# MICRO™

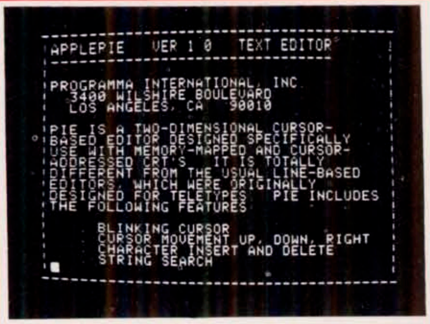
The Magazine of the APPLE, KIM, PET

and Other 6502 Systems


## APPLE MEMORY MAP

NO 15 August 1979 $2.00

# Dynamite Apple® software from RAINBOW

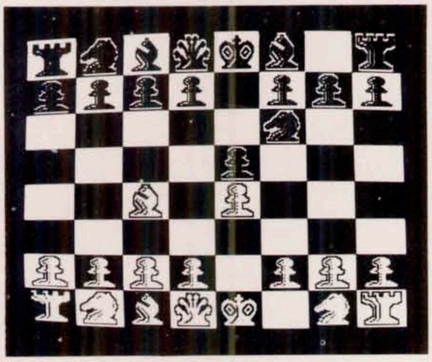



**PIE TEXT EDITOR** Machine Language, cursor-based text editor for 16K Apple.

- Features format capabilities of most text editors.
- All commands are control characters.
- Enables you to define your own function commands.

**Order PIE on Cassette: . . . . . . . . . $19.95**
**On Diskette . . . . . . . . . . . . . . $24.95**

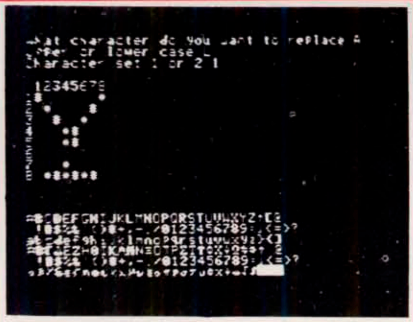
## NEW!



**SARGON** for 24K Apple

- Flip back and forth between board and text with ESC.
- Correct wrong moves
- Analyze your position

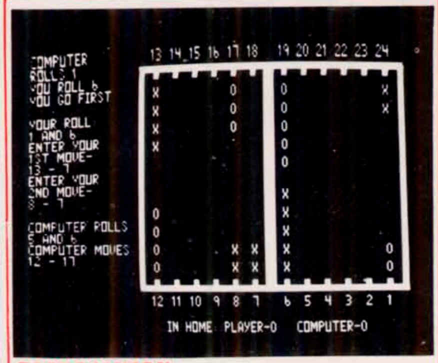
**Order SARGON on Cassette. . . . . $19.95**



**HIGH RESOLUTION CHARACTER GENERATOR** Machine language program for 16K Apple.

- Define your own character set and graphic shapes.
- Complete English upper/lower case character set.
- Complete Greek Alphabet with upper/lower character set.
- Scroll, vary window size, invert characters, switch back and forth between two character sets.

**Order Hi-Res Char. Gen. on Diskette $19.95**



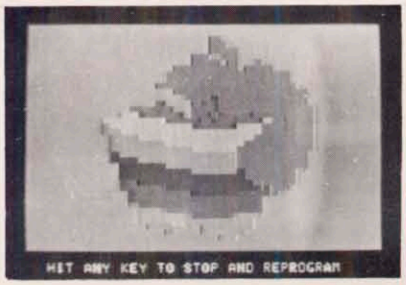

**FASTGAMMON**

A high quality, challenging game for you and the computer.

**Order FASTGAMMON on Cassette. $19.95**
**On Diskette . . . . . . . . . . . . . . . $24.95**

Call or write today for your **FREE Apple Software Catalog.** We welcome B/A-VISA and Mastercharge. Sorry, no CODs. Please add $1.25 shipping and handling. California residents add 6% Sales Tax.

We ship promptly on receipt of your prepaid order. Order direct from:




**3-D ANIMATION**

- Define a 3-D lo-res shape.
- Animate with full perspective.
- Includes 3 demo shapes.

**Order 3-D ANIMATION on diskette .$24.95**

RAINBOW COMPUTING INC.

Garden Plaza Shopping Center
9719 Reseda Blvd., Northridge, Ca 91324 (213) 349-5560

# POWERSOFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

## products for the APPLE II

### APPLESOFT II UTILITY (Diskette Only) $12.45

The Applesoft II Utility program provides the user with the following features. a) Complete automatic renumbering of any Applesoft II program. b) The creation of an EXEC File for subroutine file creation. This feature allows you to incorporate the same subroutine in various programs. c) No modification of the program in machine memory (RAM). d) Automatic running of the program. No programmer should be without this excellent utility program. REQUIREMENTS: Disk II, Applesoft II, 16K of memory.

### REAL ESTATE ANALYSIS PROGRAM $14.95

The Real Estate Analysis Program provides the user with three features. a) A powerful real estate investment analysis for buy/sell decisions and time to hold decisions for optimal rental/commercial investments. b) Generation of complete amorization schedules consistent with banking practices and schedules. c) Generation of depreciation schedules for selecting the best depreciation schedule for your use and a determination of optimal switch over points to straight line to maximize depreciation. All three features are designed for video screen or printer output. In addition, the program will plot; cash flow before taxes vs. years, cash flow after taxes vs. years, adjusted basis vs. years, capital gains vs. years, pre-tax proceeds vs. years, post-tax proceeds vs. years, and return on investment (%) vs. years. REQUIREMENTS: Applesoft II, 16K of memory without DOS or 32K of memory with DOS (Disk II).

### ADDRESS FILE GENERATOR $19.95

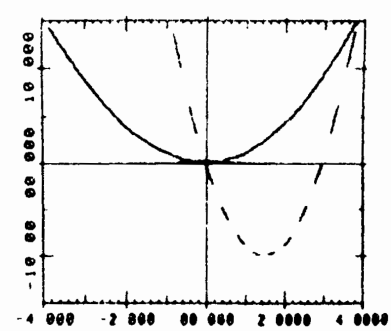
A professional piece of software which allows the user to create four different types of address files: a) Holiday File, b) Birthday File, c) Home Address File, and d) Commercial Address File. The program contains a menu of seven major commands: 1) Create a File, 2) Add to File, 3) Edit File, 4) Display File, 5) Search File, 6) Sort File, and 7) Reorganize File. Most of the major commands have subordinate commands which adds to the flexibility of this powerful software system. We doubt you could buy a better program for maintaining and printing address files. REQUIREMENTS: Disk II, Apple Printer Card, 32K of memory with Applesoft ROM Card or 48K of memory without Applesoft ROM Card.

### SUPER CHECKBOOK $19.95

A totally new checkbook program with a unique option . . . Bar Graphs. These bar graphs, outputed to a printer or video screen, provide trend analysis data on code expense, income, expenses, or gain/loss on a month by month basis. The program contains a total of fourteen options: 1) Check/Deposit Entry & Modification, 2) Reconciliation of Checks or Deposits, 3) Sort by Check Number, 4) Sort by Code for Year, 5) Sort by Code for Month, 6) Output Year to Date, 7) Output Month Activity, 8-11) Printer/Video Plot Trend Analysis-Bar Graphs, 12) Account Status, 13) Reconciled Check Status, and 14) Quit. An excellent program for maintaining your checkbook, or that of a small business. REQUIREMENTS: Disk II, 32K of memory with Applesoft ROM Card or 48K of memory without Applesoft ROM Card.

### FUNCTION GRAPHS AND TRANSFORMATIONS $14.95

This program uses the Apple II high resolution graphics capabilities to draw detailed graphs of mathematical functions which the user defines in Basic syntax. The graphs appear in a large rectangle whose edges are X and Y scales (with values labeled by up to 6 digits). Graphs can be superimposed, erased, drawn as dashed (rather than solid) curves, and transformed. The transformations available are reflection about an axis, stretching or compressing (change of scale), and sliding (translation). The user can alternate between the graphic display and a text display which lists the available commands and the more recent interactions between user and program. Expected users are engineers, mathematicians, and researchers in the natural and social sciences; in addition, teachers and students can use the program to approach topics in (for example) algebra, trigonometry, and analytic geometry in a visual, intuitive, and experimental way which complements the traditional, primarily symbolic orientation. REQUIREMENTS: 16K of memory with Applesoft ROM Card or 32K of memory without Applesoft ROM Card.



**Available at your local computer store**

*Call or write for our free SOFTWARE & ACCESSORIES CATALOG*

DEALER INQUIRIES INVITED

**POWERSOFT, INC.**

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

- Check or Money Order
- Include $1.00 for shipping and handling
- C.O.D. ($1.00 add'tl. charge)
- Master Charge and VISA orders accepted
- New Jersey residents add 5% sales tax

*Programs Available on Diskette at $5.00 Additional*

HDE inc.

BOX 120
ALLAMUCHY, N.J. 07820
201-362-6574

# HUDSON DIGITAL ELECTRONICS INC.

# THE HDE DISK SYSTEM.

## HERE'S WHAT ONE USER HAS TO SAY . . .

### REPRINTED BY PERMISSION FROM THE 6502 USER NOTES - ISSUE NO. 14

PRODUCT REVIEW of the HDE DISC SYSTEM by the editor.

A number of you have asked for details about the HDE full size disc system.

The system is based around the SYKES 8" drive with the 6502 based intelligent controller.

This drive is soft sectored, IBM compatible, and single density which lets you store about a quarter megabyte of data on a disc.

The system software, called FODS (File Oriented Disc System), manages sequential files on the disc much the same way files are written on magnetic tape - one after another. When a file is deleted, from a sequentially managed file system, the space that the file occupied is not immediately reallocated, as in some disc operating systems. As it turns out, this can be an advantage as well as a disadvantage since deleted files on the FODS system can be recovered after the file has been deleted. (This has saved my sanity more than once!) Of course when you want to recover some of the disc space taken up by a number of these deleted files, you can simply re-pack or compress the disc and all the active files will be shifted down until there are no deleted files hanging around using up space.

FODS has this ability to repack a disc.

When saving and loading in FODS you work with named files, not track and sector data or I.D. bytes. This makes life a lot easier. I've seen some disc systems where you have to specify track and sector info and/or I.D. bytes. What a pain that can be!

If you just want to save a source file temporarily, you can do that on what's known as "scratch-pads". There are two of these on a disc, "scratch-pad A" and "scratch-pad B", each of these temporary disc files can hold up to 16K or if "B" is not used, "A" can hold one file up to 32K in length. The only files that can be temporarily saved on scratch pad are files that have been built using the system text editor.

Being a dyed in the wool assembly language programmer, I really appreciate the FODS text editor! This line oriented editor is upwards compatible with the MOS/ARESCO editor but includes about everything you could ask for in a line editor. There is a full and semi-automatic line numbering feature, lines can be edited while they are being entered or recalled and edited later, strings can be located and substituted, the line numbers can be resequenced, the file size can be found, the hex address of a line can be known and comments can be appended to an assembly file after it has been found correct. Oops! I forgot to say lines can also be moved around and deleted. This isn't the complete list of FODS editor commands, just the ones that immediately come to mind.

Another very powerful feature of the system is the ability to actually execute a file containing a string of commands. For example, the newsletter mailing list is now being stored on disc. When I want to make labels, I would normally have to load each letter file and run the labels printing program. But with FODS, I can build up a "JOB" file of commands and execute it.

The job file in turn calls each lettered label file in and runs the label printer automatically. The way computers are supposed to operate right?

Here's a listing of the job file I use to print mailing labels:

```
:LIS PRTLBL
0005 LOD A:RUN %LABEL:LOD B:JMP.E000:
LOD C:JMP.E000:
0010 LOD D:JMP.E000:LOD E:JMP.E000:
LOD F:JMP.E000:
0015 LOD G:JMP.E000:LOD H:JMP.E000:
LOD I:JMP.E000:
0020 LOD J:JMP.E000:LOD K:JMP .E000:
LOD L:JMP.E000:
0025 LOD M:JMP.E000:LOD MC: JMP.E000:
LOD N:JMP.E000:
0030 LOD O:JMP.E000:LOD P.JMP .E000:
LOD R:JMP.E000:
0035 LOD S:JMP.E000:LOD T:JMP .E000:
LOD V:JMP.E000:
0035 LOD S:JMP.E000:LOD T:JMP .E000:
LOD V:JMP.E000:
0040 LOD W:JMP.E000:LOD XYZ: JMP.E000:
0045 LOD EXCH:JMP.E000:LOD COMP:
JMP.E000:
```

Remember the MOS/ARESCO assembler I reviewed several issues ago? Well HDE went and fixed up all the problem areas that I mentioned in the review and then took it several steps further. The HDE assembler is an honest to goodness two-pass assembler which can assemble anywhere in memory using multiple source files from the disc. The assembler is an optional part of the system.

If you're the kind of person (as I am) who enjoys having the ability to customize, modify, and expand everything you own - you'll enjoy the system expansion abilities FODS has to offer. Adding a new command is as simple as writing the program, giving it a unique three letter name and saving it to disc. Whenever you type those three letters the system will first go through its own command table, see that its not there and then go out and read the disc directory to see if it can find it. If it's on the disc it will read it in and execute it. Simple right? I've added several commands to my system and REALLY appreciate having this ability. Some of the things I've added include a disassembler, an expanded version of XIM (the extended machine language monitor from Pyramid Data), Hypertape, and a number of system utilities which make life easier. By the way, to get back to the system, all you need to do is execute a BRK instruction.

HDE also provides a piece of software that lets you interface Microsoft 9 digit BASIC to their disc system. The software allows you to load the BASIC interpreter itself from disc as well as saving and loading BASIC Programs to and from the disc. This particular version of the software doesn't allow for saving BASIC data but HDE mentioned that this ability may be possible with a future version.

The first thing I do with a new piece of software after I get used to using it is try to blow it up. I did manage to find a weak spot or two in the very first version of FODS (a pre-release version) but the later release version has been very tight.

The standard software that is included with the system consists of the disc driver software, the system text editor and the BASIC software interface. Several command extensions may also be included. All the necessary stuff like a power supply, the KIM-4 interface card, and all cables and connectors are included. It took me about 45 minutes to get things up and running the first time I put the system together.

Admittedly, a dual full size disc system from HDE is probably beyond the means of most hobbyists but if you or your company is looking for a dynamite 6502 development system, I would recommend this one. I've used the Rockwell System 65 while I was at MOS and feel that dollar for dollar, feature for feature, the HDE system comes out on top. The only place the HDE system falls short when stacked up next to the System 65 is in the area of packaging. At this point, there is no cabinet for the disc drives available from HDE.

So far, I've got nothing but good things to say about HDE and their products. Everything I've received from them has been industrial quality. That includes their documentation and product support. I'm very impressed with what I've seen from this company so far and quite enthusiastic over what my KIM has become since acquiring the disc system and its associated software.

ERIC

***THANK YOU MR. REHNKE!***

## HDE PRODUCTS - BUILT TO BE USED WITH CONFIDENCE
## AVAILABLE DIRECT OR FROM THESE FINE DEALERS:

| JOHNSON COMPUTER | PLAINSMAN MICROSYSTEMS | ARESCO | LONG ISLAND COMPUTER GENERAL STORE | LONE STAR ELECTRONICS |
|---|---|---|---|---|
| Box 523 | Box 1712 | P.O. Box 43 | 103 Atlantic Avenue | Box 488 |
| Medina, Ohio 44256 | Auburn, Ala. 36830 | Audubon, Pa. 19407 | Lynbrook, N.Y. 11563 | Manchaca, Texas 78652 |
| 216-725-4560 | 800-633-8724 | 215-631-9052 | 516-887-1500 | 512-282-3570 |

MICRO™

August 1979
Issue Number Fifteen

## Table of Contents



**Staff**

**Publisher**
Robert M. Tripp

**Editor**
Shawn Spilman

**Business Manager**
Maggie E. Fisher

**Circulation Manager**
Carol A. Stark

**Distribution**
Eileen M. Enos
Janet Santaguida

**Micro-Systems Lab**
James R. Witt, Jr.
Stephen L. Allen

**Comptroller**
Donna M. Tripp

MICRO™ is published monthly by:
MICRO Ink, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515
Mailing address for all correspondence, subscriptions and address changes is:
MICRO
P. O. Box 6502
Chelmsford, MA 01824
Application to mail at second-class postage rates is pending at: Chelmsford, MA 01824.
Publication Number: COTR 395770
Subscription in United States:
$15.00 per year/12 issues.


## Advertiser's Index

# APPLE II Serial Output Made Simple

**Is the APPLE II simple serial output as easy to implement as everyone claims? Almost! But a few helpful hints gleaned from this designer's experience may get that output port into service quite a bit sooner.**


**Donald W. Bixby**
**5 King Philip Trail**
**Norfolk, MA 02056**


When Apple sent the new **Apple II Reference Manual** (January 1978), I jumped at the article on page 114, "A Simple Serial Output". A printer output was badly needed in my system. I built the RS-232 output as described, typed in the program, borrowed a terminal from my place of business and started things up.

An oscilloscope on the RS-232 output disclosed that the signal was reaching +12v, but going only slightly negative.

The printer did work correctly, but I was concerned. Examination of the RS-232C specification disclosed that the printer on the data receiving end must have 3K input impedance. The printer manual stated only that the impedance was "at least" 3K. Since the Apple circuit was uses a 2.2K resistor to −12v, the source impedance, when negative, is much too high. I replaced the Apple circuit with a single inverter (74LS04) driving an RS-232 driver integrated circuit manufactured by Motorola (MC1488L). This worked fine.

The only other hardware problem related to page 115 in Apple's manual. The statement, "The signal output connects to pin 3 of the DB-25 connector", is confusing. It is correct if you are connecting it to a DB-25 connector, which is to be used with a standard RS-232 cable with the other end of the cable connected to the printer. The cable connects pin 3 at the source end to pin 2 of the receiving end. If you are connecting directly to the printer, use pin 2, not pin 3.

