP *NAME   ;WITH CRITERION NAME.
3E29- D0 08     1170              BNE AGAIN2
3E2B- C8        1180              INY
3E2C- B1 2C     1190              LDA (PTR),Y
3E2E- C5 43     1200              CMP *NAME+1
3E30- F0 23     1210              BEQ POINT   ;IF NO MATCH, THEN
3E32- 88        1220              DEY
3E33- C8        1230 AGAIN2       INY  ;      ;FIND LENGTH OF PRESENT ARRAY
3E34- C8        1240              INY
3E35- C8        1250              INY
3E36- B1 2C     1260              LDA (PTR),Y
3E38- 48        1270              PHA
3E39- 88        1280              DEY
3E3A- B1 2C     1290              LDA (PTR),Y ;AND ADD TO ARRAY POINTER
3E3C- 18        1300              CLC  ;      ;TO GET POSITION OF NEXT ARRAY.
3E3D- 65 2C     1310              ADC *PTR
3E3F- 85 2C     1320              STA *PTR
3E41- 68        1330              PLA
3E42- 65 2D     1340              ADC *PTR+1
3E44- 85 2D     1350              STA *PTR+1
3E46- C5 2F     1360              CMP *LIM+1  ;IF NOT END OF ARRAYS
3E48- D0 D9     1370              BNE CHECK   ;THEN CONTINUE SEARCHING.
3E4A- A5 2E     1380              LDA *LIM
3E4C- C5 2C     1390              CMP *PTR
3E4E- D0 D3     1400              BNE CHECK
3E50- A2 81     1410              LDX #$81    ;IF NAME NOT FOUND
3E52- 4C 69 C3  1420              JMP OUT     ;ABORT WITH ARRAY ERROR.
3E55- A0 07     1430 POINT        LDY #7      ;FOR THE MATCHING ARRAY,
3E57- B1 2C     1440              LDA (PTR),Y ;GET LENGTH OF FIRST STRING
3E59- 8D 8A 3E  1450              STA LL+1    ;AND STORE WHERE NEEDED.
3E5C- C8        1460              INY  ;      ;THEN GET POINTER TO 1ST STRING
3E5D- B1 2C     1470              LDA (PTR),Y
3E5F- 85 42     1480              STA *NAME   ;AND STORE IN THE NAME CELLS.
3E61- C8        1490              INY
3E62- B1 2C     1500              LDA (PTR),Y
3E64- 85 43     1510              STA *NAME+1
3E66- 18        1520              CLC  ;      ;STORE POSITION OF ARRAY LENGTH.
3E67- A9 06     1530              LDA #6
3E69- 65 2C     1540              ADC *PTR
3E6B- 85 54     1550              STA *L\POINTER
3E6D- A9 00     1560              LDA #0
3E6F- 65 2D     1570              ADC *PTR+1
3E71- 85 55     1580              STA *L\POINTER+1
3E73- 68        1590              PLA  ;      ;RESTORE POINTER FOR ARRAY TABLE.
3E74- 85 2D     1600              STA *PTR+1
3E76- 68        1610              PLA
3E77- 85 2C     1620              STA *PTR
3E79- 60        1630              RTS
                1640              ;
                1650              ;            ***************************
                1660              ;            LIST NODES IN TOPOLOGICAL ORDER
3E7A- A2 45     1670 TOPOL        LDX #MAX-1
3E7C- A9 00     1680 INITIALIZ    LDA #0
3E7E- 9D 6D 03  1690              STA COUNT,X ;PUT 0 IN COUNT CELLS.
3E81- A9 80     1700              LDA #128
3E83- 9D B3 03  1710              STA TLIST,X ;PUT NEGATIVES IN TLIST CELLS.
3E86- CA        1720              DEX
```

Trees *(Continued)*

```
3E87- 10 F3     1730              BPL INITIALIZ
                1740              ;           ;COUNT INCOMING EDGES FOR EACH NODE.
3E89- A0 00     1750 LL           LDY #0      ;LOAD L$ LENGTH-1.
                1760              ;
3E8B- 88        1770 IN\EDGES     DEY  ;      ;GET A RECEIVER IN L$ (2ND IN PAIR).
3E8C- B1 42     1780              LDA (NAME),Y
3E8E- 38        1790              SEC  ;      ;CONVERT CHARACTER TO INDEX #.
3E8F- E9 41     1800              SBC #65
3E91- AA        1810              TAX
3E92- FE 6D 03  1820              INC COUNT,X ;INCREMENT COUNT FOR THAT NODE.
3E95- 88        1830              DEY  ;      ;CONTINUE WITH NEXT EDGE PAIR IN L$.
3E96- D0 F3     1850              BNE IN\EDGES
3E98- A9 41     1860              LDA #'A'    ;START TLIST WITH TREE ORIGIN.
3E9A- 8D B3 03  1870              STA TLIST
3E9D- A2 00     1880              LDX #0
3E9F- 8E CB 3E  1890              STX GP+1    ;SET TLIST INDEX.
3EA2- BD B3 03  1900 NX\NODE      LDA TLIST,X ;GET CURRENT NODE FROM TLIST.
3EA5- 10 05     1910              BPL NO\LOOP
3EA7- A2 D5     1920              LDX #$D5    ;(LOOP! ABORT WITH COMPLEX ERROR.)
3EA9- 4C 69 C3  1930              JMP OUT
3EAC- AC 8A 3E  1940 NO\LOOP      LDY LL+1    ;LOOK FOR OUTGOING EDGE.
3EAF- 88        1950              DEY
3EB0- 88        1960              DEY
3EB1- D1 42     1970 SEARCH\ON1   CMP (NAME),Y
3EB3- D0 1E     1980              BNE SEARCH\ON2
                1990              ;           ;FOUND AN OUTGOING EDGE:
3EB5- 48        2000              PHA  ;      ;STORE SEARCH PARAMETERS,
3EB6- 8A        2010              TXA
3EB7- 48        2020              PHA
3EB8- C8        2030              INY  ;      ;DETERMINE RECEIVING NODE,
3EB9- B1 42     2040              LDA (NAME),Y
3EBB- 38        2050              SEC  ;      ;CONVERT CHARACTER TO INDEX #, AND
3EBC- E9 41     2060              SBC #65
3EBE- AA        2070              TAX
3EBF- DE 6D 03  2080              DEC COUNT,X ;DECREMENT COUNT FOR THAT NODE.
3EC2- D0 0B     2090              BNE RESTORE ;IF RECEIVER'S COUNT IS 0,
3EC4- 18        2100              CLC  ;      ;REMAKE INDEX # INTO CHARACTER,
3EC5- 69 41     2110              ADC #65
3EC7- EE CB 3E  2120              INC GP+1    ;INCREMENT TLIST INDEX,
3ECA- A2 00     2130 GP           LDX #0
3ECC- 9D B3 03  2140              STA TLIST,X ;AND STORE THE NODE ON TLIST.
3ECF- 68        2150 RESTORE      PLA  ;      ;RECOVER SEARCH PARAMETERS.
3ED0- AA        2160              TAX
3ED1- 68        2170              PLA
3ED2- 88        2180              DEY  ;      ;RESET Y FOR OUTGOING NODES.
3ED3- C0 00     2190 SEARCH\ON2   CPY #0
3ED5- F0 07     2200              BEQ NEXTT
3ED7- 88        2210              DEY
3ED8- F0 04     2220              BEQ NEXTT
3EDA- 88        2230              DEY  ;      ;CONTINUE THROUGH L$.
3EDB- 4C B1 3E  2240              JMP SEARCH\ON1
3EDE- E8        2250 NEXTT        INX  ;      ;WHEN ALL EDGES HAVE BEEN CHECKED,
3EDF- E0 00     2260 IL           CPX #0      ;REPEAT FOR NEXT ENTRY IN TLIST,
3EE1- D0 BF     2270              BNE NX\NODE ;STOPPING WHEN LAST NODE IS REACHED.
3EE3- 60        2280              RTS
                2290              ;
                2300              ;            ***************************
                2310              ;            DEPTH-FIRST SEARCH OF NODES
                2320              ;            REACHABLE FROM GIVEN NODE
3EE4- A9 FF     2330 TRACE        LDA #255    ;CREATE AN END MARKER ON STACK.
3EE6- 48        2340              PHA
3EE7- 48        2350              PHA
3EE8- AC 8A 3E  2360              LDY LL+1    ;GET LENGTH OF L$.
3EEB- 88        2370              DEY
3EEC- A9 88     2380              LDA #136    ;TO SEARCH UP, INSERT A DEY COMMAND
3EEE- 8D 0E 3F  2390              STA CMD     ;IN CODE.
```

Trees *(continued)*

```
3EF1- A9 00             LDA #0        ;BUT IF SEARCHING DOWN,
3EF3- F0 06             BEQ STORE\Y
3EF5- A9 C8             LDA #200      ;INSERT AN INY COMMAND,
3EF7- 8D 0E 3F          STA CMD
3EFA- 88                DEY  ;        ;AND ADJUST Y.
3EFB- 8C 06 3F STORE\Y  STY I+1
3EFE- A2 00             LDX #0        ;START GRAPH WITH NODE
3F00- A9 00    SP       LDA #0        ;GIVEN AS STARTING POINT.
3F02- 9D 7A 02 BUILD\GR STA GRAPH,X   ;STORE THE CURRENT NODE IN GR$.
3F05- A0 00    I        LDY #0        ;INITIALIZE Y.
                        ;             ;IF CURRENT GR$ NODE = NEXT L$ NODE
3F07- D1 42    SEARCH\L CMP (NAME),Y
3F09- D0 19             BNE NEXT\EDGE
3F0B- 48                PHA  ;        ;PUT NODE
3F0C- 98                TYA  ;        ;AND ITS L$ POSITION
3F0D- 48                PHA  ;        ;ON STACK.
3F0E- 88       CMD      DEY  ;        ;(DEY FOR UP SEARCH, INY FOR DOWN.)
                        ;             ;GET OTHER NODE IN EDGE.
3F0F- B1 42             LDA (NAME),Y
3F11- A0 00    RE       LDY #0        ;IF REPEATS ARE NOT WANTED
3F13- D0 0C             BNE PUT\IN
3F15- 48                PHA  ;        ;(TRANSFER X TO Y.)
3F16- 8A                TXA
3F17- A8                TAY
3F18- 68                PLA
3F19- D9 7A 02 UNIQUE   CMP GRAPH,Y   ;AND NODE ALREADY IS IN GR$
3F1C- F0 11             BEQ BACK\UP   ;THEN CONTINUE SEARCH.
3F1E- 88                DEY
3F1F- 10 F8             BPL UNIQUE
3F21- E8       PUT\IN   INX  ;        ;OTHERWISE MAKE IT THE
                        ;             ;NEW CURRENT NODE IN GR$.
3F22- 10 DE             BPL BUILD\GR
3F24- C0 00    NEXT\EDGE CPY #0
3F26- F0 07             BEQ BACK\UP
3F28- 88                DEY
3F29- F0 04             BEQ BACK\UP
3F2B- 88                DEY  ;        ;CONTINUE SEARCHING L$, OR
3F2C- 4C 07 3F          JMP SEARCH\L
3F2F- 68       BACK\UP  PLA  ;        ;GET LAST ENTRIES ON STACK
3F30- A8                TAY
3F31- 68                PLA
                        ;             ;AND CONTINUE THAT SEARCH.
3F32- 10 F0             BPL NEXT\EDGE
                        ;             ;SEARCH IS DONE IF STACK IS EMPTY.