Now the fun began. The printer I used can be operated at 110 baud, 150 baud, or 300 baud, front panel switch selectable. Apple's program was all written for 110 baud. Naturally I wanted the fastest speed. For any speed higher than 110 baud, 1 stop bit is used instead of 2. This is easily changed by writing location $03C6 with 0A.

The routine TTOUT4 causes a 9.091 msec. delay (1/110 baud = 9.091 ms). For 300 baud, I needed 3.333 ms. This was accomplished by changing location 03D4 from D7 to 4E.

The printer will now work at 300 baud with three problems remaining. The first was simple, the second took two weeks to figure out and the third was minor.

When a carriage return is transmitted, the program sends the carriage return to the printer, then automatically sends a line feed to the printer, then waits 200 ms for the carriage return to be completed. My printer requires the 200 ms. dealy, but others will be different. For example, the DECwriter requires no delay. After speeding up to 300 baud, I was not getting enough delay. I changed location 03AC from 58 to FF, an arbitrary choice, and this problem was fixed.

The program is supposed to detect when the next column to be printed, COLCNT, exceeds the number of columns available, WNDWDTH, and then transmit an automatic CR, LF, and delay. It won't, it can't and it didn't. The intention of the Apple program routine, FINISH is to make CH equal to 39 and then depend on the system monitor routines to generate the CR, LF and delay. This doesn't work.

I have modified their program to make this happen within the TTY routines. If COLCNT equals or exceeds WNDWDTH, the program branches to RETURN. This causes a carriage return and then branches to AUTOLF, the same section of program used for automatic line feed and delay by Apple.

The last problem encountered involved getting out of the printer routines and back to the video display. New code was written to solve this problem.

The new program, shown here, has been relocated to addresses 30A through 3A2. With all the components, I believe it is self explanatory. I also wrote an AppleSoft BASIC program to modify and test the machine language program.

```
10 REM PRINTER TEST AND MODIFY PROGRAM IN APPLESOFT BASIC
15 CALL -936:PRINT:PRINT
20 INPUT "110 OR 300 BAUD";A
30 IF A=110 THEN 70
40 POKE 868,10
50 POKE 882,78
52 PRINT:INPUT "200 OR 0 MS CARRIAGE RETURN DELAY";M
54 IF M=200 THEN M=255
60 POKE 843,M
70 PRINT:INPUT "HOW MANY CHARACTERS TO A LINE";N
80 POKE 787,N
90 PRINT:PRINT
100 PRINT N;"CHARACTERS TO A LINE"
110 IF A=300 THEN 220
120 POKE 868,11
130 POKE 882,215
132 PRINT:INPUT "200 OR 0 MS CARRIAGE RETURN DELAY";M
134 IF M=200 THEN M=88
140 POKE 843,M
220 PRINT:PRINT:INPUT "CHARACTERS TO BE PRINTED";A$
230 PRINT:PRINT:PRINT A$
240 PRINT:PRINT:PRINT "OUTPUT IS NOW GOING TO THE PRINTER AT A";A;"BAUD
RATE"
250 CALL 778
260 FOR J=1 TO 10
270 PRINT A$
280 NEXT J
300 CALL 914
310 PRINT:PRINT
320 INPUT "CONTINUE (Y OR N)";B$
330 IF B$="Y" THEN 230
340 END
```

```
RS-232 DRIVER ROUTINES
REVISED 3-30-79 BY DONALD W. BIXBY
REVISED 6-6-79 BY MICRO STAFF

TO CALL TTINIT FROM SYSTEM MONITOR:  *<30AG
TO CALL VIDINIT FROM SYSTEM MONITOR: *<392G

TO CALL TTINIT FROM FP BASIC:  CALL 778
TO CALL VIDINIT FROM FP BASIC: CALL 914

TO READ FROM TAPE:    *30A.3A2R
TO WRITE TO TAPE:     *30A.3A2W

TO MAKE CHANGES:

 TO CHANGE WINDOW WIDTH:

     *<313:48      (FOR 72 COLUMNS)
     *<313:50      (FOR 80 COLUMNS)
     ]POKE 787,72 (FOR 72 COLUMNS)
     ]POKE 787,80 (FOR 80 COLUMNS)

 TO CHANGE CARRIAGE RETURN DELAY:

     *<34B:58
     ]POKE 843,88

 TO CHANGE NUMBER OF STOP BITS:

     *364:0A       (FOR 1 STOP BIT)
     *364:0B       (FOR 2 STOP BITS)
     ]POKE 868,10  (FOR 1 STOP BIT)
     ]POKE 868,11  (FOR 2 STOP BITS)

 TO CHANGE THE BAUD RATE:

     *372:7D       (FOR 110 BAUD)
     *372:4E       (FOR 300 BAUD)
     ]POKE 882,215 (FOR 110 BAUD)
     ]POKE 882,78  (FOR 300 BAUD)

03A3              WNDWDT *     $0021  FOR THE APPLE II
03A3              CH     *     $0024  CURSOR HORIZONTAL POSITION
03A3              CSWL   *     $0036  CHARACTER OUT SWITCH LO ORDER
03A3              CSWH   *     $0037  CHARACTER OUT SWITCH HI ORDER
03A3              YSAVE  *     $0308
03A3              COLCNT *     $0307  COLUMN COUNT LOCATION
03A3              MARK   *     $C058
03A3              SPACE  *     $C059
03A3              WAIT   *     $FCA8
03A3              RTS1   *     $0309
030A                     ORG   $030A
030A A9 21        TTINIT LDAIM $0021  EQUALS TTOUT-768 POINTER TO
030C 85 36               STA   CSWL   RS-232 ROUTINES, LOW BYTE
030E A9 03               LDAIM $0003  EQUALS TTOUT/256
0310 85 37               STA   CSWH   HIGH BYTE
0312 A9 48               LDAIM $0048
0314 85 21               STA   WNDWDT 72 COLUMN WINDOW WIDTH
0316 A5 24               LDA   CH
0318 8D 07 03            STA   COLCNT PRESENT COLUMN
031B A9 60               LDAIM $0060
031D 8D 09 03            STA   RTS1   STORE CONSTANT
0320 60                  RTS          RETURN FROM TTINIT
0321 48           TTOUT  PHA          SAVE CHARACTER ON THE STACK
0322 48                  PHA
0323 AD 07 03     TTOUT2 LDA   COLCNT
0326 C5 24               CMP   CH     CHECK FOR A TAB
0328 68                  PLA          RESTORE CHARACTER
0329 B0 03               BCS   TESTCT IF CARRY SET, NO TAB
032B 48                  PHA
```

```
033D 85 3E           STA   A2L
033F A5 6E           LDA   $006E
0341 85 3F           STA   A2H
0343 A0 00           LDYIM $00
0345 20 2C FE        JSR   $FE2C   USE MONITOR MOVE ROUTINE
0348 38              SEC           COMPUTE DISPLACEMENT
0349 A5 6B           LDA   $006B   TO ARRAYS
034B E5 69           SBC   $0069
034D 85 1C           STA   EL
034F A5 6C           LDA   $006C
0351 E5 6A           SBC   $006A
0353 85 1D           STA   EH
0355 60              RTS           BACK TO BASIC
              *
0356 A5 1A    RECALL LDA   CL      ***ENTRY 770 - RECALL VARIABLES
0358 85 3C           STA   A1L     SET UP MOVE
035A A5 1B           LDA   CH
035C 85 3D           STA   A1H
035E A5 18           LDA   DL
0360 85 6F           STA   $006F   START OF STRINGS
0362 85 3E           STA   A2L
0364 A5 19           LDA   DH
0366 85 70           STA   $0070
0368 85 3F           STA   A2H
036A A5 69           LDA   $0069   START OF NUMERICS
036C 85 42           STA   A4L
036E A5 6A           LDA   $006A
0370 85 43           STA   A4H
0372 A0 00           LDYIM $00
0374 20 2C FE        JSR   $FE2C   USE MONITOR MOVE ROUTINE
0377 18              CLC           COMPUTE START
0378 A5 69           LDA   $0069   OF ARRAYS
037A 65 1C           ADC   EL
037C 85 6B           STA   $006B
037E A5 6A           LDA   $006A
0380 65 1D           ADC   EH
0382 85 6C           STA   $006C
0384 38              SEC           COMPUTE END OF NUMERICS
0385 A5 6F           LDA   $006F
0387 E5 1A           SBC   CL
0389 85 6D           STA   $006D
038B A5 70           LDA   $0070
038D E5 1B           SBC   CH
038F 85 6E           STA   $006E   TEMP STORAGE
0391 18              CLC
0392 A5 6D           LDA   $006D
0394 65 69           ADC   $0069
0396 85 6D           STA   $006D   TEMP VALUE
0398 A5 6E           LDA   $006E
039A 65 6A           ADC   $006A
039C 85 6E           STA   $006E   TEMP VALUE
039E A5 6D           LDA   $006D   SUBTRACT ONE
03A0 D0 02           BNE   A2
03A2 C6 6E           DEC   $006E   END OF NUMERICS
03A4 C6 6D    A2     DEC   $006D
03A6 60              RTS           BACK TO BASIC
              * ROUTINE TO SAVE AND RECALL
              * COMMON VARIABLES FOR INTEGER BASIC
              * PROGRAMS ON THE APPLE II
              *
              * WRITTEN 03/16/79 BY ROBERT F. ZANT
              * MODIFIED 7/4/79 BY MICRO STAFF
              *
0318          CL     *     $001A
0318          CH     *     $001B
0302                 ORG   $0302
0302 4C 0F 03        JMP   RECALL  ***ENTRY 770
0305 00              BRK
0306 A5 CC           LDA   $00CC   ***ENTRY 774 - SAVE VARIABLES
0308 85 1A           STA   CL      SAVE END OF
030A A5 CD           LDA   $00CD   VARIABLE TABLE
030C 85 1B           STA   CH
030E 60              RTS           BACK TO BASIC

030F A5 1A    RECALL LDA   CL      ENTRY 770 - RECALL VARIABLES
0311 85 CC           STA   $00CC   RESET END OF
0313 A5 1B           LDA   CH      VARIABLE TABLE
0315 85 CD           STA   $00CD
0317 60              RTS           BACK TO BASIC
```

```
032C A9 A0           LDAIM $00A0  PRINT A SPACE
032E 2C 09 03 TESTCT BIT   RTS1   IS CHARACTER A CONTROL?
0331 F0 03           BEQ   PRNTIT IF SO, BRANCH TO PRINT IT
0333 EE 07 03        INC   COLCNT IF NOT, INCREMENT COLUMN COUNT
0336 20 5F 03 PRNTIT JSR   DOCHAR PRINT THE CHARACTER
0339 68              PLA          RESTORE CHARACTER
033A 48              PHA          AND PUT BACK ON THE STACK
033B 90 E6           BCC   TTOUT2 DO MORE SPACES FOR TAB CHAR
033D 49 0D           EORIM $000D  CHECK FOR CARRIAGE RETURN
033F 0A              ASLA         ELIMINATE PARITY
0340 D0 0D           BNE   FINISH DONE UNLESS HAVE CARRIAGE RETU
0342 8D 07 03 AUTOLF STA   COLCNT CLEAR COLUMN COUNTER
0345 A9 8A           LDAIM $008A
0347 20 5F 03        JSR   DOCHAR PRINT A LINE FEED
034A A9 58           LDAIM $0058
034C 20 A8 FC        JSR   WAIT   200 MS DELAY FOR CR LF
034F AD 07 03 FINISH LDA   COLCNT
0352 F0 07           BEQ   SETCH  BRANCH IF COLUMN COUNTER = 0
0354 E5 21           SBC   WNDWDT ELSE SUBTRACT WINDOW WIDTH
0356 B0 30           BCS   RETURN RETURN IF IN THE MARGIN
0358 AD 07 03        LDA   COLCNT
035B 85 24    SETCH  STA   CH     STORE NEW VALUE IN CH
035D 68              PLA          RESTORE THE STACK
035E 60              RTS          RETURN FROM TTOUT
035F 8C 08 03 DOCHAR STY   YSAVE  ROUTINE TO PRINT A
0362 08              PHP          CHARACTER
0363 A0 0B           LDYIM $000B  FOR 11 BITS
0365 18              CLC          (2 STOP BITS)
0366 48       TTOUT3 PHA
0367 B0 05           BCS   MARKOU
0369 AD 59 C0        LDA   SPACE  SEND A SPACE
036C 90 03           BCC   TTOUT4
036E AD 58 C0 MARKOU LDA   MARK   SEND A MARK
0371 A9 D7    TTOUT4 LDAIM $00D7  DELAY 9.091 MS FOR 110 BAUD
0373 48       DLY1   PHA
0374 A9 20           LDAIM $0020
0376 4A       DLY2   LSRA
0377 90 FD           BCC   DLY2
0379 68              PLA
037A E9 01           SBCIM $0001
037C D0 F5           BNE   DLY1
037E 68              PLA
037F 6A              RORA         NEXT BIT
0380 88              DEY          DECREMENT Y
0381 D0 E3           BNE   TTOUT3 IF Y IS NONZERO,
0383 AC 08 03        LDY   YSAVE  DO THE NEXT BIT
0386 28              PLP
0387 60              RTS          RETURN FROM DOCHAR
0388 A9 8D    RETURN LDAIM $008D
038A 20 5F 03        JSR   DOCHAR PRINT CARRIAGE RETURN
038D A9 00           LDAIM $0000
038F 4C 42 03        JMP   AUTOLF
0392 A9 F0    VIDINI LDAIM $00F0  POINT TO VIDEO DISPLAY ROUTINI
0394 85 36           STA   CSWL   LOW ORDER BYTE
0396 A9 FD           LDAIM $00FD
0398 85 37           STA   CSWH   HIGH ORDER BYTE
039A A9 28           LDAIM $0028
039C 85 21           STA   WNDWDT 40 COLUMN WINDOW WIDTH
039E A9 00           LDAIM $0000
03A0 85 24           STA   CH     SET HORIZONTAL CURSOR
03A2 60              RTS          TO 0 AND RETURN FROM VIDINIT
```

```
SYMBOL TABLE 2000 209C
AUTOLF 0342   CH     0024   COLCNT 0307   CSWH   0037
CSWL   0036   DLYQ   0373   DLYR   0376   DOCHAR 035F
FINISH 034F   MARK   C058   MARKOU 036E   PRNTIT 0336
RETURN 0388   RTSQ   0309   SETCH  035B   SPACE  C059
TESTCT 032E   TTINIT 030A   TTOUT  0321   TTOUTR 0323
TTOUTS 0366   TTOUTT 0371   VIDINI 0392   WAIT   FCA8
WNDWDT 0021   YSAVE  0308
```

# Extending the SYM-1 Monitor

**A program relocator, a program listing utility and a selective, extended trace routine illustrate how true monitor extensions can implement additional functions and commands.**

**Nicholas Vrtis**
**5863 Pinetree S.E.**
**Kentwood, MI 49508**

When Synertek wrote the monitor for the SYM-1, they left it open-ended by vectoring many of the major functions through a system RAM vector table. By changing the addresses in the vector table, it is relatively easy to implement additional functions and commands.

The three routines described in this article are almost permanently resident in my system. They have been coded as true monitor extensions in that they use only addresses already allocated to the monitor and could easily be put into ROM.

The programs are not complex or large, but that is also one of their good points. I have them sitting up in high memory where they are out of the way but available when needed.

The first program is a modified version of one that appears in *The First Book of KIM.* It is a program relocator that adjusts all the branches, jumps, and absolute address locations in a program so that you can relocate it. It is really the next best thing to a relocating loader.

The second routine is a little program lister that prints your program, putting one instruction on each line. This is easier to read and check than the standard Verify or Paper tape formats.

Finally, there is an extended trace routine that displays the values of all the registers, and additionally allows you to specify that only a portion of your program is to be traced. Did you ever wonder what was happening to the registers when one of your subroutines is executed only five times in a two thousand repetition loop? This utility lets you determine just that. There is a price that is paid, but I will get to that later.

If you have looked at the program code yet, you may have wondered at the unusual address. After all, who ever puts an extension in low memory? When I decided to write this article, I intended to use addres $C00, where I have it on my system, but then I decided to change it to low memory.

Almost everyone has scratch memory there to work on a program. After you enter it, check the memory dump, and run a few tests; you can use the program to relocate itself!

Actually, what you have to do is block move the program to the desired adress and use the new U0 command to perform the relocation on the new copy. Tell it the correct FROM and TO address, but make the program starting address the new location. There are three locations that must be changed manually, and you are all set up.

Before I go into a discussion about the programs, I would like to mention the interfaces to the SYM monitor that are used, and a few that aren't but are sort of handy anyway. The programs themselves are not complicated, and I try to keep them pretty well commented.

The SYM manual contains a small example showing how to add a command to the monitor, but isn't really clear about how it works. For one thing, the monitor uses the unrecognized command vector for more than just the U0 through U7 user commands. It does a jump via this vector whenever it encounters a command it cannot process, or a character that is non-hex.

```
MICRO-WARE ASSEMBLER 65XX-1.0 PAGE 01


                   ****************************************************
                   * SYM-1 USER MONITOR FUNCTION EXTENSIONS           *
                   *      MODIFIED 7/3/79 BY MICRO STAFF              *
                   * U0 - RELOCATE PROGRAM                            *
                   *      P1 = FROM ADDRESS                           *
                   *      P2 = TO ADDRESS                             *
                   *      P3 = START OF PROGRAM                       *
                   * U1 - MINI-PROGRAM LISTER                         *
                   *      P1 = PROGRAM STARTING ADDRESS               *
                   *      P2 = PROGRAM ENDING ADDRESS                 *
                   * ---- USER TRACE ROUTINE       Y-X-A-FLAGS-STACK  *
                   *      A626 = INCLUSIVE TRACE STARTING ADDRESS     *
                   *      A62C = EXCLUSIVE TRACE ENDING ADDRESS       *
                   *                                                  *
                   * SYM COMMAND 'E 200' WILL SET UP VARIOUS ADDRESSES *
                   * AND VALUES FOR THESE EXTENSIONS                  *
                   ****************************************************
0200                      ORG   $0200
0200 53            INITCO =     $53     STORE "SD" USER ROUTINE VECTOR
0201 44                   =     $44
                   ****************************************************
                   * CHANGE THE FOLLOWING WHEN RELOCATING THE PROGRAM *
                   ****************************************************
0202 32                   =     $32     STORE "22C" AND CHANGE
0203 32                   =     $32     IF ADDRESS CHANGES
0204 43                   =     $43
0205 2C                   =     $2C     STORE ",A66D"
0206 41                   =     $41
0207 36                   =     $36
0208 36                   =     $36
0209 44                   =     $44
```

```
020A 0D                 =       $0D
020B 4D                 =       $4D     STORE "MA658" AND CHANGE
020C 41                 =       $41     MAX RECORD
020D 36                 =       $36     TO BE
020E 35                 =       $35     TWENTY-FOUR
020F 38                 =       $38     BYTES LONG
0210 0D                 =       $0D
0211 31                 =       $31     STORE "18"
0212 38                 =       $38
0213 0D                 =       $0D
0214 53                 =       $53     SET TRACE VECTOR
0215 44                 =       $44
0216 38                 =       $38     STORING STRING "SD80C0,A67A"
0217 30                 =       $30
0218 43                 =       $43
0219 30                 =       $30
021A 2C                 =       $2C
021B 41                 =       $41
021C 36                 =       $36
021D 37                 =       $37
021E 41                 =       $41
021F 0D                 =       $0D
0220 53                 =       $53     STORE "SD"
0221 44                 =       $44
             ****************************************************
             * CHANGE THE FOLLOWING WHEN RELOCATING THE PROGRAM   *
             ****************************************************
0222 33                 =       $33     STORE "341" AND CHANGE IF ADDRESS CHANGES
0223 34                 =       $34
0224 31                 =       $31
0225 2C                 =       $2C     STORE ",A674"
0226 41                 =       $41
0227 36                 =       $36
0228 37                 =       $37
0229 34                 =       $34
022A 0D                 =       $0D
022B 00                 =       $00     ZERO IS END OF EXEC REQUEST
             ****************************************************
             * PAGE ZERO ADDRESS LOCATIONS                        *
             ****************************************************

022C         CURAD  *       $00FE  SYM-1 "OLD ADDRESS LOW ORDER
022C         CURADH *       $00FF  AND HIGH-ORDER
022C         ADJUST *       $00FC  SYM-1 PAGE ZERO SCRATCH AREA LOW-ORDER
022C         ADJUSH *       $00FD  AND HIGH ORDER
             ****************************************************
             * BY JIM BUTTERFIELD (SEE "THE FIRST BOOK OF KIM")   *
             * MODIFIED BY N. VRTIS TO RUN AS MONITOR             *
             *      EXTENSIONS ON THE SYM-1                       *
             *                                                    *
             * THIS PROGRAM ADJUSTS ABSOLUTE AND RELATIVE         *
             * ADDRESSES OF A PROGRAM SO IT CAN BE RELOCATED      *
             * OR EXPANDED                                        *
             * >>>>> NOTES:                                       *
             *   1- PAGE ZERO REFERENCES ABOVE $8000 WILL NOT     *
             *       BE CHANGED UNLESS SPECIFIED AS ABSOLUTE      *
             *       THREE-BYTE INSTRUCTIONS                      *
             *   2- ANY REFERENCES ABOVE $8000 WILL NOT BE        *
             *       CHANGED                                      *
             *   3- PROGRAM STOPS WHEN IT FINDS AN ILLEGAL        *
             *       OPERATION CODE (CAN USE $FF)                 *
             *   4- DON'T RELOCATE DATA                           *
             *                                                    *
             * INPUT PARMS:                                       *
             *       PARM1 - RELOCATE FROM ADDRESS                *
             *               (FIRST OPCODE THAT WILL MOVE)        *
             *       PARM2 - RELOCATE TO ADDRESS (WHERE PARM1     *
             *               WILL BE MOVED TO)                    *
             *       PARM3 - PROGRAM START ADDRESS (FIRST         *
             *               INSTRUCTION IN PROGRAM               *
             ****************************************************
022C CD 57 A6           CMP   LSTCOM SEE IF COMMAND TERMINATED PROPERLY
022F F0 02              BEQ   U0     YES -- SEE WHICH COMMAND
0231 38          COMERR SEC          ELSE SET CARRY AS ERROR FLAG
0232 60                 RTS          AND RETURN TO MONITOR FOR ER XX

0233 C9 14       U0     CMPIM $14    MAKE SURE IT IS "U0"
0235 F0 03              BEQ   U0COMM BRANCH IF IT IS
```

This means that it gets used for a lot of junk in addition to the defined user commands. It also means that you can use characters other than Un as command extensions, if you want, as long as they are not used for valid SYM commands with the same number of parameters.

The monitor saves the command value in a location called LSTCOM. When a carriage return is entered, the monitor reloads the command into the A register and loads the number of parameters into X.

So, the first thing our monitor extension should do is check the character in A against the value in LSTCOM. If they are the same, the program was called after normal command termination. If they are different, the command was not terminated properly and we want to make sure the carry is set and return with an RTS instruction.

This will cause the monitor to print the standard "ER xx" message and return to command mode.

Once we know that the command was terminated properly, we have to determine which command it was. As I mentioned earlier, the monitor does not verify the command character as it is entered, so we could be here for anything, including a "valid" command with the wrong number of parameters.

Finally, if we are on the right command, and if it was terminated properly, the last check is to make sure that exactly the correct number of parameters has been entered. If not, there will be missing information, or information will be in the wrong place. For any errors, all the extension has to do is guarantee that the carry is set and return to the monitor with an RTS instruction.

As an aside, the command processor does not initialize the stack register, and so, if you are debugging an extension and stop it before the RTS to the monitor, you can quickly use up a lot of the stack area. This only hurts if you have a routine or two located there, as I usually do.

The manual claims that locations $F8 through $FF are reserved for monitor use. Did you ever wonder what they are used for? Unfortunately, these locations were not assigned a variable name in the monitor assembly, so there are no cross references to them in the listing. I have tracked down most of the applications, but I don't guarantee that I didn't miss one.

The most used locations are probably $FE and $FF. These are the locations that the monitor uses for almost all of it's indirect addressing. If you look at the command descriptions, this is where the "OLD" address is kept.

These programs use it in the same manner that the monitor does. It's impossible to display these locations via the monitor commands directly, but doing a Verify or Memory will show you what they are pointing to. Also, if you plan to use them, none of the monitor routines will change them, but almost any command will.

Another important pair of locations is $FA and $FB. These contain the address of the next byte to be obtained as input when processing in the execute mode. If your program modifies these locations, it can't be invoked from the execute mode.

As another aside about the execute mode, all input comes from RAM, so if you do a JSR INCHR and expect to get keyboard input while in execute mode it won't work. The execute command is the only one that modifies these addresses. The other locations are pretty much scratch locations; you can probably use them without affecting command operation, but I would not count on them being the same after any call to monitor service routines.

The cassette routines use $FC and $FD, as does the block move command. Terminal input uses $F8 as a character buildup area, and terminal output uses $F9 to hold the character as it is being output. There may be a few other uses, but I would stay away from these unless you are really desperate for page zero space, or you are writing monitor extensions.

The System RAM areas are much better documented in the monitor listing. They have also been assigned names, and therefore appear on the assembly cross reference list. These programs only deal with two main areas. This is $A630 through $A63F, and they are monitor scratch areas. The two bytes used here are not used by the monitor, according to the cross reference lists.

The locations $A64A through $A64F are the addresses where the monitor collects input parameters. Each is a two byte parameter area, and all three areas are initialized to zero at the start of command processing. The problems begin when you find that the labels P1, P2 and P3 are a little misleading. The monitor starts collecting parameters in the P3 area, and rotates the whole area 16 bits left for each new parameter. It works out all right for three parameters, but two parameters will end up in P3 and P2, while one ends up in P3.

```
0237 4C DE 02          JMP    U1     GO TRY AS U1 COMMAND
023A E0 03      UOCOMM CPXIM $03     MAKE SURE HAVE THREE PARMS
023C D0 F3             BNE    COMERR BRANCH FOR ERROR IF NOT

                * NOW COMPUTE THE ADJUSTMENT INCREMENT

023E 38                SEC           SET BORROW
023F AD 4C A6          LDA    P2L    GET LOW-ORDER "TO"
0242 ED 4E A6          SBC    P1L    CALC DIFFERENCE
0245 85 FC             STA    ADJUST SAVE IN PAGE ZERO LOW-ORDER
0247 AD 4D A6          LDA    P2H    SAME FOR HIGH-ORDER
024A ED 4F A6          SBC    P1H
024D 85 FD             STA    ADJUSH IT GOES INTO PAGE ZERO ALSO

                * NOW PUT PROGRAM POINTER TO PAGE ZERO

024F 20 A7 82          JSR    P3SCR
                ******************************************************
                * GET AN OPCODE HERE                                 *
                ******************************************************

0252 20 24 03   GETOP  JSR    DETLEN FIND OPCODE LENGTH AND TYPE
0255 30 07             BMI    TRIPLE MINUS IS LENGTH 3 OR BAD TYPE
0257 F0 2A             BEQ    BRANCH ZERO IS A BRANCH

                ******************************************************
                * HERE WE HAVE TO SKIP FORWARD TO NEXT OPCODE        *
                ******************************************************

0259 20 1A 03   SKIP1  JSR    ADVANC
025C F0 F4             BEQ    GETOP  AND THEN GO GET THE NEXT OPCODE

                ******************************************************
                * GOT A 3 BYTE OPCODE / ILLEGAL / OR END (SPECIAL)   *
                ******************************************************

025E C8         TRIPLE INY           BUMP Y BY ONE
025F F0 0F             BEQ    FIX3BY IF NOW ZERO IT IS A 3 BYTER

0261 20 15 83   QUITDO JSR    CRLFSZ OUTPUT LAST ADDRESS
0264 20 42 83          JSR    SPACE  FOLLOWED BY A SPACE
0267 A0 00             LDYIM $00     AND THE OPCODE
0269 B1 FE             LDAIY CURAD
026B 20 FA 82          JSR    OUTBYT
026E 18                CLC           CLEAR THE CARRY
026F 60                RTS           AND RETURN TO SYSTEM

0270 C8         FIX3BY INY           MAKE Y=1 NOW
0271 B1 FE             LDAIY CURAD   LOW-ORDER PART OF ADDRESS
0273 AA                TAX           PUT INTO X
0274 C8                INY           NOW MAKE Y=2
0275 B1 FE             LDAIY CURAD   HIGH-ORDER PART OF ADDRESS
0277 20 B6 02          JSR    ADJST  GO CHANGE ADDRESS IF NECESSARY
ID=02

027A 91 FE             STAIY CURAD   PUT HIGH-ORDER BACK
027C 88                DEY           MAKE Y=1
027D 8A                TXA           LOW-ORDER TO A
027E 91 FE             STAIY CURAD   PUT IT BACK ALSO
0280 4C 59 02          JMP    SKIP1  GO SKIP FORWARD TO NEXT OPCODE

                ******************************************************
                * GOT A BRANCH - HAVE TO CHECK
                * BOTH "TO " AND "FROM" ADDRESSES
                ******************************************************

0283 C8         BRANCH INY           MAKE Y=1
0284 A6 FE             LDX    CURAD  GET CURRENT LOCATION LOW-ORDER
0286 A5 FF             LDA    CURADH AND HIGH-ORDER
0288 20 B6 02          JSR    ADJST  FIX IT IF NECESSARY
028B 8E 30 A6          STX    SCR0   SAVE LOW-ORDER FOR NOW
028E A2 FF             LDXIM $FF     SET FLAG FOR BACK REFERENCE
0290 B1 FE             LDAIY CURAD   GET RELATIVE BRANCH AMOUNT
0292 18                CLC
0293 69 02             ADCIM $02     ADJUST THE OFFSET
0295 30 01             BMI    OVER   BRANCH IF BACKWARDS BRANCH
```

```
0297 E8                 INX           FORWARDS - MAKE FLAG ZERO
0298 8E 31 A6  OVER     STX    SCR1   SAVE THIS ALSO
029B 18                 CLC
029C 65 FE              ADC    CURAD  CALCULATE "TO" LOW-ORDER
029E AA                 TAX           PUT INTO X
029F AD 31 A6           LDA    SCR1   00 OR FF, REMEMBER?
02A2 65 FF              ADC    CURADH CALCULATE "TO" HIGH-ORDER
02A4 20 B6 02           JSR    ADJST  FIX IT IF NECESSARY
02A7 CA                 DEX           TAKE BACK OFFSET
02A8 CA                 DEX
02A9 8A                 TXA           PUT LOW-ORDER BACK INTO A
02AA 38                 SEC           RE-CALCULATE RELATIVE BRANCH
02AB ED 30 A6           SBC    SCR0
02AE 91 FE              STAIY CURAD   AND PUT IT BACK
02B0 20 CE 02           JSR    SIGNCH GO CHECK FOR SIGN CHANGE
02B3 4C 59 02           JMP    SKIP1  GO SKIP FORWARD TO NEXT OPCODE
               ****************************************************
               * EXAMINE ADDRESS AND ADJUST IT IF NEEDED
               * HIGH-ORDER IS IN A
               * LOW-ORDER IS IN X
               ****************************************************
02B6 C9 80     ADJST    CMPIM $80     MAKE SURE REFERENCE NOT TOO FAR
02B8 B0 13              BCS    OUT    DONE IF TOO HIGH
02BA CD 4F A6           CMP    P1H    CHECK HIGH-ORDER FIRST
02BD D0 03              BNE    TEST2  BRANCH IF NOT EQUAL
02BF EC 4E A6           CPX    P1L    EQUAL - NEED TO CHECK LOW-ORDER ALSO
02C2 90 09     TEST2    BCC    OUT    BRANCH IF LOW
02C4 48                 PHA           ELSE SAVE HIGH-ORDER ON STACK
02C5 8A                 TXA           PUT LOW-ORDER INTO A
02C6 18                 CLC
02C7 65 FC              ADC    ADJUST ADD LOW-ORDER ADJUSTMENT
02C9 AA                 TAX           PUT BACK INTO X
02CA 68                 PLA           PULL HIGH-ORDER BACK OUT
02CB 65 FD              ADC    ADJUSH ADD IN HIGH ORDER ADJUSTMENT
02CD 60        OUT      RTS           AND RETURN
               ****************************************************
               * CHECK TO MAKE SURE SIGN
               * BEFORE BRANCH IS SAME AS AFTER
               ****************************************************
02CE 4D 31 A6  SIGNCH   EOR    SCR1   SEE IF SIGNS ARE THE SAME
02D1 10 0A              BPL    SIGNOK BRANCH IF THE SAME
02D3 48                 PHA           SAVE "A" ON STACK
02D4 20 16 83           JSR    CRLFSZ OUTPUT CURRENT ADDRESS
02D7 20 42 83           JSR    SPACE  AND A SPACE
02DA 4C 77 81           JMP    ERNOCR AND ERROR MESSAGE
02DD 60        SIGNOK   RTS           RETURN IF SIGN IS OK
               ****************************************************
               * SYM-1 FUNCTION - MINI LISTER
               * BY: NICK VRTIS -- LSI/CCSD -- APRIL 1979
               *
               * LIST A PROGRAM BY INSTRUCTION PER LINE
               *
               * INPUT PARMS:
               *        PARM1 - PROGRAM STARTING ADDRESS
               *        PARM2 - PROGRAM ENDING ADDRESS
               ****************************************************
02DE C9 15     U1       CMPIM $15     MAKE SURE ON RIGHT COMMAND
02E0 D0 04              BNE    U1ERR  BRANCH IF WRONG
02E2 E0 02              CPXIM $02     MAKE SURE 2 AND ONLY 2 PARMS GIVEN
02E4 F0 02              BEQ    U1STRT BRANCH TO START IF CORRECT
02E6 38        U1ERR    SEC
02E7 60                 RTS
02E8 20 9C 82  U1STRT   JSR    P2SCR  SET UP BEGINNING ADDRESS
               ****************************************************
               * LIST PROGRAM EITHER 1 AT A TIME OR "MAXRC" AT A TIME
               ****************************************************
02EB AD 58 A6  LISTER   LDA    MAXRC  # OF LINES CONTROLLED BY "MAXRC"
02EE 8D 31 A6           STA    COUNT  SAVE IN SCRATCH AREA
02F1 20 16 83  LISTLP   JSR    CRLFSZ PUT OUT CURRENT ADDRESS
```

The addresses I used for the high and low trace limits are entries in the jump table. I picked these for two reasons. The first is that I don't use the jump table, so am not worried about changing it. The second is slightly more important. If you will note, the default values set in these locations during system reset turn out to cover normal user RAM. This means I don't have to worry about making sure they get set every time I reset the system.

There are a number of obscure SYM monitor routines used here, and some explanation of their function is in order now. Where possible, the names correspond to names in the monitor listing.

The routine P3SCR takes the two bytes from the P3 area and moves them to page zero locations $FE and $FF for indirect addressing. P2SCR does the same thing, but with the P2 data instead of P3. To my knowledge, there is no P1SCR or equivalent.

CRLFSZ is a very handy routine that outputs a carriage return, a line feed, and the contents of $FF and $FE (i.e. the current address). The routine INCCMP does a 16 bit add of 1 to the contents of CURAD, and compares the result to the value of P3. The compare is ignored in the relocate program; but for the lister, P3 has the program ending address so it knows when to quit. There is a reverse of this routine, called DECCMP, that subtracts 1 and does the compare. It isn't used in these routines, but might be handy some time.