3F34- E8                INX  ;        ;STORE GR$ LENGTH FOR RECALL.
3F35- 8E 4D 3F          STX GL+1
3F38- 8A                TXA  ;        ;RETURN GR$ LENGTH IN USR.
3F39- A8                TAY
3F3A- A9 00             LDA #0
3F3C- 20 6D D2          JSR INTFLP
3F3F- A9 47             LDA #'G'      ;GET POINTER TO GR$ HEADER.
3F41- 85 42             STA *NAME
3F43- A9 52             LDA #'R'
3F45- 09 80             ORA #128
3F47- 85 43             STA *NAME+1
3F49- 20 1D 3E          JSR FIND
3F4C- A9 00    GL       LDA #0        ;STORE LENGTH OF GR$
3F4E- A0 01             LDY #1
3F50- 91 54             STA (L\POINTER),Y
3F52- C8                INY  ;        ;AND POSITION OF GR$.
3F53- A9 7A             LDA #L,GRAPH
3F55- 91 54             STA (L\POINTER),Y
3F57- C8                INY
3F58- A9 02             LDA #H.GRAPH
3F5A- 91 54             STA (L\POINTER),Y
3F5C- 60                RTS
                        ;              PUT GR$ IN TOPOLOGICAL ORDER.
```

Trees *(continued)*

```
3F5D- A0 00    REORDER  LDY #0        ;START WITH 1ST ELEMENT IN TLIST
3F5F- 8C 66 3F          STY I\GR+1    ;AND GR$.
3F62- B9 B3 03 TORDER   LDA TLIST,Y   ;GET NODE FROM TLIST.
3F65- A2 00    I\GR     LDX #0        ;COMPARE TO UNORDERED NODE
3F67- DD 7A 02 FIND\NODE CMP GRAPH,X  ;IN GR$.
3F6A- D0 1D             BNE NEXT\GR   ;IF IT MATCHES
3F6C- 48                PHA  ;        ;STORE SEARCH PARAMETERS.
3F6D- 8E 88 3F          STX J\GR+1
3F70- F0 09             BEQ SHIFT
3F72- CA       DO\SHIFT DEX  ;        ;TRANSFER GR$ NODE FROM OLD POSITION
3F73- BD 7A 02          LDA GRAPH,X
3F76- E8                INX  ;        ;TO NEXT ORDERED POSITION.
3F77- 9D 7A 02          STA GRAPH,X
3F7A- CA                DEX
3F7B- EC 66 3F SHIFT    CPX I\GR+1
3F7E- D0 F2             BNE DO\SHIFT
3F80- EE 66 3F          INC I\GR+1    ;MOVE START OF UNORDERED PART OF GR$.
3F83- 68                PLA  ;        ;MOVE THE ORIGINAL CHARACTER.
3F84- 9D 7A 02          STA GRAPH,X
3F87- A2 00    J\GR     LDX #0        ;CONTINUE SEARCH TO END OF GR$.
3F89- E8       NEXT\GR  INX
3F8A- EC 4D 3F          CPX GL+1      ;(LENGTH OF GR$)
3F8D- D0 D8             BNE FIND\NODE
3F8F- C8                INY  ;        ;ADVANCE IN TLIST.
3F90- CC E0 3E          CPY IL+1      ;(# OF NODES)
3F93- D0 CD             BNE TORDER
3F95- 60                RTS
                        ;              ****************************
                        ;              REMOVE GR$ NODES FROM BR$
3F96- A9 42    DELETE   LDA #'B'      ;FIND BR$.
3F98- 85 42             STA *NAME
3F9A- A9 52             LDA #'R'
3F9C- 09 80             ORA #128
3F9E- 85 43             STA *NAME+1
3FA0- 20 1D 3E          JSR FIND
3FA3- AE 4D 3F          LDX GL+1      ;GET LENGTH OF GR$.
3FA6- CA       START\   DEX
3FA7- AC 8A 3E          LDY LL+1      ;SET CURRENT LENGTH OF BR$
3FAA- C8                INY  ;        ;AS LOOP LIMITER.
3FAB- 8C D5 3F          STY BR\END+1
3FAE- A0 00             LDY #0        ;SET INDEXES FOR REWRITING BR$.
3FB0- 8C BA 3F          STY NEW\BR+1
3FB3- 8C D3 3F          STY OLD\BR+1
3FB6- B8                CLV  ;        ;CLEAR FLAG FOR COPYING.
                        ;             ;READ CHARACTER FROM BR$
3FB7- B1 42    REWRITE  LDA (NAME),Y
3FB9- A0 00    NEW\BR   LDY #0        ;AND REWRITE IT FOR NEW VERSION.
3FBB- 91 42             STA (NAME),Y
3FBD- 70 0D             BVS COPY      ;IF CHARACTER IS NOT DELETED YET
3FBF- DD 7A 02          CMP GRAPH,X   ;AND IS SAME AS THE GR$ CHARACTER,
3FC2- D0 08             BNE COPY
3FC4- CE 8A 3E          DEC LL+1      ;DECREMENT BR$ LENGTH
3FC7- 2C E5 3E          BIT TRACE+1   ;AND SET THE FLAG FOR COPYING.
3FCA- 70 03             BVS BUMP\OLD
                        ;             ;OTHERWISE INCREMENT WRITE INDEX
3FCC- EE BA 3F COPY     INC NEW\BR+1
                        ;             ;ALWAYS INCREMENT READ INDEX,
3FCF- EE D3 3F BUMP\OLD INC OLD\BR+1
3FD2- A0 00    OLD\BR   LDY #0
3FD4- C0 00    BR\END   CPY #0        ;AND CONTINUE THROUGH BR$.
3FD6- D0 DF             BNE REWRITE
3FD8- E0 00             CPX #0        ;REPEAT FOR NEXT GR$ CHARACTER.
3FDA- D0 CA             BNE START\
3FDC- AD 8A 3E          LDA LL+1      ;STORE FINAL LENGTH OF BR$
3FDF- A0 01             LDY #1        ;IN BR$ HEADER.
3FE1- 91 54             STA (L\POINTER),Y
3FE3- 60                RTS
```

MICRO

Apple II

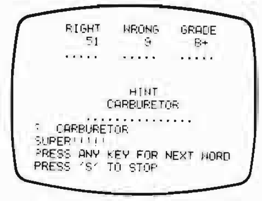
sensational software

creative computing software

CAI Programs Vol I

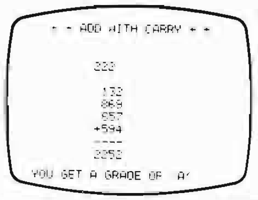
Cassette CS-4201 $11.95 — Requires 16K Apple II or Apple II Plus

U.S. Map. Identify states and their capitals.



Spelling. Study aid with your list of trouble-some words.

Math Drill. Arithmetic drill and practice with large or small display.

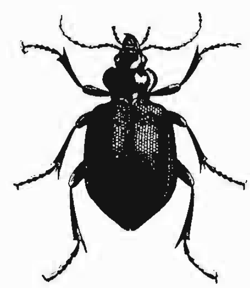


Add With Carry. Drill and practice on sums requiring numbers to be carried.

Ecology Simulations - I

Disk CS-4706. $24.95 — Requires 48K Applesoft in ROM or Apple II Plus

Sterl
STERL allows you to investigate the effectiveness of two different methods of pest control—the use of pesticides and the release of sterile males into a screw-worm fly population. The concept of a more environmentally sound approach versus traditional chemical methods is introduced. In addition, STERL demonstrates the effectiveness of an integrated approach over either alternative by itself.



Pop
The POP series of models examines three different methods of population projection, including exponential, S-shaped or logistical, and logistical with low density effects. At the same time the programs introduce the concept of successive refinement of a model, since each POP model adds more details than the previous one.

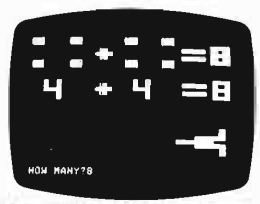
Tag
TAG simulates the tagging and recovery method that is used by scientists to estimate animal populations. You attempt to estimate the bass population in a warm-water, bass-bluegill farm pond. Tagged fish are released in the pond and samples are recovered at timed intervals. By presenting a detailed simulation of real sampling by "tagging and recovery," TAG helps you to understand this process.

Buffalo
BUFFALO simulates the yearly cycle of buffalo population growth and decline, and allows you to investigate the effects of different heard management policies. Simulations such as BUFFALO allow you to explore "what if" questions and experiment with approaches that might be disatrous in real life.

CAI Programs Vol II

Cassette CS-4202 $11.95 — Requires 16K Apple II or Apple II Plus

European Map. Identify countries and their capitals.



Meteor Math. Learn math skills by destroying menacing meteors

Music Composing Aid. Make and play your own music on the Apple. No additional hardware required. Includes a sample from Bach's Tocatta & Fugue in D minor.

Ecology Simulations - II

Disk CS-4707 $24.95 — Requires 48K Applesoft in ROM or Apple II Plus

Pollute
POLLUTE focuses on one part of the water pollution problem; the accumulation of certain waste materials in waterways and their effect on dissolved oxygen levels in the water. You can use the computer to investigate the effects of different variables such as the body of water, temperature, and the rate of dumping waste material. Various types of primary and secondary waste treatment, as well as the impact of scientific and economic decisions can be examined.

Rats
In RATS, you play the role of a Health Department official devising an effective, pratical plan to control rats. The plan may combine the use of sanitation and slow kill and quick kill poisons to eliminate a rat population. It is also possible to change the initial population size, growth rate, and whether the simulation will take place in an apartment building or an eintire city.

Malaria
With MALARIA, you are a Health Official trying to control a malaria epidemic while taking into account financial considerations in setting up a program. The budgeted use of field hospitals, drugs for the ill, three types of pesticides, and preventative medication, must be properly combined for an effective control program.

Diet
DIET is designed to explore the effect of four basic substances, protein, lipids, calories and carbohydrates, on your diet. You enter a list of the types and amounts of food eaten in a typical day, as well as your age, weight, sex, health and a physical activity factor. DIET is particularly valuable in indicating how a diet can be changed to raise or lower body weights and provide proper nutrition.

CAI Programs I and II

Disk CS-4701. $24.95
Requires 32K Integer Basic

This disk contains all 7 programs from cassettes CS-4201 and CS-4202.

Note: The ecology simulations programs are not available on cassette.

Stock & Options Analysis

Disk CS-4801. $99.95
Requires 32K Applesoft or Apple II Plus

This is a comprehensive set of four programs for the investment strategy of hedging listed options against common stocks. A complete description is in the TRS-80 section. Available August 1981.

Order Today

To order any of these software packages, send payment plus $2.00 postage and handling per order to Creative Computing, Morris Plains, NJ 07950. Attn: Cleo.
Visa, MasterCard and American Express orders may be called in toll-free.

Order today at no risk. If you are not completely satisfied, your money will be promptly and courteously refunded.

Creative Computing Software
Morris Plains, NJ 07950
Toll-free **800-631-8112**
In NJ 201-540-0445

creative computing software

PET Menu and Tape Timer

by Dale De Priest

This article describes a menu program that allows rapid access to any program on either side of a cassette tape. In addition, a tape timer is presented that supplies the fast forward times for the menu program. These two programs feature advanced cassette control and use the WAIT command extensively.

MENU and TAPE TIMER
require:
PET
cassette

Every disk operating system has a way of maintaining a directory or menu on each disk. This directory allows you to find out the names of all the programs on the disk, and allows the disk operating system to locate these programs. A workable tape operating system should also provide a directory for its tapes.