There are two other SYM monitor locations used which are not labeled monitor addresses. The ERNOCRLF label is a few instructions into the ERMSG routine. It is after the carriage return and line feed subroutine jump. Unfortunately, where I enter, ERMSG has already pushed A on the stack, so always JMP to it from a subroutine and let it do the return from your subroutine, or else your stack will get out of sync.

The last address I call DBRTN. I use it in the extended trace. It is actually the last couple of instructions of the normal trace routine. It does a check of the carry and continues tracing if the carry is clear; otherwise it returns to the monitor. This works out conveniently since the routines INSTAT and DELAY return with the carry set if a key is down or the break key on the terminal has been pressed.

The remaining addresses and routines used in the programs are defined adequately in the SYM manual, so I won't bother discussing them here.

The relocate program should not be difficult to follow. The program is made possible by the subroutine DETLEN. I have to give credit to Jim Butterfield and *The First Book of KIM* for that routine and for most of the relocate program. DETLEN not only determines the instruction length, but also classifies it as one of four types: a branch (Y=0) an absolute address reference (Y=FF) an "invalid" instruction (Y=FE) and all others (Y=number of bytes in the instruction).

The invalid opcodes detected are only those with bits 0 and 1 on. This is not all-inclusive, but it does cover quite a few of the undefined opcodes. The normal procedure for operating the program is to insert an FF after the last program statement, since the relocate program stops when it encounters an "invalid" opcode.

This sometimes catches an attempt to relocate a data area instead of a program, which is a definite no-no. The program can't tell the difference between most data and instructions, so make sure you stop it before it tries to "fix" the "addresses" in your data. If you get into the habit of collecting your data areas in one place, your programs will be easier to relocate.

If you follow the code, you will see that there is a lot more work involved in relocating a branch instruction than in fixing an absolute address reference. This is because the program has to compute the effective FROM and TO addresses before it can determine whether the relative byte count has changed.

I have also included a routine to verify that the sign (bit 7) of the new displacement is the same before and after the relocation. This routine was added shortly after the first time I relocated a backward branch into a forward branch, by overflowing the sign, and started executing one of the 6502's INMI instructions (INMI = Ignore Non-Maskable Interrupt).

The program lister was really easy to do with subroutine DETLEN available. I have a CRT running at 1200 baud, so I set the program up to list a screenfull of lines at a time, and then wait for any key before continuing with the listing. If you have a printer, or run at a slower baud rate, you might want to ignore the MAXRC count, do a call to INSTAT after each line, and only stop when the break key is entered. Remember, INSTAT returns with the carry set if the break is entered, and clear otherwise.

The extended trace routine is probably the hardest to understand. It also requires one hardware change as outlined in the SYM manual. That change is the installation of jumpers W-24 and X-25. These enable software control of the debug flip-flops, but only up to a certain point.

```
02F4 20 42 83  CUROP  JSR   SPACE  LEADING SPACE
02F7 20 24 03         JSR   DETLEN MAKE SURE GOT CURRENT LINE LENGTH
02FA A0 00            LDYIM $00    INIT Y TO ZERO

02FC B1 FE     CURRLP LDAIY CURAD  GET CURRENT OPCODE
02FE 20 FA 82         JSR   OUTBYT OUTPUT IT
0301 C8               INY          BUMP TO NEXT BYTE
0302 CC 32 A6         CPY   BYTES  SEE IF DONE
0305 D0 F5            BNE   CURRLP LOOP FOR CURRENT NUMBER OF BYTES

0307 20 1A 03         JSR   ADVANC ADVANCE TO NEXT INSTRUCTION
030A B0 0C            BCS   PGMDON SEE IF TO END
030C CE 31 A6         DEC   COUNT  ELSE DECREASE LINE COUNT
030F 10 E0            BPL   LISTLP GOT MORE TO DO IF POSITIVE

0311 20 1B 8A         JSR   INCHR  WAIT FOR ANY CHARACTER
0314 F0 02            BEQ   PGMDON EQUAL MEANS C/R AND HE WANTS QUITS
0316 D0 D3            BNE   LISTER ELSE CARRY ON

                ****************************************************
                * END OF PROGRAM ENCOUNTERED - RETURN TO MONITOR
                ****************************************************

0318 18        PGMDON CLC          CLEAR CARRY FOR OK RETURN
0319 60               RTS          AND RETURN

                ****************************************************
                * ADVANCE TO NEXT INSTRUCTION
                ****************************************************

031A AE 32 A6  ADVANC LDX   BYTES  GET BYTE COUNT
031D 20 B2 82  ADVILP JSR   INCCMP BUMP CURRENT ADDRESS
0320 CA               DEX          DECREASE COUNT
0321 D0 FA            BNE   ADVILP LOOP UNTIL ALL BYTES ARE COUNTED
0323 60               RTS          RETURN HERE

                ****************************************************
                * DETERMINE THE INSTRUCTION LENGTH
                ****************************************************

0324 A0 00     DETLEN LDYIM $00    INIT Y TO ZERO
0326 B1 FE            LDAIY CURAD  PICK UP CURRENT OPCODE

                * ENTER HERE IF "A" ALREADY HAS OPCODE IN IT

0328 A8        DETLN1 TAY          SAVE IN Y
0329 A2 07            LDXIM $07    GOT SEVEN TABLE ENTRIES TO CHECK

032B 98        CHKLOP TYA          PUT OPCODE BACK INTO A
032C 3D 82 03         ANDX  TABOUT -01 REMOVE THE DON'T CARE BITS
032F 5D 89 03         EORX  TABTST -01 TEST THE REST
0332 F0 03            BEQ   FOUND  BRANCH IF FOUND THE MATCH
0334 CA               DEX          ELSE TRY NEXT ENTRY
0335 D0 F4            BNE   CHKLOP UNTIL ALL ARE LOOKED AT

0337 BC 99 03  FOUND  LDYX  TABLEN GET LENGTH FROM TABLE
033A 8C 32 A6         STY   BYTES  SAVE THE LENGTH
033D BC 91 03         LDYX  TABTYP NOW LOAD THE OPCODE TYPE
0340 60               RTS          AND RETURN


ID=03


                ****************************************************
                * ALTERNATE USER TRACE ROUTINE
                *
                * BY: NICK VRTIS -- LSI/CCSD      FEBRUARY 1979
                *
                * ALTERNATE TRACE ROUTINE TO PRINT ADDITIONAL DATA
                *
                * WILL PRINT PROGRAM COUNTER-Y-X-A-FLAGS-STACK
                * ONLY PRINTS FOR PROGRAM ADDRESS IN RANGE OF ADDRESS
                *      SPECIFIED BY:
                *          A62C - EXCLUSIVE ENDING ADDRESS
                *                 (SYM DEFAULT IS 0000)
```

```
                        *         A626 - INCLUSIVE STARTING ADDRESS
                        *                (SYM DEFAULT IS 0000)
                        * TRACE VELOCITY IS IGNORED IF TRACE IS NOT IN RANGE
                        * KEYBOARD IS CHECKED AND RETURN
                        * IS TO MONITOR IF KEY OR BREAK
                        * REGARDLESS OF ADDRESS
                        ****************************************************

0341 AE 59 A6  USRTRA LDX   USREGS ALWAYS EXECUTES SO X IS OK
0344 AD 5A A6         LDA   USREGS +01 A WILL BE OK IF SELF TRACING

                        ****************************************************
                        * CHANGE THE FOLLOWING INSTRUCTION
                        * TO HIGH-ORDER OF PAGE LOCATED ON
                        ****************************************************

0347 C9 03            CMPIM $03    SEE IF TRACING MYSELF
0349 F0 35            BEQ   RETURN
034B CD 2D A6         CMP   THIGH  +01
034E D0 03            BNE   HI
0350 EC 2C A6         CPX   THIGH
0353 B0 28     HI     BCS   NOTRAN BRANCH IF TOO HIGH

                        ****************************************************
                        * IT IS LESS THAN THE UPPER LIMIT
                        ****************************************************

0355 CD 27 A6         CMP   TLOW   +01 CHECK AGAINST LOWER LIMIT
0358 D0 03            BNE   LO
035A EC 26 A6         CPX   TLOW
035D 90 1E     LO     BCC   NOTRAN BRANCH IF NOT IN RANGE

                        * IT IS IN RANGE - OUTPUT GOODIES

035F 20 4D 83         JSR   CRLF   START ON NEW LINE
0362 20 EE 82         JSR   OUTPC
0365 A2 05            LDXIM $05
0367 BD 5A A6  DSPREG LDAX  USREGS +01
036A 20 42 83         JSR   SPACE  OUTPUT LEADING SPACE
036D 20 FA 82         JSR   OUTBYT NOW THE DATA AS 2 HEX
0370 CA               DEX
0371 D0 F4            BNE   DSPREG
0373 EC 56 A6         CPX   TV     COMPARE 0 TO TV
0376 F0 08            BEQ   RETURN EQUAL WILL ALSO HAVE CARRY SET

                        * PERFORM THE DELAY ACCORDING TO TV VALUE

0378 20 5A 83  DODELA JSR   DELAY
037B B0 03            BCS   RETURN IF KEY WAS DOWN - DON'T CHECK AGAIN

                        * NOT IN RANGE - CHECK FOR KEY DOWN ANYWAY

037D 20 86 83  NOTRAN JSR   INSTAT CHECK FOR KEY DOWN

                        * RETURN WITH CARRY ON FOR RETURN TO MONITOR
                        * CARRY OFF TO CONTINUE TRACE

0380 4C BB 80  RETURN JMP   DBRTN  RETURN WILL CHECK CARRY

                        ****************************************************
                        * TABLES FOR DETLIN
                        ****************************************************

0383 0C        TABOUT =     $0C    MASKS TO REMOVE DON'T CARE BITS
0384 1F               =     $1F
0385 0D               =     $0D
0386 87               =     $87
0387 1F               =     $1F
0388 FF               =     $FF
0389 03               =     $03
038A 0C        TABTST =     $0C
038B 19               =     $19
038C 08               =     $08
038D 00               =     $00
038E 10               =     $10
038F 20               =     $20
0390 03               =     $03
0391 02        TABTYP =     $02
```

When I started writing this routine, it was only going to be a one night project. It turned out to be a project all right, but it was more than one night. In the mean time, I found the program bug that caused me to write the extended trace in the first place. It has been useful on a number of later projects, though.

Let me tell you some things about the SYM implementation of hardware debug. It all starts with a non-maskable interrupt which is generated at the completion of each instruction that is not a SYM monitor address, provided that the debug flip-flop is set. The 6502 picks up the address contained in locations $FFFA and $FFFB as the interrupt handler. Do to wiring "mirrors", $FFFA and $FFFB are actually $A67A and $A67B, which are system RAM addresses.

Normally, this vector contains the address of SVNMI, which is the usual trace routine. The first thing the monitor does is unprotect system RAM, and then save all the registers, flags, and program counter in the user register save area in system RAM. It then resets the debug flip-flop so that it is off. For the extended trace, this vector is changed to point to another SYM monitor routine that does the same things, but exits via an indirect jump through system RAM location TRCVEC to the user trace routine.

In theory, this means that the user routine should be able to do just about anything the monitor can do. The hard facts of life are that the debug key bounces, and the monitor does not debounce it before you get control, but it does reset the flip-flop.

This is no problem if I am in the monitor (say, waiting for input) when I press the debug key. Since the monitor does not get interrupted, by the time an interrupt is generated, the key is through bouncing, and only the interrupt is generated.

If, on the other hand, a user program is executing and I press the debug key, the extended trace routine get control before the key has finished bouncing. This means that an interrupt is generated within the extended trace and it starts tracing itself.

At first glance, the solution would seem the same as for any other bouncy input; namely, to wait for it to settle. The only problem is that the extended trace gets only ONE instruction done before the routine is interrupted. The best that I could do was check to see if it is tracing itself and exit gracefully to the monitor if so. Unfortunately, the register save area doesn't contain any more useful information, but then, there is a price for everything.

Now that we have that explanation out of the way, on to a discussion of the mechanics of the trace routine. Actually, the hardest part is making sure the carry gets set or cleared, before returning to DBRTN, so we either continue tracing or exit to the monitor. If the program is tracing itself, or if the trace velocity is zero, the return is executed immediately after a compare instruction that resulted in an equal condition which sets the carry.

If the trace velocity was not zero, then this routine uses the DELAY routine to slow down the execution rate. DELAY even checks the keyboard, via INSTAT, for a break key and sets the carry appropriately. The check of the carry is made after the jump to DELAY so that the program doesn't check the keyboard twice. The second check would probably get the opposite results if the keypad were being checked, since KEYQ debounces the keypad.

You should also note that even if the address is not in the requested range, the program does a call to INSTAT, anyway, to check for a key down or the break key. This is so you can interrupt a program outside your requested trace range. Remember, the debug key is already causing the extended trace to be invoked, so you can't stop the program with that.

The final thing to remember about the trace routine is that even for those addresses you have not selected, there are an awful lot of instructions executed before that fact is determined. Effectively, your cycle time has slowed drastically when debug is on, and I mean by orders of magnitude. This can be surprising at times, especially when the code you are bypassing initializes a two thousand byte array.

Last but not least, I would like to explain the strange code that appears at the start of the program. It comprises the ASCII commands that set up the user command vector, the MAXRC byte count, and the extended trace routine addresses. By putting them there, I only have to remember one address instead of half of a dozen. By using the SYM execute command, all the addresses get set up for me.

Don't forget to change the addresses referenced in the execute commands when you relocate these routines. Also remember that the addresses must be in ASCII, not in hex. There is also one place in the extended trace routine that must be changed to equal the high order byte of the address the routine resides at. This is so the routine can tell if it is tracing itself. It also means the program won't trace any other program on that page.

```
0392 FF                    =      $FF
0393 FF                    =      $FF
0394 01                    =      $01
0395 01                    =      $01
0396 00                    =      $00
0397 FF                    =      $FF
0398 FE                    =      $FE
0399 02           TABLEN   =      $02
039A 03                    =      $03
039B 03                    =      $03
039C 01                    =      $01
039D 01                    =      $01
039E 02                    =      $02
039F 03                    =      $03
03A0 03           PGMEND   =      $03

                  *****************************************************
                  * SYM SYSTEM ROUTINE ENTRY POINTS AND RAM ADDRESSES
                  *****************************************************

03A1              DBRTN    *      $80BB  CHECK CARRY & TRACE OR MONITOR
03A1              ERNOCR   *      $8177  "ERXX" W/O CR/LF -- JUMP TO ONLY
03A1              P2SCR    *      $829C  PUT "PARM2" INTO "CURAD"
03A1              P3SCR    *      $82A7  PUT "PARM3" INTO "CURAD"
03A1              INCCMP   *      $82B2  BUMP "CURAD" & COMPARE TO PARM3
03A1              OUTPC    *      $82EE  OUTPUT USER PROGRAM COUNTER
03A1              OUTBYT   *      $82FA  PRINT A (TWO HEX DIGITS)
03A1              CRLFSZ   *      $8316  OUTPUT CR/LF AND "CURAD"
03A1              SPACE    *      $8342  OUTPUT ONE SPACE
03A1              CRLF     *      $834D  OUTPUT CR/LF
03A1              DELAY    *      $835A  DELAY ACCORDING TO TV
03A1              INSTAT   *      $8386  GET KEY STATUS (BREAK
                     OR ANY KEY DOWN)
03A1              INCHR    *      $8A1B  GET ASCII CHAR VIA "INVEC"

                  *****************************************************

03A1              TLOW     *      $A626  TRACE LOW ADDRESS
03A1              THIGH    *      $A62C  TRACE HIGH ADDRESS
03A1              SCRO     *      $A630  SYSTEM SCRATCH AREA 0
03A1              SCR1     *      $A631  SYSTEM RAM SCRATCH AREA 1
03A1              BYTES    *      $A632  SYSTEM RAM SCRATCH AREA 2
03A1              COUNT    *      SCR1   USE SCRATCH AREA 1
03A1              P3L      *      $A64A  INPUT PARAMETER VALUES
03A1              P3H      *      $A64B
03A1              P2L      *      $A64C
03A1              P2H      *      $A64D
03A1              P1L      *      $A64E
03A1              P1H      *      $A64F
03A1              ENDAD    *      P3L    ENDING ADDRESS IS IN P3 AREA

03A1              TV       *      $A656  TRACE VELOCITY
03A1              LSTCOM   *      $A657  COMMAND END INDICATOR
03A1              MAXRC    *      $A658  MAXIMUM RECORD/BYTES FOR OUTPUT
03A1              USREGS   *      $A659  TRACE HOLD OF USER REGISTERS
ID=
```

```
SYMBOL TABLE 2000 21C8
ADJST  02B6    ADJUSH 00FD    ADJUST 00FC    ADVANC 031A
ADVILP 031D    BRANCH 0283    BYTES  A632    CHKLOP 032B
COMERR 0231    COUNT  A631    CRLF   834D    CRLFSZ 8316
CURAD  00FE    CURADH 00FF    CUROP  02F4    CURRLP 02FC
DBRTN  80BB    DELAY  835A    DETLEN 0324    DETLNQ 0328
DODELA 0378    DSPREG 0367    ENDAD  A64A    ERNOCR 8177
FIXSBY 0270    FOUND  0337    GETOP  0252    HI     0353
INCCMP 82B2    INCHR  8A1B    INITCO 0200    INSTAT 8386
LISTER 02EB    LISTLP 02F1    LO     035D    LSTCOM A657
MAXRC  A658    NOTRAN 037D    OUTBYT 82FA    OUTPC  82EE
OUT    02CD    OVER   0298    PGMDON 0318    PGMEND 03A0
PQH    A64F    PQL    A64E    PRH    A64D    PRL    A64C
PRSCR  829C    PSH    A64B    PSL    A64A    PSSCR  82A7
QUITDO 0261    RETURN 0380    SCRP   A630    SCRQ   A631
SIGNCH 02CE    SIGNOK 02DD    SKIPQ  0259    SPACE  8342
TABLEN 0399    TABOUT 0383    TABTST 038A    TABTYP 0391
TESTR  02C2    THIGH  A62C    TLOW   A626    TRIPLE 025E
TV     A656    UP     0233    UPCOMM 023A    UQ     02DE
UQERR  02E6    UQSTRT 02E8    USREGS A659    USRTRA 0341
```

# Computer Forum

- BUSINESS
- EDUCATIONAL
- PERSONAL

14052 EAST FIRESTONE BOULEVARD · SANTA FE SPRINGS, CALIFORNIA 90670

(213) 921-2111 (714) 739-0711


## Apple II Software for the apple® computer

**We do not claim to have the largest selection of APPLE II software available, however, we do have a good selection and it is growing larger every week. This is a partial list of the software that we normally have in stock. To get a complete list of all our software, send for our new APPLE II software catalog that will be available next month. This catalog will give you descriptions of most of the software listed so that you can tell what the program will do before you buy. Due to the ease of copying software, we cannot give any refunds for any software. Should something be wrong with the media that the software is on, we will exchange it for a good one. Starting with this ad, we will pay all shipping and handling charges for orders over $25.00 in the continental United States. Orders under $25.00 add $2.50 for shipping & handling. Orders outside the U.S. add 10% or $10.00 whichever is more.**

| | | |
|---|---|---|
| 001-00015 | $20.00 | APPLE CHECKBOOK |
| 001-00021 | $15.00 | STARWARS/STARTREK |
| 001-00036 | $ 7.50 | COLOR DEMOS/BREAKOUT |
| 001-00042 | $20.00 | APPLESOFT II |
| 001-00057 | $ 7.50 | RAM TEST |
| 001-00063 | $ 7.50 | BLACKJACK/SLOT MACHINE |
| 001-00079 | $ 7.50 | BIORHYTHM/MASTERMIND |
| 001-00090 | $ 7.50 | DATA MOVER/TELEPONG |
| 001-00106 | $25.00 | DOW JONES |
| 001-03813 | $12.00 | BULLS AND BEARS |
| 001-03829 | $12.00 | WARLORDS |
| 001-03834 | $12.00 | MICROTRIVIA |
| 001-03840 | $12.00 | KIDSTUFF |
| 001-03855 | $12.00 | SPORTS TRIVIA |
| 001-03861 | $16.50 | TRANSACTIONAL ANALYSIS |
| 001-03877 | $16.50 | FINANCIAL ANALYSIS |
| 002-00112 | $14.95 | SWARMS |
| 002-00128 | $ 9.95 | APPLE DERBY |
| 002-00133 | $ 9.95 | APPLE CASINO |
| 002-00149 | $24.95 | FILE EDITOR |
| 002-00154 | $24.95 | ASSEMBLER |
| 002-00160 | $ 9.95 | FINANCIAL WIZARD-I |
| 002-00176 | $ 9.95 | FINANCIAL WIZARD-II |
| 002-00181 | $ 9.95 | FINANCIAL WIZARD-III |
| 002-00197 | $ 9.95 | FINANCIAL WIZARD IV |
| 002-00203 | $ 9.95 | CUBIK |
| 002-00219 | $14.95 | STATISTICS I |
| 002-00225 | $ 7.45 | MEMORY DUMP |
| 002-00230 | $ 9.95 | SPACE MAZE |
| 002-01473 | $ 9.95 | ROCKET PILOT |
| 002-01489 | $ 9.95 | SAUCER INVASION |
| 002-01495 | $ 9.95 | STAR WAR |
| 002-01571 | $19.95 | APPLE II ORGAN |
| 002-07123 | $ 9.95 | WAMPUS HUNT |
| 002-07138 | $14.95 | AUTOMOTIVE DIAGNOSIS |
| 002-07144 | $ 7.45 | PROGRAM UNLOAD |
| 002-07159 | $ 9.95 | BASIC STATISTICS |
| 002-07165 | $ 9.95 | ELECT. ENGINEERING |
| 002-07171 | $14.95 | FUNCTION GRAPHS |
| 002-07186 | $ 9.95 | VECTOR ANALYSIS |
| 002-07192 | $ 9.95 | RADAR INTERCEPTION |
| 004-00343 | $39.95 | INSTANT LIBRARY (TAPE) |
| 004-00337 | $39.95 | INSTANT LIBRARY (DISC) |
| 004-00358 | $ 9.95 | MUSIC KALEIDOSCOPE |
| 004-00364 | $ 9.95 | BOMBER |
| 004-00370 | $19.95 | INDEX-CARD FILE |
| 004-00385 | $39.95 | BEST-OF-BISHOP |
| 004-00391 | $12.95 | TALKING CALCULATOR |
| 004-00406 | $19.95 | APPLE-LIS'NER |
| 004-00428 | $15.95 | APPLETALKER |
| 004-00433 | $12.95 | STARWARS/SPACE MAZE |
| 004-00449 | $12.95 | ROCKET PILOT/SAUCER INVASION |
| 004-00454 | $19.95 | TIC-TAC-TALKER/SPECTRUM ANAL |
| 004-07214 | $19.95 | SCREEN MACHINE |
| 004-07220 | $14.95 | BACKGAMMON |
| 044-07235 | $14.95 | APPLE '21' |
| 005-00460 | $19.00 | PROGRAM CATALOG (DISC) |
| 005-00476 | $15.00 | NAME & ADDRESS (TAPE) |
| 005-10502 | $19.00 | NAME & ADDRESS (DISC) |
| 005-00481 | $14.00 | HANGMAN W/460 WORDS |
| 005-00497 | $ 6.95 | MENU ASSORTMENT |
| 005-03904 | $14.95 | MAGAZINE FILES |
| 009-00706 | $ 9.95 | PERSONAL EXP. RECORD |
| 009-00712 | $ 9.95 | BLACKJACK TUTOR |
| 009-00728 | $ 9.95 | FOREIGN LEGION/COMMANDO |
| 009-00733 | $ 9.95 | ROADRACE/SPACEWAR |
| 009-00749 | $ 9.95 | WORDSMITH/WAMPUS |
| 009-00754 | $11.95 | CODEBREAKER/SECRET WRITING |
| 016-01195 | $13.50 | PRESCHOOL IQ-BUILDER |
| 016-01238 | $13.50 | ANALOGIES |
| 016-01259 | $13.50 | VOCABULARY BLDR. I |
| 016-01314 | $13.50 | VOCABULARY BLDR. II |
| 016-01271 | $13.50 | NUMBER SERIES |
| 016-01356 | $39.95 | STEP BY STEP |
| 016-01399 | $13.50 | WORD MASTER/STORY BUILDER |
| 016-01410 | $13.50 | MEMORY BUILDER |
| 035-03626 | $ 6.95 | FOOL'S SPOOL |
| 035-03631 | $ 9.95 | CONEY ISLAND |
| 035-03647 | $15.95 | DEPTH CHARGE |
| 035-03668 | $ 9.95 | COUNTRY DRIVER |
| 035-03674 | $ 9.95 | KALEIDOSCOPE |
| 035-03689 | $ 9.95 | CROSS CHASE |
| 035-03695 | $ 6.95 | LEAP FROG |
| 035-03700 | $ 9.95 | CHASE |
| 035-03716 | $ 6.95 | ECHO |
| 035-03722 | $ 6.95 | SUB COMMAND |
| 035-03737 | $ 9.95 | FOOTBALL |
| 035-03743 | $ 9.95 | MATCHWITS |
| 035-03758 | $ 9.95 | LASER TURRET |
| 035-03764 | $ 9.95 | TAROT CARDS |
| 035-03770 | $15.95 | 3-D DOCKING |
| 035-03785 | $ 9.95 | GALACTIC BATTLE |
| 035-03791 | $ 6.95 | TIME CLOCK |
| 035-03807 | $15.95 | PHASOR ZAP |
| 035-06955 | $ 9.95 | FLYSWATTER |
| 035-06961 | $ 6.95 | QUARTERHORSE RACE |
| 035-06977 | $ 9.95 | GUNFIGHTER |
| 035-06982 | $ 9.95 | MAGIC SQUARES |
| 035-07004 | $15.95 | BATTLESTAR I |
| 035-07010 | $ 9.95 | PIRATES |
| 035-08087 | $ 9.95 | ACK ACK |
| 035-08233 | $15.95 | I CHING (TAPE) |
| 035-08249 | $19.95 | I CHING (DISC) |
| 035-08109 | $ 6.95 | KING |
| 035-08109 | $ 6.95 | KING |
| 035-08115 | $ 6.95 | SCRAMBLE |
| 035-06940 | $ 9.95 | SUB VIEW |
| 035-03652 | $ 9.95 | STAR DODGER |
| 035-06891 | $ 9.95 | PERPETUAL CALENDAR |
| 035-06907 | $15.95 | SUPER STARWARS |
| 035-06913 | $19.95 | INCOME TAX 1978 |
| 035-06929 | $ 9.95 | COLOR STARTREK |
| 035-06934 | $ 9.95 | SUB DETECT |
| 035-06998 | $19.95 | BUSINESS & FINANCE |
| 035-07031 | $19.95 | PROGRAMMA'S TALKING DISC |
| 035-07047 | $24.95 | 3-D ANIMATION DISC |
| 035-07052 | $19.95 | AUDIO ENGINEER DISC |
| 035-07068 | $19.95 | EDITOR ASSEMBLER DISC |
| 035-07074 | $19.95 | DATA BASE DISC |
| 035-07089 | $19.95 | PMAIL MAILING LABLES DISC |
| 035-07095 | $15.95 | STRATO LASER |
| 035-07101 | $29.95 | COMPUREAD DISC |
| 035-07117 | $19.95 | ANALYST DISC |
| 035-08093 | $ 6.95 | RETREAT |
| 035-08121 | $34.95 | APPLEFORTH 1.2 (TAPE) |
| 035-08136 | $49.95 | APPLEFORTH 1.2 (DISC) |
| 035-08142 | $19.95 | APPLE PIE 1.0 (TAPE) |
| 035-08157 | $24.95 | APPLE PIE 1.0 (DISC) |
| 035-08163 | $ 9.95 | HOME ACCOUNTING (TAPE) |
| 035-08179 | $15.95 | HOME ACCOUNTING (DISC) |
| 035-08206 | $15.95 | SUPER OTHELLO |
| 035-08212 | $ 9.95 | LUNARSCAPE |
| 035-08228 | $19.95 | FOOTBALL PREDICTIONS |
| 035-08276 | $19.95 | STATISTICS |
| 035-08281 | $ 9.95 | BIOTHYRHM |
| 035-08297 | $69.95 | ASM/65 EDITOR ASSEMBLER 1.0 |
| 033-03485 | $12.95 | TANK WAR |
| 033-03491 | $17.95 | U-DRAW |
| 033-03501 | $12.95 | MAZE GAME |
| 033-03513 | $12.95 | ESCAPE |
| 033-03529 | $17.95 | TEXT EDITOR |
| 033-03534 | $ 9.95 | NUMBER CRUNCHER |
| 033-03540 | $12.95 | MUSIC BOX |
| 033-03555 | $12.95 | SIDE SHOWS |
| 033-03561 | $17.95 | ELECTRIC CRAYON |
| 033-03577 | $ 9.95 | CHRISTMAS CAROLER |
| 033-03582 | $49.95 | DR. MEMORY |
| 042-04420 | $59.95 | TEXT EDITOR |
| 042-04435 | $39.95 | MAILING LIST (DISC) |
| 042-04441 | $29.95 | MEMO CALENDAR (DISC) |
| 042-04456 | $19.95 | SINGLE DISC COPY |
| 042-08763 | $24.95 | DANS DISC UTILITY |
| 042-08779 | $39.95 | DANS DISC UTIL/SNGL DISC CPY |
| 032-03368 | $14.95 | STIMULATING SIMULATIONS |
| 032-03389 | $14.95 | BRIDGE CHALLENGER |
| 032-03400 | $19.95 | MICROCHESS |
| 015-01152 | $19.95 | FASTGAMMON |

Computer Forum

- BUSINESS
- EDUCATIONAL
- PERSONAL

14052 East Firestone Blvd.
Santa Fe Springs, CA 90670
(213) 921-2111 • (714) 739-0711

NAME

ADDRESS

CITY STATE

ZIP PHONE

| DESCRIPTION | PRICE | QUANTITY | AMOUNT |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

*Yes, you may charge it!*
☐ B of A Card ☐ Mastercharge

Subtotal

Calif. Residents 6% tax

Shipping

Expiration Date ____________

Card # ____________

Signature ____________

TOTAL AMOUNT ENCLOSED

# Replace that PIA with a VIA

E. D. Morris, Jr.
3200 Washington Street
Midland, MI 48640

**Sound effects, timed interrupts and a versatile shift register are a few of the benefits offered by this useful hardware equipment.**

If your microcomputer board uses the 6520 Peripheral Interface Adapter for an I/O port, you might consider replacing it with a 6522 Versatile Interface Adapter. For the two dollars increase in price you get all the functions of the 6520 plus two timers, a shift register, input data latching, and a much more powerful interrupt system.

A block diagram of the VIA is shown in Figure 1. The 6522 appears to the CPU as sixteen memory locations, compared to four for the 6520. Table 1 shows how the various registers are addressed using the register select pins. In some cases, accessing a register triggers another function such as resetting an interrupt flag or starting the timer.

The timers are loaded with data and then decremented at the system clock rate to create a delay. This can be used to generate interrupts at preset intervals.

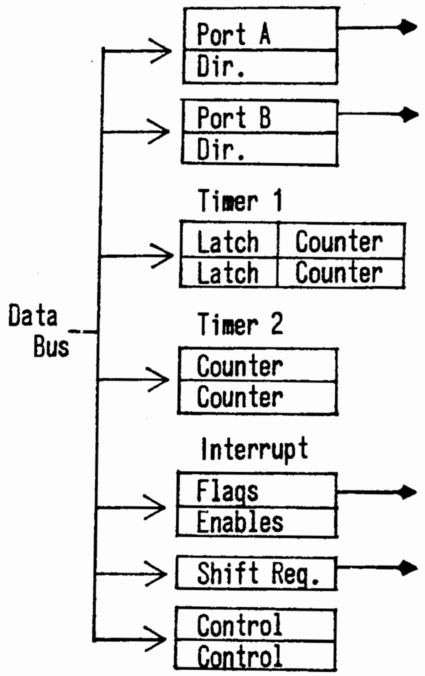



Figure 1: Block Diagram of the 6522

***Table 1: 6522 Register Address List***

| RS3 | RS2 | RS1 | RS0 | FUNCTION |
|---|---|---|---|---|
| L | L | L | L | I/O port B |
| L | L | L | H | I/O port A |
| L | L | H | L | Data direction B |
| L | L | H | H | Data direction A |
| L | H | L | L | Timer 1 counter low byte |
| L | H | L | H | Timer 1 counter high byte |
| L | H | H | L | Timer 1 latch low byte |
| L | H | H | H | Timer 1 latch high byte |
| H | L | L | L | Timer 2 low byte |
| H | L | L | H | Timer 2 high byte |
| H | L | H | L | Shift register |
| H | L | H | H | Timer and shift register control |
| H | H | L | L | I/O handshake control |
| H | H | L | H | Interrupt flags |
| H | H | H | L | Interrupt enables |
| H | H | H | H | I/O port A |

Another use is to connect an amplifier and speaker to the shift register output. By storing a 11110000 or 11001100 in the shift register and placing it in the free running mode, square waves at audio frequencies are produced. BASIC can then POKE constants to timer 2 to produce various audio tones. You can create electronic music, or add sound effects to those mute game programs. In fact, this scheme is used for the PET sound effects.

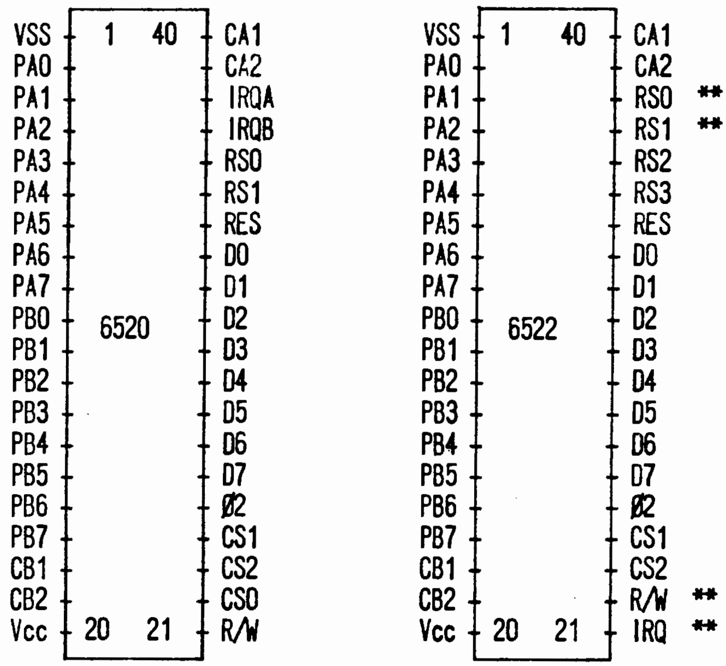
The timers can be set to cause interrupts at equally spaced time intervals. This saves the CPU the chore of keeping time or chasing its tail in loops to create delays. I found the timed interrupt very convenient in writing a single-step machine language debugging program. The timer is set so the CPU can just escape from the monitor and execute one step of the main program before another interrupt forces it back to the monitor. A recent issue of MICRO gives details of using the 6522 timers with a SYM computer.

So how do you install this super chip in your system? Figure 2 compares the pinouts of the 6520 and the 6522. Thirty-six of the forty pins are identical, so that is a good start. However changes must by made to your circuit board at pins 21, 22, 37 and 38. The 6522 needs 4 address lines compared to 2 for the 6520. I jumpered RS0 and RS1 to address lines 2 and 3 somewhere on the CPU board. To reduce foil cutting, I left RS2 and RS3 connected to address 0 and 1. You will have to make your own list of register addresses depending how you connect the RS lines to your address buss. IRQ and R/W must be re-jumped to the proper pins. My CPU board did not use CS0, so this was no loss.