The PET operating system is capable of locating and loading programs from tape but it is slow. We need some way to index the programs so that they can be located with the fast-forward mode. Unfortunately Commodore didn't provide for an index counter on their tape drive. So, I decided to develop a program to simulate an index counter and the directory or menu operation.

Menu Program

My original goal in creating the menu program was to provide a loader that would allow you to access any portion of the tape in as much time as it would take to get to the second program on a conventional tape. Since an average program takes about 90 seconds to load, I needed a program that would load in 30 seconds, thereby allowing 60 seconds for the fast-forward time.

Listing 1

```
50 REM DOCUMENTED SAMPLE MENU PROGRAM
52 REM WRITTEN BY DALE DE PRIEST
54 :
56 :
58 REM VARIABLE USEAGE
60 REM A = INPUT NUMERIC VALUE
62 REM B = LOCATION OF INPUT BUFFER
64 REM C = CASSETTE #1 SWITCH FLAG
66 REM D = ADDRESS POINTING TO NUMBER OF CHARACTERS IN INPUT BUFFER
68 REM F = NUMBER OF PROGRAMS ON THIS ENTIRE TAPE
70 REM G = THE NUMBER OF THE FIRST PROGRAM ON THE OTHER SIDE
72 REM J = VARIABLE FOR 'FOR' LOOP
74 REM T1 = TIME AT THE BEGINNING OF THE SEARCH
76 REM A$ = INPUT VARIABLE
78 REM B$ = A MESSAGE
80 REM C$ = THE LOAD/RUN COMMAND STRING
82 REM A$(F) = NAMES OF ALL THE PROGRAMS ON THIS TAPE
84 REM A(F) = FAST FORWARD SEARCH TIMES FOR EACH PROGRAM
86 :           IN TENTHS OF SECONDS
88 :
98 REM INITIALIZE AND RANDOMIZE THE RANDOM NUMBER GENERATOR
100 F=36:G=20:A=RND(-TI):PRINT"■■MENU":DIMA$(F),A(F)
106 :
108 REM CORRECTS ADDRESSES FOR OLD AND NEW ROMS
110 B=527:C=519:D=525:IFPEEK(5E4)THEND=158:B=623:C=249
116 :
118 REM PRINT THE NAMES OF ALL THE PROGRAMS AND THE SELECTION NUMBER
120 FORJ=1TOF:IFJ=21THENPRINT"■■"
130 READA$(J):READA(J):IFJ>20THENPRINTTAB(20);
140 PRINTJ;A$(J):NEXT:IFF>20THENFORJ=FTO39:PRINT:NEXT
150 PRINT"■ENTER THE NUMBER BESIDE THE■"
156 :
158 REM THE 4 BLANKS AFTER 'WISH' ARE SHIFTED BLANKS
160 INPUT"■PROGRAM YOU WISH    ■■■■■";A$:A=VAL(A$):IFA<1ORA>FTHEN160
170 IFA(A)<>1THEN200
176 :
178 REM THIS SIMULATES THE RUN/STOP KEY
180 C$="LOAD"+CHR$(13)+"RUN"+CHR$(13):FORJ=0TO8:POKEB+J,ASC(MID$(C$,J+1))
190 NEXT:POKED,J:END
198 :
200 B$="■PLEASE PRESS STOP ON THE CASSETTE;":PRINTB$:WAIT59408,16
206 :
208 REM THE OPERATING SYSTEM AUTOMATICALLY GIVES MOTOR CONTROL TO THE USER
210 PRINT"■PRESS THE FAST FWD BUTTON":WAIT59408,16,16
216 :
218 REM THE POKE RETURNS MOTOR CONTROL BACK TO THE COMPUTER
220 PRINT"■LOOKING FOR "A$(A):POKEC,52:T1=TI
240 IFTI-T1<A(A)*6THEN240
246 :
248 REM THE COMPUTER ACTUALLY STOPS THE TAPE MOTION
250 POKE59411,61:PRINT"■"MID$(B$,9):WAIT59408,16:IFA<GTHEN180
260 PRINT"■TURN THE CASSETTE OVER":FORJ=1TO3000:NEXTJ:GOTO180
496 :
498 REM SAMPLE NAMES AND SEARCH TIMES
500 DATACOVER16,1,NAB!,52,FIRE!,133,ALIENS!,195,BONZO!,246,CATCH!,309
510 DATACOVER17,349,POLICE!,393,SPOT,460,RULER,507,LETTER,539,MERGE,594
520 DATANPACK,630,COVER18,661,DROMEDA!,691,JOUST,750,WEATHER,797
530 DATAHI RES,856,SHEEP,900
536 :
538 REM THESE PROGRAMS ARE ONTHE BACK SIDE OF THE TAPE
540 DATACOVER19,938,FROG!,896,GODZILLA!,847,MINER!,795,RAIL,746,GBOOKA,703
550 DATAGBOOKB,663,COVER20,617,MUSIC!,584,BETS,528,CHECKERS,470
560 DATACURVES,420,EQUIP,375,COVER21!,304,CAPTURE!,239,DANCE!,178,
    BOSWAIN,93
564 :
566 REM THIS LINE IS NOT USED BY THE PROGRAM BUT PROVIDES DATA
    FOR THE USER IF
568 REM   YOU WISH TO ADD AN ADDITIONAL PROGRAM TO THIS TAPE
570 DATAEND OF TAPE,948,1790
```

Listing 1 shows the result of this effort. It allows for up to 40 programs to be located on the same tape and will find any one of 30 programs in approximately 60 seconds. This program should be the first program on each of your cassettes, but it could be on a separate tape and contain the menu for several tapes. This would be desirable for tapes such as Cursor Magazine. Several interesting things can be learned from this program, so let me show you how it works.

Line 100: The statement A = RND (– TI) doesn't have anything to do with the rest of the program. It is simply an easy way to insure that the RND function is randomized for every program on the tape.

Line 110: The variable C points to the location of a flag that the PET uses to determine whether it or the user has control of the cassette motor. When you push one of the switches down, the PET turns on the motor for you. Since it believes that the operator should have control, it won't let the program stop the drive unless variable C is changed. A zero in this location means that the operator has control; any other value gives control to the PET.

Line 170: By convention the first program after the menu will contain a fast-forward value of one. This means no fast-forward is required.

Line 180-190: This is a special trick on the PET. This line stores the two commands LOAD and RUN in the PET's input buffer. You may wonder why the simpler method of

```
180 POKE B, 131:POKE D,1: END
```

is not used. This line forces the PET to respond as if the operator just hit the RUN/STOP key. The PET will load and run the next program. Unfortunately BASIC 4.0 directs this command to the disk, so you must put the command in the buffer yourself to make this program work with all versions of PET software. This technique is frequently useful when one program wants to turn control over to another program.

Line 200: This is the mysterious WAIT command. Here it is used to detect whether or not a key has been pressed. 59408 is the address whose contents will change and 16 is the decimal equivalent for the bit (bit 4) which will change to a 1 when the key is pressed. (If the bit were changing from 1 to 0, then the command would be WAIT 59408,16,16.)

Line 500-540: These lines contain the DATA statements that define the program names and the search times. The names recorded here do not need to be exactly the same as the name the program is stored under. They are only used by this program and not by the actual load routine. When setting up a tape for the first time you may not yet know the names of the programs that you are going to put on that tape. Be sure that you add enough filler to allow room to add these names later. I usually copy the menu from a different tape and then change the names as I need to. Setting F in line 100 will avoid confusion of the real contents of tape and the data that may be present simply to take up space. As each program is added I go back and update the menu program. This is easy since the long leaders prevent wiping out the second program.

This program works fine, but there is no way to easily determine what the fast-forward times should be. Now look at listing 2.

Tape Timer

This program supplies the fast-forward times for the menu program. In addition, it will provide a listing of your tape along with the load times for each program. The significant details are outlined below.

Line 120: This line finds headers and then measures the length of time between them. You should notice that although a program LOAD can distinguish between programs and data, an OPEN cannot. The last statement on the line initializes the program name variable.

Line 130: This line shows a method of getting the name of a program or data header into the program. The technique is to build a string right from the header buffer area. This is a useful line of code and shows you how to read beyond the sixteenth character of the header.

Line 150: H is the variable that is used to accumulate the time for all programs. Fast forward on a cassette tape is not linear. That is, there is no direct correlation between fast-forward times and normal play times. Although there is probably a mathematical formula that could be developed to calculate this relationship, the program uses a group of straight lines to approximate the curve. It then calculates a proportional relationship between these values. Empirically I have determined that the relationship between H and M for the first 200 seconds is .78. Therefore if a load time of 100 seconds were measured, then this program would calculate the fast-forward time to be 7.8 seconds. All fast-forward times are calculated in tenths of seconds.

Lines 160-210: These lines continue the straight line approximations to the curve. Each value has been determined empirically to stop the tape 10 to 15 seconds in front of a header. This program should be fairly accurate for cassettes up to 60 minutes long. Longer cassettes use thinner tape which will invalidate the times.

Line 230: This line assumes that the menu program was stored with the name MENU as the first four characters. This is a convention that I always follow. Checking for this name allows the program to recalibrate itself to do all timings just after the menu load. The program will also work correctly when your program menu is on a different tape since the time is then from the beginning of tape.

Line 250: This program assumes that tapes have been properly ended with an end-of-tape header. The search times may then be copied to enter into the menu program. This program uses a feature of the cassette unit that causes it to shut off automatically when it reaches the end of a tape. When this happens, the program calculates the actual length of the tape. If you were to fast-forward the tape from the other end, it would then compute the fast-forward times. These times should be entered on a MENU program located at the beginning of the back side of the tape.

Putting It All Together

Now that you have seen the two programs, let's see how we can put them to work. First, you must build your tape by putting a MENU on the front of the tape followed by several programs. The tape must end with a file named "END OF TAPE." The easiest way is to type the following command in immediate mode.

```
OPEN1,1,2,"END OF TAPE":CLOSE1
```

Then load the TAPE TIMER program and time the tape. Write the front side search times as well as the names on a piece of paper. Hit the space bar and record the reverse side search times. Load in a copy of the MENU program and modify it with the correct names and reverse search times. Set the values of F and G to the correct values. Store this new program as the first program on the back side.

Next, fast-forward the tape to the beginning of side one. Change the search times to the correct ones for the front side. Change variable G, and save this new update over the top of the old MENU program. You now have a complete tape that can be searched from either direction. Of course you could also add programs to the back of the tape.

To use this program, rewind the tape to either end. Then simply use the SHIFT/RUN keys to get the MENU program in and running. (This will not work for BASIC 4.0 users.) The MENU program will then do the rest. This way, someone who is not familiar with programming will be able to use your tapes.

Manual Updating

There will be times when you want to add a program to the end of a tape. Simply position your tape just past the current last program, either by loading or verifying it, and then save your program. Be sure to add a new "END OF TAPE" header. You could then load in the tape timer and retime your tape. Then go back and update your MENUs. There is an easier way. It takes a little bit of advance planning but it works like a charm.

When the tape timer program finds the end of tape it outputs two things about that point: the fast-forward search time and the total load time. When you create your menu program for that tape you should add these two numbers to the program. They can be added as additional data statements that are never read or they can be added in a remark. Now when you wish to add a program you will be ready.