I made this modification on an OSI 500 CPU board (Kilobaud March 1979). After reading the Trouble Shooter's Corner (Kilobaud September 1978), I was very apprehensive about taking on this project. However the OSI board has no "bogus" clock pulsus running around, so I had no trouble.

Any of seven events can cause as interrupt and set a flag in the interrupt flag register. The shift register rate is controlled either by timer 2 or by an external clock. Two control registers allow selection of the many options available in the 6522 VIA. More details of the 6522 can be obtained from Synertek, P.O. Box 552, Santa Clara CA 95052.

So what does the 6522 gain you as far as programming? Well, the shift register can be used as a serial output port to drive a Teletype or printer. The baud rate is software controlled by the constant stored in timer 2.




| | | |
|---|---|---|
| PA,PB | = | I/O Port |
| CA,CB | = | Handshake Control |
| RS | = | Register Select (Address) |
| RES | = | Reset |
| D | = | Data Bus |
| CS | = | Chip Select |
| IRQ | = | Interrupt |

***Figure 2: Pin-outs of the 6522 VIA and 6520 PIA***

# PET Cassette I/O

**Ronald C. Smith**
**P. O. Box 1125**
**Reseda, CA 91335**

**No more lost files, missing data or elusive end of file marks! Now that great cassette I/O capability can be put to work.**

At first glance it would appear that cassette data storage on the Commodore PET would be a snap. Upon trying it, you soon discover otherwise. Three major problems soon emerge to frustrate the uninitialed. The PET does not read back all of the data you wrote on the tape. It misses the end of file mark, causing the system to crash, and occasionally it even refuses to find a file which you have written.

The first two problems are related. An end of file mark is, after all, data, so if the PET is skipping data it could (and does!) just skip the end of file mark. Fixing the problem of skipping data will fix the problem of missing the end of file.

The PET writes data onto the cassette tape in blocks of 192 characters, including carriage returns. The cassette motor is turned off in between writing blocks. Before writing the next block the motor must be turned on, and time allowed for the tape to come up to its steady, proper speed. Apparently, when the PET operating system was written, the cassette decks came up to speed much faster than the cassette units supplied with production PETs.

Because of this, the pause (interblock gap) is insufficient. When the PET attempts to read the block back, data starts before the tape is up to speed, resulting in the first few bytes of the block being garbled. Unfortunately, those few bytes are what identify the block as data rather than noise. As a result, the block is ignored completely and the PET keeps searching until it comes to the next block. Of course, the tape is at its correct speed by now, so this block is read properly. The bottom line is that you lose every other block of data!

To solve this problem you need to funnel all of your output to tape through a subroutine. The subroutine counts how many characters have been written and placed into the tape buffer. When it detects that the 192nd character is about to be written, it should reset its counter to zero, start up the cassette motor, and pause 1/6 second before allowing the character to be written. To start cassette #1, POKE 59411,53. For cassette #2, it's POKE 59456,207.

Use of this subroutine will eliminate the problem of skipped blocks. It will also insure that the end of file mark is not missed.

The problem of unrecognized files is another operating system idiosyncrasy, fortunately much simpler to fix. According to Commodore, upon occasion the system will not properly initialize the tape buffer before opening a file. This causes the data to be placed in the wrong place in the memory or buffer. The system can't recognize the data when it opens for input because it just can't find it! The fix is simple. For tape unit #1, POKE 243,122; POKE 244,2 before opening the file. For tape unit #2, POKE 243,58; POKE 244,3 before opening. These POKEs initialize the pointers and eliminate the problem.

The subroutines shown illustrate one way to use the methods just described. Set PR or PR$ equal to the variable which you wish to print and jump to the approriate subroutine entry point. Do not forget to write an interblock gap before closing the file.

Please note that even though you have stored numbers as ASCII strings on the tape, this is what the PET does anyway! You can still read it as a number. This information should help you employ the great file handling capabilities built into your PET.

```
100 REM PRINT NUMERIC
110 PR$ = STR$(PR)
120 REM PRINT STRING
130 LN% = LN% + LEN(PR$) + 1
140 IF LN% = 191 THEN LN% = 0 : GOSUB 180
150 PRINT#1,PR$
160 RETURN
170 REM INTERBLOCK GAP
180 DT = TI
190 POKE 59411,53 : IF DT + 10 = TI GOTO 190
200 RETURN
```

# Tokens

E. D. Morris Jr.
3200 Washington Street
Midland, MI 48640

**The speed and efficiency of Microsoft BASIC result from an insightful software design technique.**

Microsoft BASIC used in the PET and OSI computers is fast and memory efficient. One reason for this is that the BASIC commands are abbreviated through use of tokens. For example, if you write the BASIC program:

```
10  IFA=BTHENGOSUB99
```

you will not find the words IF, THEN or GOSUB should you PEEK into the BASIC program. If OSI owners with BASIC in ROM run the following in immediate mode:

```
FOR X=768 TO 781
PRINT PEEK(X)
NEXT X
```

The BASIC line will look like this:

```
0 14 3 10 0 138 65 171 66 160 140 57 57 0
```

So let's try to pick this apart and see what happened. The leading and trailing 0's are delimiters to separate BASIC lines. The "14 3" in the second and third byte means the next BASIC line starts at memory location 14 + 3*256 = 782 (decimal). The "10 0" in the next two bytes indicates this is BASIC line 10 + 0*256 = 10. If you look in a table of ASCII codes, 65, 66 and 57 are the ASCII values for A, B and 9.

Thus our code deciphering so far yields:

```
0 14 3 10 0 138 65 171 66 160 140 57 57 0
 \ / \  \     |      /          /    /
 782  #10     A      B         9    9 END
```

A little inspection of what is still missing indicates that somehow, "138" means IF, "171" means EQUALS, "160" means THEN and "140" means GOSUB. These are the tokens used in Microsoft BASIC.

The following program will decode tokens for OSI users.

```
10  REM
20  INPUT X
30  POKE 773, X
40  LIST 10
```

Start the program via "RUN 20" to skip over the first line. Then input a number between 65 and 195. For example, if you INPUT a 138, line 10 will now contain an IF.

Table 1 is a list of tokens for the OSI system. This will help in PEEKing around your BASIC programs. You could even write a program that rewrites itself. PET owners: Don't worry, I haven't forgotten you. To look at the first line of the BASIC program, run in immediate mode:

```
FOR X = 1024 TO 1037
PRINT PEEK(X)
NEXT X
```

Line 30 of the token decoder program should be changed to:

```
30  POKE 1029,X
```

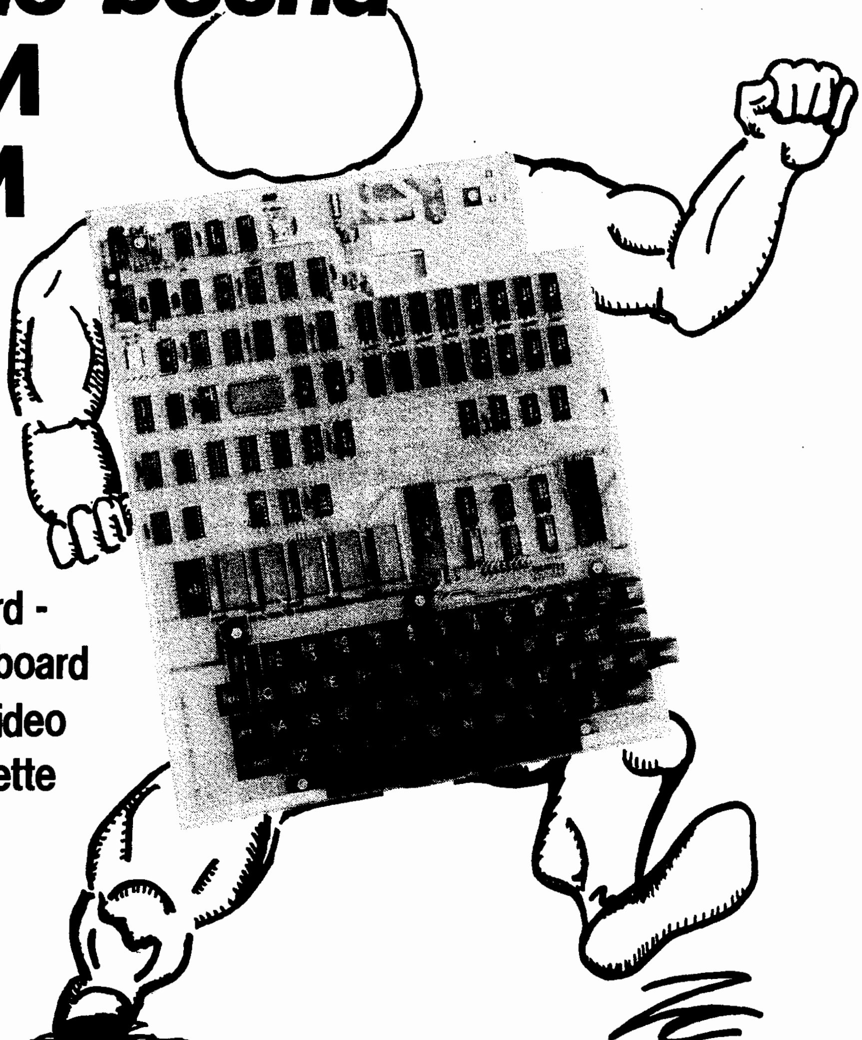
You will find the PET tokens are not identical to OSI's. So I leave it to you to build your own list.

*Editor: Thanks to Alvin L. Hooper, 207 Self St., Warner Robbins, GA 31093 who submitted an equivalent table of OSI BASIC tokens.*

***Table 1: OSI BASIC Token Index***

| | | | | | |
|---|---|---|---|---|---|
| **151** | **PRINT** | **128** | **END** | **174** | **INT** |
| **152** | **CONT** | **129** | **FOR** | **175** | **ABS** |
| **153** | **LIST** | **130** | **NEXT** | **176** | **USR** |
| **154** | **CLEAR** | **131** | **DATA** | **177** | **FRE** |
| **155** | **NEW** | **132** | **INPUT** | **178** | **POS** |
| **156** | **TAB(** | **133** | **DIM** | **179** | **SQR** |
| **157** | **TO** | **134** | **READ** | **180** | **RND** |
| **158** | **FN** | **135** | **LET** | **181** | **LOG** |
| **159** | **SPC(** | **136** | **GOTO** | **182** | **EXP** |
| **160** | **THEN** | **137** | **RUN** | **183** | **COS** |
| **161** | **NOT** | **138** | **IF** | **184** | **SIN** |
| **162** | **STEP** | **139** | **RESTORE** | **185** | **TAN** |
| **163** | **+** | **140** | **GOSUB** | **186** | **ATN** |
| **164** | **–** | **141** | **RETURN** | **187** | **PEEK** |
| **165** | ***** | **142** | **REM** | **188** | **LEN** |
| **166** | **/** | **143** | **STOP** | **189** | **STR$** |
| **167** | **(power of)** | **144** | **ON** | **190** | **VAL** |
| **168** | **AND** | **145** | **NULL** | **191** | **ASC** |
| **169** | **OR** | **146** | **WAIT** | **192** | **CHR$** |
| **170** | **>** | **147** | **LOAD** | **193** | **LEFT$** |
| **171** | **=** | **148** | **SAVE** | **194** | **RIGHT$** |
| **172** | **<** | **149** | **DEF** | **195** | **MID$** |
| **173** | **SGN** | **150** | **POKE** | **197-211** | **BASIC Error Codes** |

# Faster than a speeding mini
# Able to leap tall micros
# in a single bound
# 4K RAM
# 8K ROM



## $279

Ohio Scientific Superboard - the computer on a board - even includes a keyboard and interface for video display and a cassette recorder.

## IT'S SUPERBOARD

## COMPUTERSHOP

288 Norfolk St. (Cor. Hampshire St.)
Cambridge, Mass. 02139
617-661-2670

590 Commonwealth Ave.
Boston, Mass. 02215
617-247-0700

Route 16B
Union, N.H. 03887
603-473-2323

# A Better LIFE for Your APPLE

**An enhancement to LIFE makes it easy to establish an initial pattern, monitor successive generations, and modify the pattern at any particular generation. This input technique is cursor oriented and keyboard driven to facilitate entering complex patterns.**

**L. William Bradford**
**7868 Naylor Avenue**
**Los Angeles, CA 90045**

It was a distinct pleasure to see Richard F. Suitor's article, *Life For Your Apple* in MICRO 8:11. Since my introduction to this mathematical game through a program written by an associate, I have derived a great deal of pleasure from watching the evolution of many "life" forms. I was quite taken by the execution speed of Mr. Suitor's program, but I feel that his method of designating a living cell is awkward, especially for large complex patterns.

I would like to pass on to other MICRO readers a technique employed by W.P. Hennessy in that very first LIFE program I used. While I have made sustantial changes to make the program easier and a little more versatile, the technique remains the same.

Instead of using the inconvenient INPUT X,Y, the operator may move a cursor about the screen, depositing or erasing cells, or moving without disturbing cells. The cursor is a single white "brick" whose motion is controlled by depressing one of the keys described below:

| KEY | DIRECTION OF MOTION |
|---|---|
| N,U | Bottom to Top |
| E,R | Left to Right |
| S,D | Top to Bottom |
| W,L | Right to Left |

The keys N, E, W, and S have a very different function than the U, D, R, and L keys, since the former move the cursor without affecting the screen, while the latter cause a cell to be deposited or erased from the screen. In every case, the cursor moves one space per keystroke.

The U, D, R, and L keys are used in two modes, the "write" mode and the "erase" mode, with "write" mode being the default. As an example, suppose that the program is in the default mode, and the operator depresses the U key. The cursor will move one space up, leaving a live cell in the square just vacated. The erase mode is entered by depressing the ESC key, and the write mode re-entered by depressing the O (as in orange) key.

Assuming that the cursor is centered on a live cell, and that the program is in the erase mode, depressing the U key will cause the live cell to be deleted and the cursor to move up. There is no effect on unoccupied cells. If this sounds complicated at first, it is nonetheless simple in practice.

Once a pattern has been entered, the RETURN key is depressed to start the program. I have retained the heart of Mr. Suitor's BASIC program which sets up the timing loops and calls the machine language subroutines. I have made some slight changes to his routine to generate a random pattern by setting up a default grid size and using a different randomization.

In the present version of the program, execution will stop briefly after some number of generations. The number of generations is a function of the default timer loop interval which the operator designates. During the pause, the program will be examining the keyboard, looking for certain keys. These keys and their functions are described in Table 1.

The duration of the pause can be controlled by changing the value of the variable JK at statement 315. If the user should wish to pause after each generation, the following statements will effect that change:

```
306  GOSUB 315: NEXT I
350  RETURN
366  IF IN = 82 THEN RETURN
```

The program also allows the operator to run without any pauses provided that he answers in the affirmative to the question at statement 14. In general, this is the way that I run the program.

The APPLE LIFE fan will find that the code presented here, when coupled with Richard Suitor's excellent machine language code, will provide many hours of entertainment and mental stimulation. John Conway's game of LIFE is surely one of the more exciting uses of the personal computer.