Listing 2

```
10 REM TAPE TIMER
12 REM WRITTEN BY DALE DE PRIEST
14 :
16 :
18 REM VARIABLE USAGE
20 REM C = CASSETTE #1 SWITCH FLAG
22 REM H = ACCUMULATED TOTAL LOAD TIME
24 REM I = VARIABLE FOR 'FOR' LOOP
26 REM J = COUNTER FOR NUMBER OF PROGRAMS
28 REM M = COMPUTED SEARCH TIME
30 REM T1 = INITIAL TIME OF TIMER
32 REM A$(24) = NAMES OF EACH OF THE PROGRAMS
34 REM A(24) = LOAD TIMES FOR EACH OF THE PROGRAMS
36 REM LR = LENGTH OF RECORD IN BYTES
38 :
48 :
50 PRINT"▯PLEASE INSERT AND START THE TAPE YOU":PRINTTAB(13)"————
60 PRINT"WISH TO TIME IN TAPE # 1
66 :
68 REM DEFINE FUNCTION FOR ADDRESS PEEKING AND CORRECT FOR OLD AND NEW ROMS
70 DEFFNAD(C)=PEEK(C)+PEEK(C+1)*256:C=519:IFPEEK(5E4)THENC=249
74 :
76 REM WAIT UNTIL TAPE STOP IS DEPRESESSED THEN WAIT FOR THE PLAY BUTTON
78 REM   THEN GIVE MOTOR CONTROL BACK TO THE COMPUTER
80 WAIT59408,16:WAIT59408,16,16:POKEC,52
90 DIMA(30),A$(30):A$(0)="BEGINING OF TAPE      "
100 PRINT"▯NAME                 SEARCH TIME  LOAD TIME
106 :
108 REM ALL TAPES MUST END WITH A FILE NAMED "END OF TAPE"
110 PRINTA$(J);INT(M),:IFLEFT$(A$(J),11)="END OF TAPE"THEN250
114 REM INITIALIZE TIME & START THE TAPE MOVING
115 T1=TI:POKE59411,53
116 :
118 REM SPACE OVER THE PROGRAM FILE WITHOUT READING
120 IFTI-T1<LRGOTO120
122 :
124 REM OPEN WILL FIND BOTH PROGRAM AND DATA FILES
125 J=J+1:OPEN1,1:A(J)=(TI-T1)/60:A$(J)=""
126 :
128 REM THIS LINE FINDS THE PROGRAM NAME FROM THE TAPE HEADER
130 FORI=639TO659:A$(J)=A$(J)+CHR$(PEEK(I)):NEXT:CLOSE1:PRINTINT(A(J))
140 GOSUB150:LR=1:IFPEEK(634)=1THENLR=.0182*60*(FNAD(637)-FNAD(635))+200
145 GOTO110
146 :
148 REM THESE LINES ACTUALLY COMPUTE THE FAST FORWARD TIMES
150 H=H+A(J):M=H:IFM<200THENM=.78*M:GOTO230:REM <200
160 M=M-200:IFM<200THENM=153+.67*M:RETURN:REM 200-400
170 M=M-200:IFM<200THENM=285+.58*M:RETURN:REM 400-600
180 M=M-200:IFM<200THENM=401+.54*M:RETURN:REM 600-800
190 M=M-200:IFM<200THENM=507+.49*M:RETURN:REM 800-1000
200 M=M-200:IFM<300THENM=605+.46*M:RETURN:REM 1000-1200
210 M=M-300:M=740+.42*M:RETURN:REM >1200
230 IFLEFT$(A$(J-1),4)="MENU"THENH=0:M=1
240 RETURN
244 :
246 REM TURN THE MOTOR ON AND WAIT FOR THE AUTO SHUT OFF FEATURE
248 REM "H" NOW REPRESENTS THE TOTAL ACCUMULATED LOAD TIMES
250 T1=TI:POKE59411,53:PRINTINT(H)
260 PRINT"WAIT UNTIL THE PHYSICAL END":WAIT59408,16:H=H+(TI-T1)/60-10
270 PRINT"▯PRESS ▯ SPACE ■ FOR REVERSE SEARCH TIMES":WAIT59410,4,4
276 :
278 REM TOTAL TAPE LENGTH = LOAD TIMES + BLANK TAPE AT END + MENU LOAD TIME
280 IFLEFT$(A$(1),4)="MENU"THENH=H+A(2)
290 PRINT"▯CORRECTED TAPE LENGTH"INT(H)
296 :
298 REM THE REVERSE SIDE SEARCH TIMES ARE NOW CALCULATED
300 FORI=1TOJ:H=H-A(I):M=H:IFM<200THENM=.78*M:GOTO320
310 GOSUB160:IFM>740THENM=M+3
320 PRINTA$(I);INT(M)
330 IFI/22=INT(I/22)THENPRINT"PRESS ▯SPACE■ TO CONTINUE":WAIT59410,4,4
340 NEXT
```

First position the tape to the correct point and then type in the following line to save your program.

```
TI=TI:SAVE "NAME" :?(TI-T)/60
```

The program will save in the normal fashion, but it will also tell you how long it took. Note this time, rewind the tape and load in your menu. The menu can be updated by adding your new program name, changing variables F and G and using the search time number that you had previously entered. Then calculate a new search time for the next addition, add the save time to the accumulated time, and change your note in the program. Use this time and the listing for the tape timer to calculate your new time.

Simply pick the line you need from the choices beginning at line 150. The remarks at the end of each line should make your choice easy, and the PET calculator mode will make the calculation easy. The value of M in the equation is the length of the tape minus the left number in the remark statement. For example, suppose your time to that point is 682 and you just saved an 87-second program. Adding those two numbers gives you 769 seconds. That places us on line 180 of the listing so we must subtract 600 to get M (169). The new search value is 401 + .54*169 or 492 (49.2 seconds).

By the way, if the record and play buttons weren't already pressed when you typed in the save command, you'll have to be very careful to get the times to come out right. Type the line in but do not hit the return key. Then push the record and play keys at the same time that you push the return key.

Final Thoughts

When typing in the menu program be sure to remove all remarks. Your program will load about twice as fast that way. I hope that you get as much use out of this as I have. These programs have greatly enhanced my use of the PET Tape Operating System.

Dale De Priest is the manager of Circuit Development and Document Control at ISS Sperry Univac. He has an associate degree in Electrical Engineering Technology from Central Technical Institute in Kansas City. He can be contacted at: 611 Galen Drive, San Jose, CA 95123.

MICRO

MICRObits

(Continued from page 21)

OSI Screen Editor

Never retype a line with Edit-All. Dynamic insert/delete corrects any line on the screen. With user-definable scroll window and full cursor control and screen clear. For OS-65D systems - $19.95. Universe (by *Galaxian* author) plays like *Cobra Copter* - $14.95 tape/disk.

DMP Systems
319 Hampton Blvd.
Rochester, N.Y. 14612

OSI Superboard II, C1P

Star Chase: avoid and shoot the oncoming asteroids. 4K version - $7.95; 8K version -$9.95. *Star Fire*: Plays like *Defender*. 4K -$9.95; 8K - $12.95. *Spanish I*: Computer dictionary of Spanish words. 8K - $9.95 cassette only. Send for free catalog.

Swany's OSI Software
2652 37th West
Seattle, WA 98199

(Continued on page 105)

MICRO™
Hardware Catalog

Name: **16 Plus Board**
System: Radio Shack Color Computer TRS-80
Memory: 16K
Description: The *16 Plus Board* expands the Color Computer memory from 16K to 32K quickly, easily and inexpensively. There is no soldering. It plugs in neatly under the RF shield cover. There are no software modifications to existing software and it is completely compatible with anticipated disk systems. Plus it allows the graphics display to reside anywhere within the 32K of memory.
Price: $84.95
Includes memory board and instructions
Available:
Computerware
P.O. Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **CBM 8032P (Bi-Directional Printer)**
Memory: 1K RAM
Description: Low cost, high speed bi-directional printer. This new printer connects directly to a Commodore computer and is a 136-column printer with tractor and friction feed. The 8032P is dot-matrix, and prints 150 characters per second, prints all alphabetic and graphic characters available with a Commodore computer, as well as user-defined characters.
Price: $995.00
Available:
Commodore Business Machines, Inc.
Computer Systems Division
(through Commodore authorized dealers.)

Name: **Model 5000/D Computer Console Cabinet**
Description: Handsome, functional and portable cabinet, specially designed to house most of the components of a microcomputer system. In addition to providing ample space for the computer and disk drive units, the console includes a bookshelf for manuals, a top support for a printer or CRT terminal and a slide out stabilizer for servicing heavy units in extended position. The hinged rear door can be lifted off entirely for service access. Computer units are placed on sliding shelves, eliminating the need to mount rack slices on each chassis. An AC outlet strip with 6 grounded outlets, power switch indicator and circuit breakers are included.
Price: $795 list, OEM/Dealer discounts available
Includes Shepard casters.
Options - Drawers which fit on any shelf in place of a computer unit. Standard finish gunstock walnut laminated plastic. Other colors/woodgrains and finishes and smoked tempered glass doors available.
Available:
Micro Solutions, Inc.
6305 Arizona Ave.
Los Angeles, CA 90045

Name: **AMS8000 Megabyter**
System: 8" floppy disk drive system
Memory: 2 megabytes, expandable to 4 megabytes
Language: DOS 3.3, CP/M, Pascal
Hardware: Shugart 850 drives, SVA ZVX4 Controller Card
Description: Dual-density, double-sided drive packages support single-density and single-sided disks.
Price: $2995
Includes cable, power supply fan in Apple-matching cabinet
Available:
All Apple Dealers

Name: **DATASAVER™ AC Power Back-up Unit**
Hardware: 90 Watt Micro-UPS for Microcomputers
Description: The *DATASAVER* continuously conditions and monitors the microcomputer line voltage while charging its own battery. Upon a brownout or loss of line voltage, the *DATASAVER* automatically transfers to battery operation within ½ cycle to assure uninterrupted AC power. Hold-up time is at least 15 minutes for Apple II, TRS-80 III, and OSBORNE 1 size units, and five minutes for Apple III.
Price: $395 (120VAC@60Hz)
$424 (220VAC@50HZ)
Includes proprietary voltage surge and transient suppressor
Available:
Microcomputer System Dealers
(substantial wholesale discounts available)
or factory direct:
Cuesta Systems, Inc.
3440 Roberto Court
San Luis Obispo, CA 93401
(805) 541-4160

Name: **TNW 1000, 2000, 232-D Dual Port and TNW 103 Low Speed Modems**
Description: TNW's serial interfaces are designed to interface RS-232 devices to the Commodore PET/CBM and other IEEE-488 bus-capable computers. Each unit can share the IEEE bus with other peripheral devices and each provides a connector for daisy-chaining. TNW's 103 is a full-service, auto answer, auto dial modem.
Price: $129 TNW-1000;
$229 TNS-2000;
$389 TNS-103;
$369 TNW-232D.
Available:
TNW Corporation
3444 Hancock Street
San Diego, CA 92110
(714) 296-2115
TWX 910-335-1194

Name: **8-Bit, 8-Channel A/D System**
System: Apple
Hardware: A/D System
Description: The Applied Engineering A/D board is an 8-bit, 8-channel, memory-buffered, data acquisition system. It consists of an 8-bit successive approximation A/D converter, an 8-channel multiplexer and 8 × 8 random access memory. The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on-board memory at the end of each conversion. No A/D converter could be easer to use.
Price: $129 (Texas Residents add 5% sales tax.)
Available:
Applied Engineering
P.O. Box 470301
Dallas, Texas 75247
(214) 492-2027

Name: **Prism 80 and Prism 132 Column Printers**
Language: Full 96 ASCII characters/ additional resident sets
Description: Up to 200 CPS data speed (optional), 150 CPS proportional spacing; correspondence print quality; 9-wire staggered print head; cut sheet and pinfeed fanfold paper types; programmable features; print mode, line spacing, forms length, intercharacter spacing, margins, tabs; 10/12/16.8 CPI plus double width characters; fixed and proportional spacing; text justification; forms length control; line spacing; slew rate 7.5 IPS; parallel centronics compatible serial, RS-232 (DTR or XON/XOFF); baud rate up to 9600; graphics optional 84 × 84 DPI; cartridge ribbon. Options: cut sheet feeder, graphics, data speed, color printing, 9-wire in line head, special character set. (Above subject to change without notice.)
Price: $1995
Includes 132-column with color, sheet feeding and data speed. ($899 for 80-column without options.)
Available:
Representatives and Distributors;
over 900 retail outlets
(800) 258-1386

MICRO

Write for MICRO!

MICRO Magazine is actively seeking articles to present to our unique audience of sophisticated computerists. We welcome contributions on any aspect of 6502/6809 hardware and software. And send us your latest ready-to-key-in program!

Each issue of **MICRO** presents articles on a wide range of systems and topics. In addition, each issue includes a special feature. Our Editorial Schedule for the rest of 1982 is shown below. Please send

material for:		to:	no later than:
September	68000	Marjorie Morse	June 11
October	Education	Marjorie Morse	July 9
November	Atari & Games	Mary Ann Curtis	August 6
December	PET	Loren Wright	September 10

Standard **MICRO** categories that may be included in any issue are:

I/O • BASIC Aids • Machine Language Aids • Graphics
Games • Applications • Programming Techniques • Utilities
Programming Languages • Hardware

Anyone who is interested in writing about networking or online database searching should contact Laurence Kepple, Senior Editor, either at **MICRO** (617/256-5515) or at 71535,231 on the CompuServe network.

Special Note to all MICRO authors: please begin sending your text as well as your programs in machine-readable format. This will enable us to edit the text with the help of word processing. Continue to include a printed text version as you include a printed program listing.

MICRO™

Software Catalog

Name: **Color Editor**
System: TRS-80C
Memory: 32K
Language: Assembly Language
Description: *Color Editor* is used for program development and text processing. It allows use of the upper and lower case features of the Color Computer and can print letters or programs on a printer attached to the RS-232 port. It has change and search commands that work on one or all lines and can copy or move sentences or paragraphs anywhere in the file. Lines can be inserted, deleted, or moved. Your work can be saved and later retrieved on cassette.
Price: $29.95 includes cassette and instructions
Available:
Computerware
P.O. Box 668
Encinitas, CA 92024
(714) 436-3512

Name: **The Merger VisiBlend**
System: Apple II or Apple II Plus
Memory: 48K
Language: Applesoft in ROM
Hardware: Disk II
Description: *The Merger* is a utility that aids users of *The Data Factory* and *The Invoice Factory*. Merge data from fields in either program into those of another file. *VisiBlend* allows users of VisiCalc™ to combine the data in multiple VisiCalc™ files, merging data across files. (VisiCalc is a trademark of Personal Software, Inc.)
Price: $50 each
Available:
Micro Lab
2310 Skokie Valley Rd.
Highland Park, IL 60035

Name: **PET Library Card Maker**
System: PET
Memory: 16K
Language: BASIC
Hardware: IDS 460 printer
Description: Prints a full set of library cards on tear-off card stock. The information is typed once to the PET screen. You can preview the card, make corrections, then with one keystroke, print a set of up to 7 library cards. The program does all the formatting. A tape file can be made. For all small libraries.
Price: $80 - Canadian
Includes cassette tape
Author: J. Horemans
Available:
M&W Computer Stores
Sheridan Corporate Centre
2155 Leanne Blvd., Unit 3
Mississauga, Ontario
Canada L5K 2K8

Name: **Inventory System**
System: OS65U
Memory: 48K
Language: BASIC
Hardware: OSI C-2 or C-3 series
Description: *Inventory System* is an integrated portion of EIS General Accounting Systems. It has perpetual inventory, sales invoicing, accounts receivable, bills of materials, and interrelated purchase orders; information on availability, cost ordering of low or out-of-stock inventory items.
Price: $1,200.00
Includes three program disks and a step-by-step user's manual.
Available:
Electronic Information Systems, Inc.
P.O. Box 5893
Athens, GA 30604
(404) 353-2858

Name: **Turf Management**
System: OSI C4P MF
Language: BASIC under OSI 65D
Hardware: Disk drive, optional printer
Description: A program that provides the characteristics of eight common grass species, giving optimum growing conditions, use, techniques for establishment, lime and fertilizer requirements, and pest management and control for general insect and weed problems. Rates of seed required for establishment and amounts of chemicals needed for each required application can be calculated given the dimensions of the area. The program can be customized to suit the area of the country and availability of chemicals used for fertilizer and pest control.
Price: $100.00
Includes 5¼" disk and documentation ppd.
Modification to operate with other systems can be requested.
Author: J. Benton Jones, Jr.
Available:
Benton Laboratories, Inc.
P.O. Box 5455
Athens, GA 30604

Name: **SCORE: The Academic Assistant**
System: Apple II Plus (or Apple II with Applesoft on firmware card)
Memory: 48K
Language: Applesoft and machine
Hardware: 80-column printer; optical mark reader (Chatsworth, HEI or Scan-tron) highly recommended
Description: *SCORE* is a comprehensive set of programs which will score multiple choice tests, conduct comprehensive item analyses, maintain academic records, prepare frequency distributions, and individulazied student feedback, and much more. This package interfaces the Apple with the Chatsworth, HEI, or Scan-tron optical mark readers.
Price: $395.00
Includes program disk, back-up disk, data disk, comprehensive manual, ongoing support
Author: Bryan Hendricks, and Bob Bermant
Available:
Scientific Software Assoc., Ltd.
P.O. Box 208
Wausau, WI 54401
(715) 845-2066

Name: **Napoleon's Campaigns: 1813 & 1815**
System: Apple II
Memory: 48K
Language: Applesoft in ROM
Hardware: Disk Drive
Description: Corps-level game simulating the last campaigns of Napoleon: Leipzig and Waterloo. Displayed on 18 × 21 hex grid maps in hi-res graphics. Computer acts as corps commander.
Price: $59.95
Includes one diskette, rulebook, player aid card, two-sided map boards, 100 counters.
Available:
Strategic Simulations, Inc.
465 Fairchild Drive
Suite 108
Mountain View, CA 94043

Name: **TRS-80 Color Computer Learning Lab (26-3153)**
System: TRS-80 Color Computer
Memory: 4K, 16K, 32K
Language: Color BASIC
Hardware: Cassette
Description: A new self-instruction system that teaches user how to program in Color BASIC. Allows student to develop gradually through writing and editing longer, more complex programs. Example programs are practical and can be used for educational, family and personal purposes. The lab is divided into three sections: introduction to the computer; programming the computer; programming guides and tools to make programming easier, faster and more fun. The lessons take full advantage of color graphics and sound available from the TRS-80 Color Computer.
Price: $49.95
Includes eight program cassettes and 30-lesson text
Author: Radio Shack
Available:
Radio Shack

(Continued on page 94)

Name: **The Terrapin Logo Language**
System: Apple II or Apple II Plus
Memory: 64K, 48K Apple with 16K memory extension
Language: Terrapin Logo Language
Hardware: 1 disk drive
Description: The Logo language is the most powerful interpretive language ever devised for the Apple II. Yet, it is probably the easiest to use as well. It is designed so that young children can easily control the power of computers without having to know how to program. However, advanced programmers will enjoy the many features common to artificial intelligence research languages permitting programs of great power to be written quickly and easily. (Language is licensed by Mass. Institue of Technology.)
Price: $149.95
Includes language disk, utilities disk and documentation including tutorial and technical manual
Author: Leigh Klotz, Pat Sobalvarro, Steve Hain
Available:
Terrapin, Inc.
678 Massachusetts Ave.
Cambridge, MA 02139
(617) 492-8816

Name: **AlkemstoneTM**
System: Apple II, Apple II Plus, Apple III
Memory: 48K
Language: Machine
Hardware: DOS 3.3
Description: *Alkemstone*, is a computer adventure which offers a $7500 cash reward to the first person who can recover the missing Alkemstone. The quest for the Alkemstone will lead the player through underground paths of the lair of the original owner. There are unusual messages, fragments of words, sketches and other clues written on the walls. Some items are distributed randomly, so that one may be visible numerous times while some will only appear once in several trips. Each trip will result in a different combination of possible hints. If all bits of information are pieced together correctly, then the location of the Alkemstone will be obvious.
Price: $39.95
Includes 1 disk, 24-page booklet
Author: Level-10
Available:
Level-10, a division of Dakin5 Corp.
7475 Dakin St.
Denver, CO 80221
or local Apple dealer

Name: **Chem Lab Simulations #3 and #4**
System: Apple II
Atari 800
Memory: 48K
Language: Applesoft or Atari BASIC
Hardware: 48K Apple II with disk drive or 40K Atari 800 with disk drive
Description: High Technology Software Products, Inc., is pleased to announce the third and fourth additions to its series of chemistry laboratory simulations. *Chem Lab Simulations #3* contains four calorimentory experiments through which Hess' Law is demonstrated. *Chem Lab Simulations #4* utilizes two capillary tube experiments to illustrate principles of thermodynamics. Designed for college-level introductory chemistry courses, these simulations are also well suited for advanced high school students.
Price: $100 each
Includes program diskette, 3-ring binder with complete documentation.
Author: J.I. Gelder
Available:
High Technology Software Products, Inc.
2201 N.E. 63rd St.
P.O. Box 14665
Oklahoma City, OK 73113
or computer retailers

Name: **K-RAZY Shoot-Out**
System: Atari 400/800
Memory: 8K ROM
Language: 6502 machine
Description: Fast action game.

(Continued on page 96)

WHO ARE YOU THIS TIME?

Are you the seeker of lost treasures in an enchanted realm of magical beings? Perhaps you're an astronaut, thousands of light years from earth, searching the galaxy's rim for the fabulous treasures and advanced technologies of a long-dead civilization. Maybe you're the plunderer of ancient pyramids in a maddeningly dangerous land of crumbling ruins. and trackless desert wastes.

Imagine.

You wish to soar to other worlds, to behold wonders never beheld by mortal eyes, to dream unrestrained to the furthest limits of your imagination.

Imagine now that you can.

The ADVENTURE SERIES by Scott Adams has been reviewed by **every** major microcomputing magazine, receiving only the highest praise for its mind-puzzling challenge & refreshing originality. Tens of thousands of adults and children have matched wits with the Adventure Master. Now, it's your turn. If you crave the challenge and panorama of the exotic, touched with sparkling humor, then the Adventure Series is for you.

With the incredible Adventure Series by Scott Adams,
YOU ARE WHOMEVER YOU WANT TO BE, WHENEVER YOU WANT TO BE!

Adventure INTERNATIONAL
A DIVISION OF SCOTT ADAMS, INC.