*Table 1: Single Key Functions*

| KEY | FUNCTION |
|---|---|
| P | Stop execution and wait |
| K | Stop and clear screen, get new pattern |
| X | Exit to Basic |
| M | Stop to allow modification of pattern |
| G | Restart execution |

```
0 TEXT : GOTO 2
1 Q= PEEK (-16384): IF Q<127 THEN
  1:Q=Q-128: POKE -16368,0: RETURN

2 CALL -936: VTAB 9: TAB 15: PRINT
  "** LIFE **": PRINT : PRINT

3 PRINT " A VERSION OF JOHN CONWAY
  'S GAME OF LIFE": PRINT
4 TAB 10: PRINT "WRITTEN FOR THE A
  PPLE II"
5 VTAB 15: PRINT " ASSEMBLY LANGUA
  GE ROUTINES WRITTEN BY   RICHARD
   F SUITOR AND PUBLISHED IN ISSUE
  "
6 PRINT "NO. 8 OF 'MICRO' COPYRIGH
  T 1978": PRINT "BASIC ROUTINES B
  Y L.W. BRADFORD 1978"
7 VTAB 22: INPUT "DO YOU WANT INST
  RUCTIONS?",X$
8 CALL -936
9 IF X$="Y" THEN 2000
10 TEXT : GR
12 ZZ=0
14 INPUT "DO YOU WANT THE PROGRAM T
   O RUN WITHOUT   EXTERNAL COMMAND
   S",X$
15 IF X$#"Y" AND X$#"N" THEN 14
   : IF X$="N" THEN 20: IF X$=
   "Y" THEN ZZ=1
20 CALL -936
21 INPUT "ENTER DEFAULT VALUE FOR T
   IMER INTERVAL",KX1
32 INPUT "DO YOU WANT A RANDOMLY OC
   CUPIED SPACE",X$
33 IF X$#"Y" AND X$#"N" THEN 32
   : IF X$="N" THEN 100
40 INPUT " STANDARD GRID SIZE (0<X<
   39,0<Y<47) ",X$
41 IF X$#"Y" AND X$#"N" THEN 40
   : IF X$="N" THEN 54
42 J1=1:J2=46:I1=1:I2=38: GOTO
   59
54 INPUT "ENTER X DIRECTION LIMITS
   (0 TO 39)",I1,I2
55 IF I1<0 OR I2>39 THEN 54
56 INPUT "ENTER Y DIRECTION LIMITS
   (0 TO 47)",J1,J2
57 IF J1<0 OR J2>47 THEN 56
59 SI= RND (4)+1:SJ= RND (3)+1

60 GR : POKE -16302,0
61 CALL -1998
62 FOR I=I1 TO I2 STEP SI
63 FOR J=J1 TO J2 STEP SJ
64 COLOR=11:X= RND (2)+1:X=X*(
    RND (2))+1: IF RND (X) THEN
   COLOR=0
65 PLOT I,J
66 NEXT J
67 NEXT I
68 GOTO 292
100 GR : POKE -16302,0
101 COLOR=0
105 FOR JK=0 TO 39: VLIN 0,47 AT
    JK
106 NEXT JK
110 LIVE=11:DEAD=0:CURS=15:TEMP=
    LIVE
115 COLOR=0: FOR X=1 TO 38: VLIN
    1,46 AT X: NEXT X
120 X=18:Y=23
125 SC1= SCRN(X,Y)
128 COLOR=CURS: PLOT X,Y
130 GOSUB 1
132 IF Q=27 THEN TEMP=0: IF Q=79
     THEN TEMP=11: IF Q=27 OR Q=
    79 THEN 130
133 COLOR=TEMP
134 IF Q=69 OR Q=87 OR Q=83 OR
    Q=78 THEN COLOR=SC1
136 PLOT X,Y
140 IF Q=13 THEN 290
142 IF Q=32 THEN 200
144 IF Q=69 OR Q=82 THEN 200
146 IF Q=87 OR Q=76 THEN 210
148 IF Q=83 OR Q=68 THEN 220
150 IF Q=78 OR Q=85 THEN 230
160 FOR JZ=1 TO 10
161 J= PEEK (-16336): NEXT JZ
162 GOTO 125
200 X=X+1: IF X>38 THEN X=38: GOTO
    125
210 X=X-1: IF X<1 THEN X=1: GOTO
    125
220 Y=Y+1: IF Y>46 THEN Y=46: GOTO
    125
230 Y=Y-1: IF Y<1 THEN Y=1: GOTO
    125
290 COLOR=0: PLOT X,Y
292 GOTO 307
294 FOR I=1 TO K3
296 CALL 2088
298 FOR K=1 TO K1: NEXT K
300 CALL 2265
302 FOR K=1 TO K2: NEXT K
306 NEXT I
307 KX= PDL (0)-10
308 IF KX>240 THEN KX=KX1
309 IF KX<0 THEN KX=0
310 K2=KX*2:K1=KX*6
311 K3=500/(K1+50)+1
312 IF ZZ=1 THEN 294
315 JK=100
320 FOR NN=1 TO JK
325 IN= PEEK (-16384)
330 IF IN>127 THEN 360
335 POKE -16368,0
340 NEXT NN
352 GOTO 294
360 IN=IN-128
365 POKE -16368,0
369 IF IN=77 THEN 120
370 IF IN=75 THEN 10
372 IF IN=71 THEN 294
373 IF IN=80 THEN 400
374 FOR IJ=1 TO 20
375 KK= PEEK (-16336)
376 NEXT IJ
380 IF IN=88 THEN 1000
400 IN= PEEK (-16384)
```

```
 410 IF IN>127 THEN 360
 415 POKE -16368,0
 420 GOTO 400
1000 TEXT : CALL -936
1001 END
2000 VTAB 3: PRINT " YOU GENERATE A S
     ET OF 'LIVE' CELLS": PRINT
     "BY MOVING THE CURSOR WITH THE"
     : PRINT "KEYS DESCRIBED BELOW"
     : PRINT
2001 PRINT "  IN THE 'WRITE' MODE THE
     SE": PRINT "CHARACTERS GENERATE
     A LIVE CELL": PRINT
2002 PRINT "  IN THE 'ERASE' MODE THE
      SAME": PRINT "CHARACTERS ERASE
     A LIVE CELL"
2003 PRINT : PRINT "YOU START OUT IN
     THE 'WRITE' MODE"
2004 PRINT "AND STAY THERE UNTIL YOU
     HIT 'ESC'"
2005 PRINT : PRINT "TYPE A 'O' TO RE-
     ENTER THE 'WRITE' MODE": PRINT

2006 PRINT "U=UP D=DOWN R=RIGHT L=LEF
     T": PRINT
2007 PRINT "TYPE ANY KEY TO CONTINUE"
     : GOSUB 1
2008 CALL -936: VTAB 2
2009 PRINT " TO MOVE WITHOUT WRITING"
     : PRINT " OR ERASING ANYTHING"

2010 PRINT "USE THE FOLLOWING CHARACT
     ERS"
2011 PRINT : PRINT "N=UP S=DOWN E=RIG
     HT W=LEFT": PRINT
2012 PRINT "WHEN FINISHED, HIT 'RETUR
     N'": PRINT
2020 PRINT "AFTER EACH GENERATION, YO
     U MAY,"
2021 PRINT "BY USING THE APPROPRIATE
     KEY"
2022 PRINT : PRINT "PAUSE (TYPE A 'P'
     ) OR": PRINT
2024 PRINT "CONTINUE FROM THE GENERAT
     ION ON THE"
2025 PRINT "SCREEN  (TYPE A 'G') "
     : PRINT
2026 PRINT "RETURN TO BASIC TYPE AN '
     X' ": PRINT
2027 PRINT "TYPE ANY KEY TO CONTINUE"
     : GOSUB 1: CALL -936
2028 PRINT "MODIFY THE PRESENT PATTER
     N (TYPE AN 'M')"
2029 VTAB 4: PRINT "OR TYPE A 'K' TO
     START A NEW GAME"
2030 VTAB 8: PRINT "AFTER YOU HAVE HI
     T 'S', YOU MAY TYPE": PRINT

2031 TAB 7: PRINT "M,P,G,K, OR X"
     : PRINT
2040 PRINT
2042 PRINT " AN ADDITIONAL FACILITY O
     F A RANDOMLY": PRINT "OCCUPIED S
     PACE IS ALLOWED"
2045 PRINT : PRINT
2048 PRINT "TYPE ANY KEY TO CONTINUE"
     : GOSUB 1
3000 CALL -936
3035 PRINT : PRINT : PRINT "TYPE ANY
     KEY TO START THE GAME": GOSUB
     1
3036 GOTO 10
```

# EPROM for the KIM

**Circuits and suggestions for the selection, installation and utilization of EPROM. This fully buffered EPROM board is easy to build and use. It requires no special interfacing.**

**William C. Clements, Jr.**
**Department of Chemical and Metallurgical Engineering**
**University of Alabama**
**University, AL 35486**

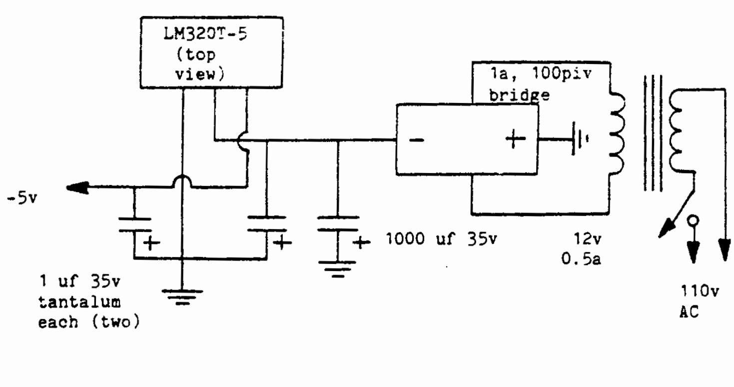
One of the handiest additions for the expansion-minded KIM owner to consider is an EPROM board. There's nothing like being able to summon your favorite programs as soon as the computer is turned on. Most people think of PROM's in terms of holding BASIC or an operating system, but there's no reason your favorite games and utilities shouldn't be there too. The most heavily used routines in my 2708s are Hypertape and Browse, both from the *The First Book of Kim*, and the XIM Teletype utilities. Tiny BASIC will go in PROM as soon as I can find time to relocate it. QUICK, a reaction-time game from *The First Book of Kim*, is there too; it's fun, and a nice way to show off the computer.

There are lots of articles from which one can build EPROM programmers, and some of these are specifically for use with KIM. The most EPROM for the money currently seems to be the 2708. Prices in the $6 range for 1K 8-bit words (650 ns access time, fine for KIM) are hard to beat for any type of computer memory. Just one of these things holds as much as the entire user RAM! 2708/2716 programmers are also available as kits or assembled from dealers, but most are quite expensive. An exception is Optimal Technology's unit, which is in the $50 range; that's what I have, and it works beautifully. Incidentally, their programming software can be relocated easily by hand, and it now resides in a PROM too.

There seems to be considerably less information available on using PROMs with KIM. Most of the commercial boards and construction articles are for the S-100 bus, which doesn't help the KIM owner a bit unless he already has a KIMSI or similar interface. Fortunately, a fully buffered EPROM board with address decoding is very easy to build and use with KIM with no special interfacing. My unit is shown on the accompanying schematic. It was wire-wrapped by hand on a small piece of Vector perfboard, using sockets held in place with G.E. silicone cement, and contains address decoding for up to 16 EPROM's beginning at address C000 hex.

Two type 8T97 hex buffers are used to buffer the lower ten address lines, since all the EPROM's are in parallel across this part of the address bus. Two sections in the second 8T97 were left over, and were used to buffer KIM's lines AB14 and AB15 rather than let them be unused; substituting a 74LS00 in place of the 7400 would provide a similar load on the address bus, but I wanted to buffer as many address lines as I could to make further expansion easier. The 74LS154 four-to-sixteen line decoder provides the CS signal that gates a different EPROM for each 1K of memory space, and the NAND gate activates this decoder when bits 14 and 15 of the address bus are both high (address ≥ C000).

The vector-fetch and decode-enable signals required by KIM are generated in my system by expansion RAM boards; you will have to provide them yourself if you don't already have some form of memory expansion. Although not shown on the diagram, 0.01 or 0.1 mf bypass capacitors were used from +5V, +12V, and −5V points to ground on most ICs. A LM32OT-5 IC regulator provided −5V for the 2708s from my existing power supply.



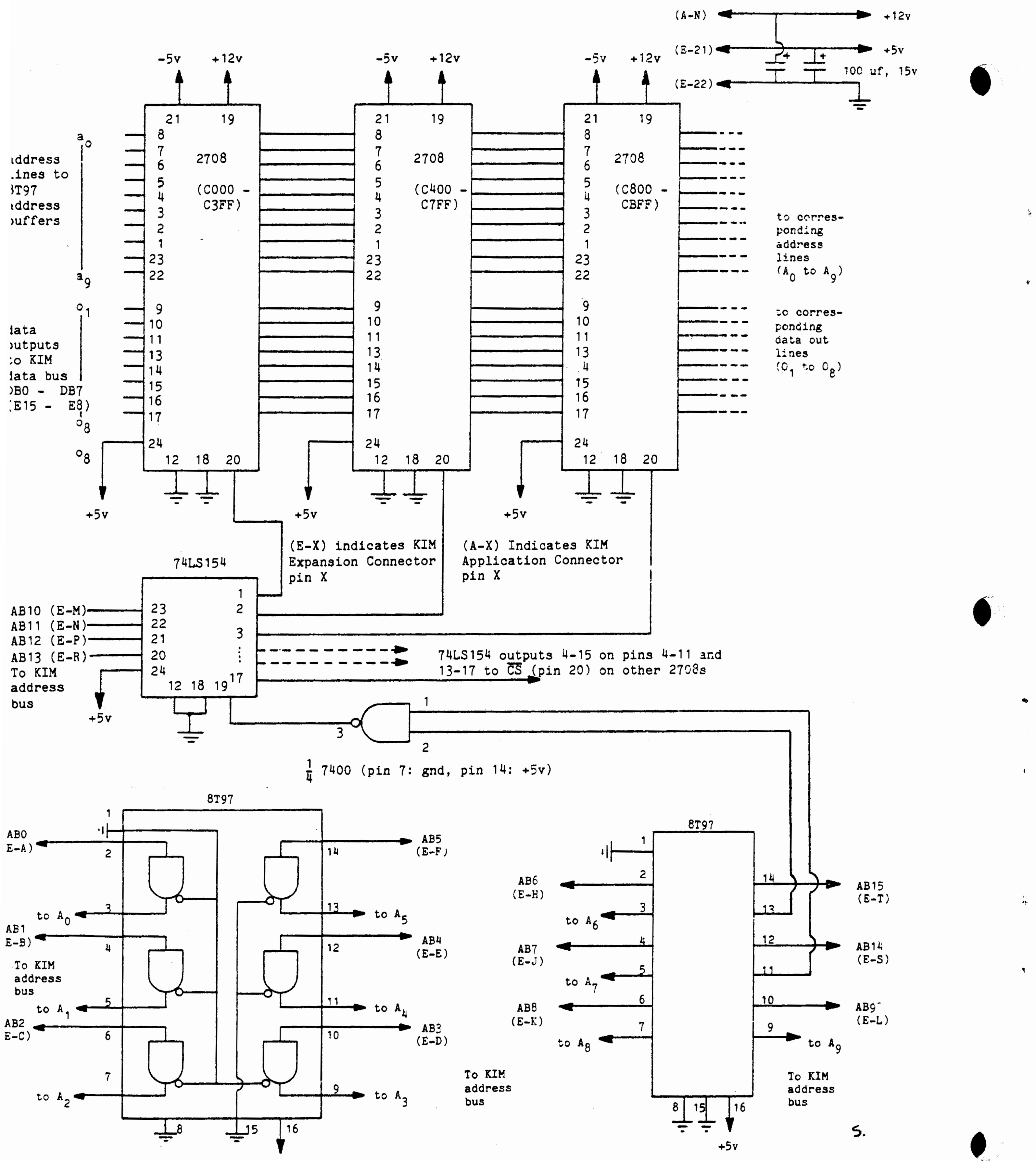

There is a beneficial side-effect from using EPROM's which is not enough talked about. Use of these devices provides a strong encouragement toward cleaning up and refining your programming habits! If you are not already careful that your program contains "clean" or non self-modifying code, you will quickly get into the habit if you have any kind of ROM board.

A certain amount of ingenuity can often show you how to adapt other's software to PROM. If a table in page zero needs to be initialized before running a program, just append your own short program to move the data block from PROM down to page zero, and then transfer control to the start of the main program. I like to write short driver routines like this when PROMming a program that requires register initialization from the keyboard to run different cases.

If the program is going to be kept in PROM for years, it is easy to forget which numbers go where and at what times. I'd rather just have to remember a single starting address for each separate case, and let my driver program do the initializing. For instance, I begin Microchess at one address for superblitz play, at another for blitz, and at a third for regular play. These addresses set the proper constants for each level of play; the original version required changes of instructions in the program itself, which is not possible in ROM.

If a program is self-modifying, and you can't figure out how to fix it without starting over, don't despair; put it as is (unrelocated) into PROM, along with a little routine that copies it into lower memory and then transfers control to it there.

Using such a routine, the program appears to the user as though it is executing directly from PROM except, of course, that the lower memory is not available for other uses during execution. If that is not a problem, you could even store *all* your programs in PROM, preceded by a move routine, and be spared the work of relocating or modifying any of them! If you have lots of expansion RAM, this is probably the most hassle-free way to go. However, you choose to do it, relocating and running direct from PROM, or moving and running an unmodified program, using EPROM's will be a lot of fun. And think of all the tape you'll save!


(A-N)
+12v
(E-21)
+5v
(E-22)
100 uf, 15v
-5v
+12v
2708
(C000 - C3FF)
2708
(C400 - C7FF)
2708
(C800 - CBFF)
address lines to 8T97 address buffers
data outputs to KIM data bus DB0 - DB7 (E15 - E8)
to corres- ponding address lines ($A_0$ to $A_9$)
to corres- ponding data out lines ($O_1$ to $O_8$)
+5v
(E-X) indicates KIM Expansion Connector pin X
(A-X) Indicates KIM Application Connector pin X
74LS154
AB10 (E-M)
AB11 (E-N)
AB12 (E-P)
AB13 (E-R)
To KIM address bus
74LS154 outputs 4-15 on pins 4-11 and 13-17 to $\overline{CS}$ (pin 20) on other 2708s
1/4 7400 (pin 7: gnd, pin 14: +5v)
8T97
AB0 (E-A)
to $A_0$
AB1 (E-B)
To KIM address bus
to $A_1$
AB2 (E-C)
to $A_2$
AB5 (E-F)
to $A_5$
AB4 (E-E)
to $A_4$
AB3 (E-D)
to $A_3$
To KIM address bus
8T97
AB6 (E-H)
to $A_6$
AB7 (E-J)
to $A_7$
AB8 (E-K)
to $A_8$
AB15 (E-T)
AB14 (E-S)
AB9 (E-L)
to $A_9$
To KIM address bus
+5v

# PROGRESSIVE SOFTWARE

**Presents**
**Software and Hardware for your APPLE**

**SALES FORECAST** provides the best forecast using the four most popular forecasting techniques: linear regression, log trend, power curve trend, and exponential smoothing. Neil D. Lipson's program uses artificial intelligence to determine the best fit and displays all results for manual intervention. **$9.95**

**CURVE FIT** accepts any number of data points, distributed in any fassion, and fits a curve to the set of points using log curve fit, exponential curve fit, least squares, or a power curve fit. It will compute the best fit or employ a specific type of fit, and display a graph of the result. By Dave Garson. **$9.95**

**PERPETUAL CALENDAR** may be used with or without a printer. Apart from the usual calendar functions, it computes the number of days between any two dates and displays successive months in response to a single keystroke. Written by Ed Hanley. **$9.95**

**STARWARS** is Bob Bishop's version of the original and best game of intergallactic combat. You fire on the invader after aligning his fighter in your crosshairs. This is a high resolution game, in full color, that uses the paddles. **$9.95**

**ROCKET PILOT** is an exciting game that simulates blasting off in a rocket ship. The rocket actually accelerates you up and over a mountain; but if you are not careful, you will run out of sky. Bob Bishop's program changes the contour of the land every time you play the game. **$9.95**

**SPACE MAZE** puts you in control of a rocket ship that you must steer out of a maze using paddles or a joystick. It is a real challenge, designed by Bob Bishop using high resolution graphics and full color. **$9.95**