© 1982

BOX 3435, LONGWOOD FL 32750
(305) 862-6917 (QUESTIONS)
ORDER FROM YOUR FAVORITE DEALER or CALL TOLL FREE (800) 327-7172
(ORDERS ONLY PLEASE A.I. #119)

	SINGLE TAPE	12 PACK TAPE	SINGLE DISK	TRIPLE DISK	12 PACK DISK
APPLE	$19.95 24K	$129.95† 24K	$29.95* 48K	$69.95* 48K	—
ATARI	$19.95 24K	$129.95† 24K	—	$39.95 40K	$129.95† 48K
CP/M (Z-80)	—	—	—	—	$129.95†
TRS-80 MOD 1&3	$19.95 16K	$129.95† 16K	—	$39.95 32K	$129.95† 32K
TRS-80 MOD 2	—	—	—	—	$129.95

PRICES SUBJECT TO CHANGE WITHOUT NOTICE

Adventure Hint Book (All 12)	$ 7.95
Adventure Hint Sheet (Single)	1.00
Commodore VIC (Cartridge)	39.95
T.I. 99/4 (Tape or Disk) Requires Adventure Command Module	29.95

*Scott Adams' Graphic Adventure Series Apple Disks-Full Color Hi-Res Graphics.
†12 Adventure Hint Book Incl.

SHIPPING & HANDLING ARE EXTRA

Software Catalog *(Continued)*

You're trapped in an alien fortress and you must clear your sector of alien droids and proceed to the next sector. Each sector is more challenging and demanding with faster and more aggressive droides. Available in solid state ROM pack.

Price: $49.95
Includes ROM pack and multi-color 12 page instruction booklet.

Author: K-BYTE

Available:
K-Byte
1705 Austin
Troy, Michigan 48099
or your dealer

Name: **Debug**
System: OSI C1P/MF, C4/MF
Memory: No Additional
Language: Machine

Description: This machine-code program is used with OSI's Extended Monitor to provide single stepping of your computer to trace machine-code programs one instruction at a time. As you step through the program the display will show the mnemonic instruction before it is executed in the same form as the disassembler shows it. You can set high and low trace limits; this lets the program execute without displaying the instructions until the address is within these limits. The third feature is the ability to set a breakpoint. This will let you stop execution at any address in memory. This breakpoint, and tracing, can be done in ROM.

Price: $12.95
Includes 5¼" disk, documentation.

Author: Dave Pompea

Available:
DMP Systems
319 Hampton Blvd.
Rochester, N.Y. 14612

Name: **UTIL1**
System: AIM 65
Memory: 16K
Language: AIM Assembler
Hardware: Standard AIM

Description: *UTIL1* is a 2K extension of the AIM 65 monitor. It interfaces to AIM *via* the user I/O ports and the user function key 3. It adds 18 commands to AIM. Eight of these are associated with a Buffer Manager that gives AIM a virtual I/O capability. Up to 8 I/O devices are emulated in RAM. This gives the AIM editor move and copy capability. These I/O devices can be used with any AIM firmware or software that uses the AIM Active Output Device (AOD) and the Active Input Device (AID). An additional 10 commands provide utilities such as memory display, search and move, and an offset loader for AIM object files.

Price: $25 object on cassette
$5 16-page manual
$25 commented assembly listing
Includes object assembled to your specified address.
Should reside at top of RAM. Specify Address.

Author: Joel Swank

Available:
Nehalem Bay Software
P.O. Box 2006
Beaverton, OR 97075

Name: **Creature of the Maze**
System: Ohio Scientific
Memory: 8K
Language: BASIC-in-ROM
Hardware: Challenger C1P or Superboard Series I or II

Description: Incredibly realistic 3-D graphics lock you into combat with the "Creature of the Maze." Each game starts with a new and different maze, created and displayed for you to ponder, but only for a short moment. Then the screen clears and you find yourself looking down long corridors, peeking around corners and searching for your enemy. The hallways explode with your lazer blasts as the message across the screen spells out, "The monster is near." Tremendous fun with ten skill levels and hundreds of maze sizes to choose from.

Price: $14.95
Includes cassette, user's manual with objectives, options, and suggestions for modification

Author: John H. DeRosa

Available:
Dee Products
150 Birchwood Road
Lake Marion, IL 60110

MICRO

MICRO™

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6809 Bibliography

18. Commodore Interface (July, 1981)

Anon., "Super PET," pg. 18-20.
Questions and answers on the new Super PET based on a 6502 plus 6809.

19. MICRO No. 41 (October, 1981)

Puckett, Dale L., "The 6809 and the S-50 Bus," pg. 68-73.
The 6809 is much easier to work with than the 6800, and the programs are about 30 percent shorter and run about 30-40 percent faster.

Steiner, John, "The Radio Shack Color Computer: A 6809-based System," pg. 9-10.
The TRS-80 Color Computer is one of the most popular and versatile 6809-based systems to date. An outline of the features is given.

20. Apple Assembly Line 2, Issue 1 (October, 1981)

Wiggs, Chris and Sander-Cederlof, Robert, "6809 Cross Assembler," pg. 12.
Patches for the S-C Assembler Version 4.0 are available to give a brand new assembler for the 6809.

21. MICRO No. 42 (November, 1981)

Steiner, John, "Lunar Lander," pg. 41-44.
Animated graphics in BASIC for the 6809-based TRS-80 Color Computer.

Capouch, Brian, "OS-9 and the 6809: Revolutionary Tools," pg. 81-86.
The Microware OS-9 operating system is an advanced software package for the 6809. Also described is BASIC09, a high-level programming system (alternative to a BASIC interpreter or compiler). Illustrative listings of a pair of BASIC09 procedures in source code are given.

22. FWAUG Newsletter 2, No. 6 (October, 1981)

Hardenburg, Hal and McVay, Ray, "Concerning the 6502 and 6800," pg. 25-26.
Notes comparing the 6502, 6809 and the 68000 microprocessors, including a report that the Apple in the future may use the new 68000 microprocessor.

23. BYTE 6, No. 11 (November, 1981)

Walker, Gregory, "Toward a Structured 6809 Assembly Language, Part 1," pg. 370-382.
An introduction to structured assembly language for the 6809.

24. Stems From Apple 4, Issue 9 (September, 1981)

Hardenbergh, Hal W., "To Persons Interested in Both the 6502 and the 6800," pg. 5-14.
A comparison of the 6502, 6809 and the 68000 microprocessors, including an assembly language program for the 68000 multiply routine.

25. Compute! 3, No. 10, Issue 18 (November, 1981)

MacLean, Bill, "SuperPET: A Preview," pg. 38-40.
A rundown on the SuperPET micro which incorporates both the 6502 and the 6809.

26. BYTE 6, No. 12 (December, 1981)

Barden, William, "Color Computer from A to D," pg. 134-160.
A detailed look at the Radio Shack TRS-80 Color Computer, based on the 6809 microprocessor.

Walker, Gregory, "Toward a Structured 6809 Assembly Language," pg. 198-228.
Part 2 discusses implementing a structured assembler.

27. KB Microcomputing 5, No. 12, Issue No. 160 (December, 1981)

Stark, Peter A., "68XX Secrets," pg. 116-130.
A review of Dynamite, a good disassembler running on a 6809 FLEX 9 disk operating system. Notes on building a 6809 48K system.

28. Compute! 3, No. 12 (December, 1981)

Anon., "A Look at SuperPET," pg. 130-132.
Features of the CBM SuperPET and several useful utilities for this 6809-based system.

29. Compute! 4, No. 1, Issue 20 (January, 1982)

Mansfield, Richard, "BRANCH NEVER and QUIF Assembling on SuperPET," pg. 146-149.
Discussion of using some of the special 6809 statements available when assembling on the SuperPET.

30. MICRO No. 44 (January, 1982)

Tenny, Ralph, "Experimenters and the Color Computer," pg. 18-22.
A summary of the normal capabilities of the TRS-80 Color Computer and an examination of the unit's I/O capability. Also information on hardware for I/O use.

31. Apple Assembly Line 2, Issue 3 (December, 1981)

Sander-Cederlof, Bob, "EXCEL-9: A 6809 Card with FLEX," pg. 1.
A board with a 6809E CPU, 8K of ROM and an interval timer with built-in linkage routines for calling 6809 routines from Applesoft, Integer BASIC, or from 6502 machine language.

32. KB Microcomputing 6, No. 1, Issue 61 (January, 1982)

Wolf, Michael A., "Changing Chips in Midstream," pg. 96-100.
Discussion of the use of the 6809 microprocessor in the Radio Shack Color Computer.

33. KB Microcomputing 6, No. 2, Issue 62 (February, 1982)

Stark, Peter A., "6800 Secrets," pg. 84-98.
More bench tests on various microprocessor-equipped systems. Includes several related to the 6809 chip.

34. MICRO No. 45 (February, 1982)

Garrett, Leo E., "Utilities for the Color Computer," pg. 9-15.
A versatile routine allowing TRS-80 Color Computer users to dump or disassemble the 6809 or ASCII code in any section of memory, including the BASIC or expansion ROMs.

Staff, "MICRO Software Catalog," pg. 117-121.
Includes items of software for 6809 systems.

MICRO

MICRO™

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

6502 Bibliography

1. MICRO No. 43 (December, 1981)

Traeger, John C., "Data Collection with Your Micro," pg. 9-11.

How to construct and implement an interface which enables high-speed sampling and recording of experimental data. Written for an AIM 65, it is readily adapted to any 6502 microprocessor with either a 6502 or 6222 interface adapter.

2. PEEK(65) 2, No. 12 (December, 1981)

Cook, William H., "Add an 8-Inch Floppy to the C2-4F/C4P," pg. 2-4.

A hardware article for OSI users. Modifications for the 502 CPU board and detailed information on the interconnections of the 470 Floppy Disk Controller board with the disk drive are given. Thirteen signal lines are run to the Siemens FDD-100-8 drive.

3. BYTE 6, No. 12 (December, 1981)

Jacobs, Jacob R., "Generating Programs Automatically," pg. 352-362.

Let your Apple II do the programming. Three programs are written in Applesoft BASIC. These utilities help set up your desired program with data entry, data output, instructions, etc. Sample dialogs in running the program are given in the article.

4. BYTE 6, No. 12 (December, 1981)

Kopp, Gregory L, "Discovering Atari's 'Hidden' Graphics," pg. 98-102.

Improper graphics commands on the Atari often leads to unexpected results. Some of these undocumented commands may be used to advantage. A chart of useful 'hidden' commands are given and example listings demonstrate the effect.

5. Apple-Dayton 2, No. 12 (December, 1981)

Brungart, David L., "Organizing Applesoft," pg. 19-25.

Listing and dicussion of a program utility package used to set up temporary utility routines to ease the task of writing Applesoft programs in an orderly manner.

6. KB Microcomputing 5, No. 12, Issue No. 60 (December, 1981)

Young, John E., "Poor Man's Memory Expansion for the OSI," pg. 56-60.

An inexpensive way to expand the memory of the Superboard II or Challenger C1P. Instead of using a $300 OSI 610 expansion board, a method to implement a $30 16K static RAM board is described.

7. Call —A.P.P.L.E. 4, No. 9 (November/December, 1981)

Anon., "Puffin," pg. 13-42.

A DOS to Pascal File Converter for the Apple. A menu of four commands is presented: Catalog, Display, Transfer and Quit. Earlier a program called Huffin to convert Pascal files back to DOS was published (Call —A.P.P.L.E. Oct., 1982).

8. Softalk 2, No. 4 (December, 1981)

Coats, Douglas E. and Waldman, Cye H., "FORTRAN," pg. 160-172.

Comparisons of Apple FORTRAN and Microsoft FORTRAN for the Apple. Includes versions using "The Mill" and the Softcard accessories for the Apple.

9. Nibble 2, No. 8 (December, 1981)

Exner, Chris; Guy, Rudy; and Harvery, Mike, "Trend Reporting, Analysis, and Control," pg. 7-29.

A group of three extensions to the Apple-based TRAC system. Budget TRAC allows you to be aware of where your money goes, TRAC Spending Graph will graph the data in hi-res, and TRAC Plus shows how to get the best use of the system.

10. Creative Computing 7, No. 12 (December, 1981)

Brewster, Keith, "Who's Afraid of the Big Bad Matrix?", pg. 168-173.

Arrays and Matrix operations on the Atari are discussed and illustrated with numerous listings.

MICRO

Introducing...

A2·PB1

PINBALL

This beautiful and sophisticated simulation offers superior ball action, superior flipper control, and many subtle perfections which will result in your superior enjoyment.

Please don't compare A2-PB1 Pinball with other pinball simulations. Nothing else even comes close. $29.95

The engineering and graphics experts opening a new era in computer simulation.

subLOGIC

Communications Corp.
713 Edgebrook Drive
Champaign, IL 61820
(217) 359-8482
Telex: 206995

See your dealer . . .

or for direct orders, add $1.50 and specify UPS or first class mail.
Illinois residents add 5% sales tax.
Visa and MasterCard accepted.

Annual Index

June 1981 — May 1982
(Issues 37 - 48)

Articles

APPLE

OSI

(Continued on next page)

AIM

PET

KIM

GENERAL 6502

6809

Atari

SYM

MICRO

MICRObits

(Continued from page 87)

HTS OSI Products

Manual documents circuitry and provides software to implement Real-Time Clock. User-friendly system prompts and displays date/time. Add 4K CMOS RAM, 4K EPROM, and PORTS using designs documented in *General Expansion Board Construction Manual*. Each manual - $15.

Hunter Technical Services
P.O. Box 359
Elm Grove, WI 53122

PET/CBM Operating Systems 3.0

Matric expands Commodore BASIC with 14 new commands for handling arrays. Algebraic-style syntax. Checks for conformability. Extended error messages. Machine language (5K) program lets you display a matrix on screen and change its values, transfer data between matrices or fill a matrix with a constant, transpose, transfer diagonals between matrices, eigenvalues and eigenvectors. Specify size and ROM set, tape or disk. 32-page manual. Price: $125.00.

Cognitive Products
P.O. Box 1658
Bloomington, IN 47402

RTTY Without Hardware

Receive/Transmit RTTY with Egbert Disc Program. Apple generates and demodulates tones with screen display and other features. Apple II, 3.2/3.3, 48K, disk drive. State S.W.L. or HAM call. Send $39.95 plus $2.50 shipping (California residents add 6% tax).

W.H. Nail Co.
275 Lodgeview Drive
Oroville, CA 95965

COMPendium

COMPendium each month abstracts all articles in 20-plus microcomputer publications. Indispensable listings of software, hardware and book reviews. Find *all* information for *your* machine. Annual Index each December. Keeps track of published programs you half-remember. Sample - $1.50; yearly - $18.

Epicurious
P.O. Box 129
Lincolndale, NY 10540

Computer Covers

Keep dust and dirt out of your equipment with a computer cover. Cover for full Apple $11.50, disk drive $4.25, stack disk (2) $6.25, monitor (state size) $10.25. Send check plus $1.50 postage.

M.P. Computer Services
P.O. Box 2396 Encinal Stations
Sunnyvale, CA 94087

For Sale

System utilities, terminal emulator software, data encryption systems for secure communications. For OSI Superboard, Apple, other 6502 systems.

D. Wolf, Ph.D.
Box 565
Port Hueneme, CA 93041

Target - An AIM 65 Newsletter

Need information for your AIM 65 computer? News, software, and hardware are examples of items covered in the newsletter. Yearly subscription rate is $6.00 in the US and Canada, $12.00 elsewhere. Back issues are available beginning with 1979 at the same per year rate.

Target
C/O Donald Clem
RR#2
Spencerville, OH 45887

Joystick Interface

For PET, AIM 65, SYM, KIM or other 6502-based computer. Uses five VIA ports to give eight-bit conversion of up to eight resistance devices. Requires 64 bytes of memory (software included). Assembled, tested — $29.95; bare board - $12.95.

Sydney S. Koegler
Micro-K Computer Products
2339 Carriage Ave.
Richland, WA 99352

MICRO

PET/CBM

PET — "Personal Electronic Transactor"

CBM — "Commodore Business Machine"

- 6502-based computer, manufactured by Commodore Business Machines
- PET models include graphic keyboard
- CBM models include business keyboard
- Available in 4K, 8K, 16K, 32K, and 96K configurations
- All models, except 8000 series, include 25-row by 40-column screen. 8016, 8032, and 8096 have 25 by 80 screen.
- Two 256-character sets — one for graphics, one (with lower case) for business
- Memory expansion bus, parallel interface, IEEE-488 instrumentation bus standard
- Reliable cassette operating system, powerful screen editing and character-programmable cursor control are characteristic of PET/CBM.
- VIC and SuperPET have many features in common with PET/CBM.

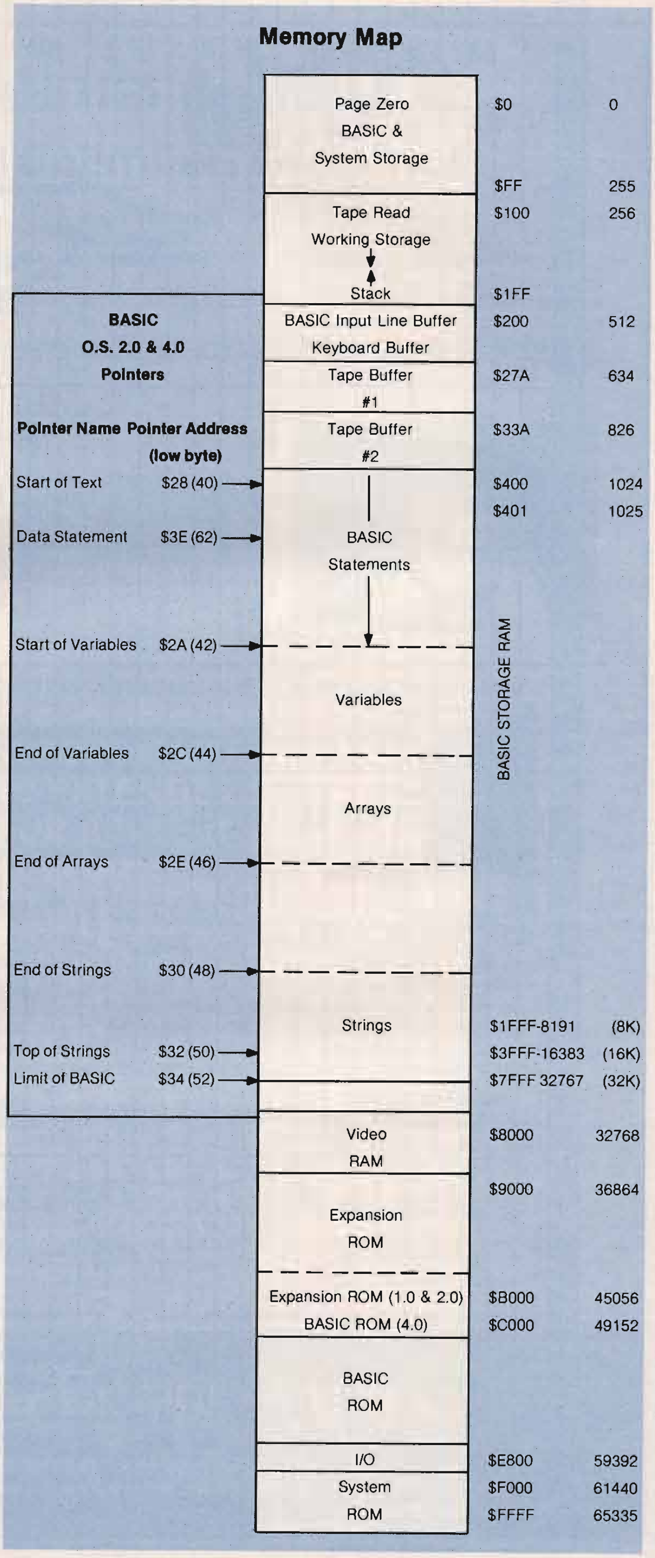
Parallel Port Connector

Upper Pin Identification Character	Signal Label	Lower Pin Identification Character	Signal Label
1	Ground	A	GND
2	T.V. Video	B	CA1
3	IEEE-SRQ	C	PA0
4	IEEE-EOI	D	PA1
5	Diagnostic Sense	E	PA2
6	Tape #1 READ	F	PA3
7	Tape #2 READ	H	PA4
8	Tape Write	J	PA5
9	T.V. Vertical	K	PA6
10	T.V. Horizontal	L	PA7
11	GND	M	CB2
12	GND	N	GND

IEEE-488 Connector

PET Edge-card Pin Numbers	Standard IEEE Connector Pin Numbers	IEEE Signal Mnemonic
Upper Pins		
1	1	DI01
2	2	DI02
3	3	DI03
4	4	DI04
5	5	EOI
6	6	DAV
7	7	NRFD
8	8	NDAC
9	9	IFC
10	10	SRQ
11	11	ATN
12	12	GND
Lower Pins		
A	13	DI05
B	14	DI06
C	15	DI07
D	16	DI08
E	17	REN
F	18	GND
H	19	GND
J	20	GND
K	21	GND
L	22	GND
M	23	GND
N	24	GND

Memory Map



PET/CBM

MICRO Data Sheet #4

Decimal	Hex	ASCII	Screen	BASIC	6502
0	00		@	end line	BRK
1	01		A		ORA(I,X)
2	02		B		
3	03		C		
4	04		D		
5	05	white	E		ORA Z
6	06		F		ASL Z
7	07	bell	G		
8	08	lock	H		PHP
9	09	unlock	I		ORA #
10	0A		J		ASL A
11	0B		K		
12	0C		L		
13	0D	carr. ret.	M		ORA
14	0E	text	N		ASL
15	0F	top-left	O		
16	10		P		BPL
17	11	cur. down	Q		ORA(I),Y
18	12	reverse	R		
19	13	cur. home	S		
20	14	delete	T		
21	15	del. line	U		ORA Z,X
22	16	erase begin	V		ASL Z,X
23	17		W		
24	18		X		CLC
25	19	scroll up	Y		ORA Y
26	1A		Z		
27	1B		[		
28	1C	red	\		
29	1D	cur. right	]		ORA X
30	1E	green	↑		ASL X
31	1F	blue	←		
32	20	space	space	space	JSR
33	21	!	!	!	AND(I,X)
34	22	"	"	"	
35	23	#	#	#	
36	24	$	$	$	BIT Z
37	25	%	%	%	AND Z
38	26	&	&	&	ROL Z
39	27	'	'	'	
40	28	(	(	(	PLP
41	29	)	)	)	AND #
42	2A	*	*	*	ROL A
43	2B	+	+	+	
44	2C	,	,	,	BIT
45	2D	–	–	–	AND
46	2E	.	.	.	ROL
47	2F	/	/	/	
48	30	0	0	0	BMI
49	31	1	1	1	AND(I),Y
50	32	2	2	2	
51	33	3	3	3	
52	34	4	4	4	
53	35	5	5	5	AND Z,X
54	36	6	6	6	ROL Z,X
55	37	7	7	7	
56	38	8	8	8	SEC
57	39	9	9	9	AND Y
58	3A	:	:	:	CLI
59	3B	;	;	;	
60	3C	<	<	<	
61	3D	=	=	=	AND X
62	3E	>	>	>	ROL X
63	3F	?	?	?	

Decimal	Hex	ASCII	Screen	BASIC	6502
64	40	@		@	RTI
65	41	A	,a	A	EOR(I,X)
66	42	B	,b	B	
67	43	C	,c	C	
68	44	D	,d	D	
69	45	E	,e	E	EOR Z
70	46	F	,f	F	LSR Z
71	47	G	,g	G	
72	48	H	,h	H	PHA
73	49	I	,i	I	EOR #
74	4A	J	,j	J	LSR A
75	4B	K	,k	K	
76	4C	L	,l	L	JMP
77	4D	M	,m	M	EOR
78	4E	N	,n	N	LSR
79	4F	O	,o	O	
80	50	P	,p	P	BVC
81	51	Q	,q	Q	EOR(I),Y
82	52	R	,r	R	
83	53	S	,s	S	
84	54	T	,t	T	
85	55	U	,u	U	EOR Z,X
86	56	V	,v	V	LSR Z,X
87	57	W	,w	W	
88	58	X	,x	X	CLI
89	59	Y	,y	Y	EOR Y
90	5A	Z	,z	Z	
91	5B	[		[	
92	5C	\		\	
93	5D	]		]	EOR X
94	5E	↑		↑	LSR X
95	5F	←		←	
96	60				RTS
97	61				ADC(I,X)
98	62				
99	63				
100	64				
101	65				ADC Z
102	66				ROR Z
103	67				
104	68				PLA
105	69				ADC #
106	6A				ROR A
107	6B				
108	6C				JMP(I)
109	6D				ADC
110	6E				ROR
111	6F				
112	70				BVS
113	71				ADC(I),Y
114	72				
115	73				
116	74				
117	75				ADC Z,X
118	76				ROR Z,X
119	77				
120	78				SEI
121	79				ADC Y
122	7A				
123	7B				
124	7C				
125	7D				ADC X
126	7E				ROR X
127	7F				

Decimal	Hex	ASCII	Screen	BASIC	6502
128	80		r-@	END	
129	81		r-A	FOR	STA(I,X)
130	82		r-B	NEXT	
131	83		r-C	DATA	
132	84		r-D	INPUT#	STY Z
133	85	f1	r-E	INPUT	STA Z
134	86	f3	r-F	DIM	STX Z
135	87	f5	r-G	READ	
136	88	f7	r-H	LET	DEY
137	89	f2	r-I	GOTO	
138	8A	f4	r-J	RUN	TXA
139	8B	f6	r-K	IF	
140	8C	f8	r-L	RESTORE	STY
141	8D	carr. ret.	r-M	GOSUB	STA
142	8E	graphic	r-N	RETURN	STX
143	8F	btm. right	r-O	REM	
144	90	black	r-P	STOP	BCC
145	91	curs. up	r-Q	ON	STA(I),Y
146	92	rvs. off	r-R	WAIT	
147	93	clear	r-S	LOAD	
148	94	insert	r-T	SAVE	STY Z,X
149	95	ins. line	r-U	VERIFY	STA Z,X
150	96	erase-end	r-V	DEF	STX Z,Y
151	97		r-W	POKE	
152	98		r-X	PRINT#	TYA
153	99	scroll dn.	r-Y	PRINT	STA Y
154	9A		r-Z	CONT	TXS
155	9B		r-[	LIST	
156	9C	magenta	r-\	CLR	
157	9D	curs. left	r-]	CMD	STA X
158	9E	yellow	r-↑	SYS	
159	9F	cyan	r-←	OPEN	
160	A0			CLOSE	LDY #
161	A1		r-!	GET	LDA(I,X)
162	A2		r-"	NEW	LDX #
163	A3		r-#	TAB(	
164	A4		r-$	TO	LDY Z
165	A5		r-%	FN	LDA Z
166	A6		r-&	SPC(	LDX Z
167	A7		r-'	THEN	
168	A8		r-(	NOT	TAY
169	A9		r-)	STEP	LDA #
170	AA		r-*	+	TAX
171	AB		r-+	—	
172	AC		r-,	*	LDY
173	AD		r--	/	LDA
174	AE		r-.	↑	LDX
175	AF		r-/	AND	
176	B0		r-0	OR	BCS
177	B1		r-1	>	LDA(I),Y
178	B2		r-2	=	
179	B3		r-3	<	
180	B4		r-4	SGN	LDY Z,X
181	B5		r-5	INT	LDA Z,X
182	B6		r-6	ABS	LDX Z,Y
183	B7		r-7	USR	
184	B8		r-8	FRE	CLV
185	B9		r-9	POS	LDA Y
186	BA		r-:	SQR	TSX
187	BB		r-;	RND	
188	BC		r-<	LOG	LDY X
189	BD		r-=	EXP	LDA X
190	BE		r->	COS	LDX Y
191	BF		r-?	SIN	

Decimal	Hex	ASCII	Screen	BASIC	6502
192	C0		Reverse of ASCII (192–223)	TAN	CPY #
193	C1	,a		ATN	CMP(I),Y
194	C2	,b		PEEK	
195	C3	,c		LEN	
196	C4	,d		STR$	CPY Z
197	C5	,e		VAL	CMP Z
198	C6	,f		ASC	DEC Z
199	C7	,g		CHR$	
200	C8	,h		LEFT$	INY
201	C9	,i		RIGHT$	CMP #
202	CA	,j		MID$	DEX
203	CB	,k		GO	
204	CC	,l		CONCAT	CPY
205	CD	,m		DOPEN	CMP
206	CE	,n		DCLOSE	DEC
207	CF	,o		RECORD	
208	D0	,p		HEADER	BNE
209	D1	,q		COLLECT	CMP(I),Y
210	D2	,r		BACKUP	
211	D3	,s		COPY	
212	D4	,t		APPEND	
213	D5	,u		DSAVE	CMP Z,X
214	D6	,v		DLOAD	DEC Z,X
215	D7	,w		CATALOG	
216	D8	,x		RENAME	CLD
217	D9	,y		SCRATCH	CMP Y
218	DA	,z		DIRECTORY	
219	DB				
220	DC				
221	DD				CMP X
222	DE				DEC X
223	DF				
224	E0				CPX #
225	E1				SBC(I),Y
226	E2				
227	E3				
228	E4				CPX Z
229	E5				SBC Z
230	E6				INC Z
231	E7				
232	E8				INX
233	E9				SBC #
234	EA				NOP
235	EB				
236	EC				CPX
237	ED				SBC
238	EE				INC
239	EF				
240	F0				BEQ
241	F1				SBC(I),Y
242	F2				
243	F3				
244	F4				
245	F5				SBC Z,X
246	F6				INC Z,X
247	F7				
248	F8				SED
249	F9				SBC Y
250	FA				
251	FB				
252	FC				
253	FD				SBC X
254	FE				INC X
255	FF				

MICRO Data Sheet #4

Advertiser's Index

MICRO INK, Inc is not responsible for claims made by its advertisers. Any complaint should be submitted directly to the advertiser. Please also send written notification to MICRO.

Next Month in MICRO

June Applications Feature

- **A Low-Cost Digitizer for the Apple II —** This article tells you how to use a sheet of half-reflecting plastic to build a digitizer for your high-resolution graphics computer.
- **AIM Logic Trainer —** Test your logic circuits with this program. It allows you to control the inputs and send the outputs to LEDs or printer.
- **Face Synthesizer —** This program creates an animated face on a PET screen that changes expression on keyboard command. Animation can also be controlled from BASIC programs, so the PET face can be used in any application — education, marketing, games, etc.
- **Program for Inverting a Matrix —** Include this short BASIC routine in your mathematical and statistical programs.

Plus...

AIM User Function Dispatcher
Additional Output Ports on the KIM
Tiny Pilot for the PET
Disk to Tape Backup Utility for the Apple
Sequential File Handler for C1P

1982 Editorial Schedule

July — Apple
August — Programming Techniques
September — 68000
October — Education
November — Atari/Games
December — PET

WE MAY HAVE ALREADY BUILT THE BOARD YOU ARE STARTING TO DESIGN!

While you may not have heard of us before, you certainly know our customers: Fortune 500 companies, Universities and Government Agencies. Since 1976 we have been providing high quality microcomputer products, ranging from expansion boards, to stand-alone controllers, to complete systems. Before you start your next project, consider how easy it might be to use some of our products.

FLEXI PLUS

A multi-function controller which handles:
- **8" and mini diskettes, double-sided, double-density**
- **RS-232 Communications with programmable features**
- **IEEE-488 Instrumentation Bus fully implemented**
- **6809E Microprocessor, up to 56K RAM, ROM, EPROM**
- **Parallel/Serial I/O, Cassettes and TTY Interfaces**

This versatile controller may be used as an expansion board for any 6502 or 6809 system; as a stand-alone controller; or, as the basis of a complete microcomputer system.

MICRO PLUS

A video-oriented controller which includes:
- **Programmable display up to 132 columns by 30 rows**
- **Programmable character sets in EPROM and RAM**
- **Character and Bit-Mapped Graphics**
- **ASCII Keyboard and Light Pen Interfaces**
- **RS-232 Programmable Communications Interface**
- **6502 Microprocessor, up to 7K RAM, 2K EPROM**

This video controller may be used to expand almost any 6502- or 6809-based system; or as a stand-alone intelligent terminal; or, as the basis for a complete 6502-based computer system.

DRAM PLUS

A multi-purpose expansion board which features:
- **Up to 40K RAM memory with a memory manager**
- **Up to 16K ROM or EPROM memory**
- **EPROM Programmer for 2516, 2716, 2532 and 2732**
- **Multiple parallel/serial I/O ports and timers/counters**
- **Prototyping area for custom circuits**

This memory-oriented expansion board permits addressing of memory on 4K boundaries, supports swapping of sections of memory, and works with most 6502- or 6809-based systems.

FOCUS

An Industrial quality system which features:
- **Two mini disk drives, double-sided, double-density for over 640K bytes of on-line storage**
- **Commercial quality keyboard with numeric pad**
- **Upper/lower case ASCII with programmable characters and display formats, plus bit-mapped graphics**
- **High-resolution video monitor with green phosphor**
- **Heavy-duty aluminum case for desk or rack mounting**
- **RS-232 Communications built in; second optional**
- **IEEE-488 may be added to existing boards**
- **Includes 48K RAM, 4K EPROM, 6809 microprocessor**

This extremely versatile system may be used for system development, for developing stand-alone products, as an end-user system, as an in-house business computer, as a word processor, and much more. Software is available from a number of suppliers and includes compiled BASIC, PASCAL, FORTH and many complete application packages.

$3495.00

If you have a requirement which involves 6502- or 6809-based products, join the growing number of OEMs and System Integration Houses who look to us first. For additional information and our current product literature, please contact us at 617/256-3649 or TELEX 955318 INTL DIV.

THE COMPUTERIST®

34 Chelmsford Street
Chelmsford, MA 01824