**MISSILE ANTI-MISSILE** displays a target on the screen and a three dimensional map of the United States. A hostile submarine appears and launches a pre-emptive nuclear attack controlled by paddle 1. As soon as the hostile missile is fired, the U.S. launches its anti-missile controlled by paddle 0. Dave Moteles' program offers high resolution and many levels of play. **$9.95**

**MORSE CODE** helps you learn telegraphy by entering letters, words or sentences, in English, which are plotted on the screen using dots and dashes. Ed Hanley's program also generates sounds to match the screen display, at several transmission speed levels. **$9.95**

**POLAR COORDINATE PLOT** is a high resolution graphics routine that displays five classic polar plots and also permits the operator to enter his own equation. Dave Moteles' program will plot the equation on a scaled grid and then flash a table of data points required to construct a similar plot on paper. **$9.95**

**UTILITY PACK 1** combines four versatile programs by Vince Corsetti, for any memory configuration.

- **Integer to Applesoft conversion:** Encounter only those syntax errors unique to Applesoft after using this program to convert any Integer BASIC source.
- **Disk Append:** Merge any two Integer BASIC sources into a single program on disk.
- **Integer BASIC copy:** Replicate an Integer BASIC program from one disk to another, as often as required, with a single keystroke.
- **Applesoft Update:** Modify Applesoft on the disk to eliminate the heading always produced when it is first run.
- **Binary Copy:** Automatically determines the length and starting address of a program while copying its binary file from one disk to another in response to a single keystroke. **$9.95**

**BLOCKADE** lets two players compete by building walls to obstruct each other. An exciting game written in Integer BASIC by Vince Corsetti. **$9.95**

**TABLE GENERATOR** forms shape tables with ease from directional vectors and adds additional information such as starting address, length and position of each shape. Murray Summers' Applesoft program will save the shape table anywhere in usable memory. **$9.95**

**OTHELLO** may be played by one or two players and is similar to chess in strategy. Once a piece has been played, its color may be reversed many times, and there are also sudden reverses of luck. You can win with a single move. Vince Corsetti's program does all the work of keeping board details and flipping pieces. **$9.95**

**SINGLE DRIVE COPY** is a special utility program, written by Vince Corsetti in Integer BASIC, that will copy a diskette using only one drive. It is supplied on tape and should be loaded onto a diskette. It automatically adjusts for APPLE memory size and should be used with DOS 3.2. **$19.95**

**SAUCER INVASION** lets you defend the empire by shooting down a flying saucer. You control your position with the paddle while firing your missile at the invader. Written by Bob Bishop. **$9.95**

## HARDWARE

**LIGHT PEN** with seven supporting routines. The light meter takes intensity readings every fraction of a second from 0 to 588. The light graph generates a display of light intensity on the screen. The light pen connects points that have been drawn on the screen, in low or high resolution, and displays their coordinates. A special utility displays any number of points on the screen, for use in menu selection or games, and selects a point when the light pen touches it. The package includes a light pen calculator and light pen TIC TAC TOE. Neil D. Lipson's programs use artificial intelligence and are not confused by outside light. The hi-res light pen, only, requires 48K and ROM card. **$34.95**

## POSTAGE AND HANDLING

Please add $1.00 for the first item
and $.50 for each additional item.

- **Programs accepted for publication**
- **Highest royalty paid**

## TO ORDER

Send check or money order to:

**P.O. Box 273**
**Plymouth Meeting, PA 19462**

PA residents add 6% sales tax.

**U.S. and foreign dealer and distributor inquiries invited**
**All programs require 16K memory unless specified**

# What's Where in the APPLE

**Professor William F. Luebbert**
**Dartmouth College**
**Hanover, NH 03755**

**Whether you are programming in BASIC or assembly language, a memory map helps save time, reduce program size and improve performance. This is the most complete and up to date APPLE memory map ever published.**

To get the most out of an APPLE, or any other computer with limited resources, it is important to know a good deal about the hardware and software environment.

When one graduates from simple programs to more ambitious programs involving careful control of man-machine interaction, analog to digital or digital to analog conversion, extensive use of computer graphics, the control of external devices, database management, sorting, word-processing or any of a wide variety of interesting tasks, this knowledge tends to become more important. When (and if) one gets into real time programming, adding his own specialized interfaces, performs activities where one must get the absolute maximum speed or gets into other situations where machine language programming is appropriate, it becomes critical.

Not every serious programmer needs to become a machine language level programmer. However, good programmers know that when the computer is running their programs there is a good deal of machine language code in the machine providing an operating environment for their programs. This operating environment typically includes the system monitor, a BASIC interpreter and possibly a disk operating system (DOS) and/or extra ROM packages.

When one looks at interesting programs described in magazines and user group newsletters, he finds that these programs often contain PEEKs, POKEs and CALLs. These are commands which are extensions of BASIC (or other higher level languages). They are provided to allow one to interface with the computer hardware, operating environment software, and other machine language programs or subprograms.

PEEKs, POKEs and CALLs all refer to memory locations which are identifiable as to what they contain or what they do. a PEEK examines the contents of a specified memory location and allows one to use that content in a program. POKE changes the contents of a designated memory location to some specified value. It can be used to change parameters of the operating environment or to set up or change pieces of program or data. A CALL transfers program control to a particular memory location and sets up a return linkage for transfer back to the CALLing routine in the user's program.

Pieces of the monitor or some other parts of the operating environment can often be accessed via CALLs, POKEs and PEEKs to modify system operation or to perform desired functions without the necessity of additional code. Usually this code has been carefully written in machine language and optimized by good programmers, so it runs faster and takes less space or less computer time than the same function would require if programmed totally by the user.

A programming manual intended for serious programmers should supply some sort of memory map and information about the most important and frequently used PEEKs, POKEs and CALLs. A good memory map can show the user where he can get information from the computer, what potentially useful software is available but perhaps hidden away inside the computer, and the "hooks" provided to perform a wide variety of functions by means of CALLs, POKEs and/or PEEKs. Often it becomes the most well-worn section of the manual. Once programmers begin using it as a source of information, they begin to wish for a more complete atlas which will let them find more and more information and guide them in their own explorations inside the computer and its software.

The memory map presented here was developed initially as a programming aid for my own personal programming. Important sources of information for its creation included the *APPLESOFT II Manual, the APPLE Reference Manual, WOZPAC* and various issues of *MICRO, Call-Apple* and *NEAT* as well as my own investigations inside the computer.

The map is being circulated for comment, correction and modification by many of the more active members of the New England Apple Tree User's Group. They have suggested valuable changes, corrections and additions. Inevitably there will still be errors and omissions. For these I beg your indulgence.

This memory atlas is stored on-line on the Dartmouth Timeshare System in a database which can be used for selective retrieval and report generation using standard database management software. The author would appreciate corrections or suggested changes or additions. Please mail them to him at Hinman, Box 6166, Dartmouth College, Hanover, NH 03755.

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $0000-$00FF | 0-255 | . | HARDWARE PAGE ZERO |
| $0000-$0005 | 0-5 | . | JUMP INSTRUCTIONS TO CONTINUE IN APPLESOFT |
| $0000-$0001 | 0-1 | ROL~ROH | SWEET-16 (16-BIT INTERPRETER) REGISTER R0 |
| $0000 | 0 | LOC0 | MONITOR MEMORY LOCATION 'LOC0' |
| $0001 | 1 | LOC1 | MONITOR MEMORY LOCATION 'LOC1' |
| $000A-$000C | 10-12 | . | LOCN FOR USR FUNCTION'S JUMP INSTRUCTION |
| $000D-$0017 | 13-23 | . | GENERAL PURPOSE COUNTERS/FLAGS FOR APPLESOFT |
| $001A-$001B | 26-27 | . | HI-RES GRAPHICS ON-THE-FLY SHAPE POINTER |
| $001A-$001B | 26-27 | SHAPEL~SHAPEH | HIRES POINTER TO SHAPE LIST |
| $001C | 28 | . | HI-RES GRAPHICS ON-THE-FLY COLOR BYTE |
| $001C | 28 | HCOLOR1 | HIRES RUNNING COLOR MASK |
| $001D | 29 | COUNTH | HI-RES GRAPHICS HIGH-ORDER BYTE OF STEP COUNT FOR LINE |
| $001E-$001F | 30-31 | R15L~R15H | SWEET-16 (16-BIT INTERPRETER) REGISTER R15 |
| $0020-$004F | 32-79 | . | APPLE II SYSTEM MONITOR RESERVED LOCATIONS |
| $0020 | 32 | WNDLEFT | SCROLLING WINDOW: LEFT SIDE (0-39 OR $0-$27) |
| $0021 | 33 | WNDWDTH | SCROLLING WINDOW: WIDTH (1-40 OR $1-$28)(WNDLEFT+WNDWDTH<40) |
| $0022 | 34 | WNDTOP | SCROLLING WINDOW: TOP LINE (0-23 OR $0-$16) |
| $0023 | 35 | WNDBTM | SCROLLING WINDOW: BOTTOM LINE (0-23 OR $0-$16)(WNDBTM>WNDTOP) |
| $0024 | 36 | CH | CURSOR: HORIZONTAL POSITION (0-39 OR $0-$27) |
| $0025 | 37 | CV | CURSOR: VERTICAL POSITION (0-23 OR $0-$17) |
| $0026-$0027 | 38-39 | GBASL~GBASH | LO-RES GRAPHICS POINTER TO LEFTMOST BYTE OF CUR. PLOT LINE |
| $0026-$0027 | 38-39 | HBASL~HBASH | HI-RES GRAPHICS ON-THE-FLY BASE ADDRESS |
| $0028-$0029 | 40-41 | BASL~BASH | MONITOR BASE ADDRESS POINTER |
| $002A-$002B | 42-43 | BAS2L~BAS2H | MONITOR BASE ADDRESS POINTER 2 |
| $002C | 44 | H2 | LOW RES COLOR GRAPHICS H2 |
| $002C | 44 | LMNEM | MONITOR MEMORY LOCATION 'LMNEM' |
| $002C-$002D | 44-45 | RTNL~RTNH | MONITOR RETURN POINTER |
| $002D | 45 | V2 | LOW-RES COLOR GRAPHICS V2 |
| $002D | 45 | RMNEM | MONITOR MEMORY LOCATION 'RMNEM' |
| $002D | . | V2 | MONITOR MEMORY LOCATION 'V2' |
| $002E | 46 | MASK | LOW-RES COLOR GRAPHICS MASK |
| $002E | 46 | CHKSUM | MONITOR MEMORY LOCATION 'CHKSUM' |
| $002E | 46 | FORMAT | MONITOR & MINIASSEMBLER MEMORY LOCATION 'FORMAT' |
| $002F | 47 | LASTIN | MONITOR MEMORY LOCATION 'LASTIN' |
| $002F | 47 | LENGTH | MONITOR & MINIASSEMBLER MEMORY LOCATION 'LENGTH' |
| $002F | 47 | SIGN | MONITOR MEMORY LOCATION 'SIGN' |
| $0030 | 48 | COLOR | LO-RES COLOR GRAPHICS COLOR (FOR PLOT/HLIN/VLIN FUNCTIONS |
| $0030 | 48 | HMASK | HI-RES GRAPHICS HMASK ON-THE-FLY BIT MASK |
| $0031 | 49 | MODE | MONITOR & MINIASSEMBLER MEMORY LOCATION 'MODE' |
| $0032 | 50 | INVFLG | VIDEO FORMAT CONTROL: 255($FF)=NORMAL; 127($7F)=FLASHING; 63($3F)=INV |
| $0033 | 51 | PROMPT | PROMPT CHARACTER: PRINTED ON GETLN CALL |
| $0034 | 52 | YSAV | MONITOR & MINIASSEMBER MEMORY LOCATION 'YSAV' |
| $0035 | 53 | YSAV1 | MONITOR MEMORY LOCATION 'YSAV1' |
| $0035 | 53 | L | MINIASSEMBER MEMORY LOCATION 'L' |
| $0036-$0037 | 54-55 | CSWL~CSWH | PROGRAM COUNTER FOR USER EXIT ON COUT ROUTINE (MONITOR) |
| $0038-$0039 | 56-57 | KSWL~KSWH | PROGRAM COUNTER FOR USER EXIT ON KEYIN ROUTINE (MONITOR) |
| $003A-$003B | 58-59 | PCL~PCH | USER PROGRAM COUNTER SAVED HERE ON BRK TO MONITOR |
| $003C | 60 | XQT | MONITOR MEMORY LOCATION 'XQT' |
| $003C | . | XQTNZ | MONITOR MEMORY LOCATION 'XQTNZ' |
| $003C-$003D | 60-61 | A1L-A1H | MONITOR WORK BYTE PAIR A1 |
| $003E-$003F | 62-63 | A2L-A2H | MONITOR WORK BYTE PAIR A2 |
| $0040-$0041 | 64-65 | A3L-A3H | MONITOR WORK BYTER PAIR A3 |
| $0042-$0043 | 66-67 | A4L-A4H | MONITOR WORK BYTE PAIR A4 |
| $0044 | 68 | FMT | MINIASSEMBLER MEMORY LOCATION 'FMT' |
| $0044-$0045 | 68-69 | A5L-A5H | MONITOR WORK BYTE PAIR A5 |
| $0045 | 69 | ACC | USER AC SAVED HERE ON BRK TO MONITOR |
| $0046 | 70 | XREG | USER X-REG SAVED HERE ON BRK TO MONITOR |
| $0047 | 71 | YREG | USER Y-REG SAVE HERE ON BRK TO MONITOR |
| $0048 | 72 | STATUS | USER P STATUS SAVED HERE ON BRK TO MONITOR |
| $0049 | 73 | SPNT | USER STACK POINTER SAVED HERE ON BRK |
| $004A-$004B | 74-75 | LOMEML~LOMEMH | POINTER TO LOMEM |
| $004C-$004D | 76-77 | HIMEML~HIMEMH | POINTER TO HIMEM |
| $004E-$004F | 78-79 | RNDL~RNDH | 16 BIT NO. RANDOMIZED WITH EACH KEY ENTRY |
| $0050-$0061 | 80-97 | . | GENERAL PURPOSE POINTERS FOR APPLESOFT |
| $0050-$0051 | 80-81 | ACL~ACH | MONITOR POINTER 'AC' |
| $0050-$0051 | 80-81 | DXL~DXH | HIRES GRAPHICS DELTA-X FOR HLIN SHAPE |
| $0051 | 81 | SHAPEX | HIRES GRAPHICS SHAPE TEMP. |
| $0052 | 82 | DY | HIRES GRAPHICS DELTA-Y FOR HLIN SHAPE |
| $0052-$0053 | 82-83 | XTNDL~XTNDH | MONITOR 16-BIT POINTER 'XTND' |
| $0053 | 83 | QDRNT | HI-RES GRAPHICS QDRNT: 2 LSB'S ARE ROTATION QUADRANT FOR DRAW |
| $0054 | 84 | EL | HI-RES GRAPHICS ERROR FOR HLIN |
| $0054-$0055 | 84-85 | AUXL~AUXH | MONITOR 16-BIT POINTER 'AUX' |
| $0054-$0055 | 84-85 | EL~EH | HI-RES GRAPHICS ERROR FOR HLIN |
| $0055 | 85 | EH | HI-RES GRAPHICS ERROR FOR HLIN |
| $0062-$0066 | 98-102 | . | RESULT OF LAST MULTIPLY/DIVIDE |
| $0067-$0068 | 103-104 | START. PROG. PTR | POINTER TO BEGINNING OF PROGRAM. NORMALLY $0801 |
| $0069-$006A | 105-106 | LOMEM: | POINTER TO START OF SIMPLE VARIABLE SPACE |
| $006B-$006C | 107-108 | ARRAY POINTER | POINTER TO BEGINNING OF ARRAY SPACE |
| $006D-$006E | 109-110 | FREE SPACE PNTR | POINTER TO END OF NUMERIC STORAGE IN USE |
| $006F-$0070 | 111-112 | STRING POINTER | POINTER TO START OF STRING STORAGE. STRINGS TO END OF MEMORY |

LUEBBERT'S COPYRIGHT APPLE MEMORY ATLAS - MICRO MAGAZINE VERSION

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $0083-$0084 | 131-132 | . | POINTER TO THE LAST-USED VARIABLE'S VALUE |
| $0085-$009C | 133-156 | . | GENERAL USAGE |
| $0095 | . | PICK | MONITOR MEMORY LOCATION 'PICK' |
| $009D-$00A3 | 157-163 | . | MAIN FLOATING-POINT ACCUMULATOR |
| $00A4 | 164 | . | GENERAL USE IN FLOATING POINT MATH ROUTINES |
| $00A5-$00AB | 165-171 | . | SECONDARY FLOATING POINT ACCUMULATOR |
| $00AC-$00AE | 172-174 | . | GENERAL USAGE FLAGS/POINTERS |
| $00AF-$00B0 | 175-176 | PROGRAM POINTER | POINTER TO END OF PROGRAM. NOT CHANGED BY LOMEM: |
| $00B1 | 177 | . | CHRGET S/R CALL - GETS NEXT SEQUENTIAL CHR OR TOKEN |
| $00B1-$00C8 | 177-200 | . | CHRGET ROUTINE. CALLED WHEN A-S WANTS ANOTHER CHARACTER |
| $00B7 | 183 | CHRGOT | CHRGOT S/R CALL. CHRGET INCREMENTS TXTPTR. CHRGOT DOES NOT |
| $00B8-$00B9 | 184-185 | . | PTR TO LAST CHAR OBTAINED THRU CHRGET ROUTINE |
| $00B8-$00B9 | 184-185 | TXTPTR | TXTPTR - POINTS AT NEXT CHAR OR TOKEN FROM PROG (C/A DEC 78) |
| $00C9-$00CD | 201-205 | . | RANDOM NUMBER |
| $00CA-$00CB | 202-203 | PPL~PPH | BASIC START-OF-PROGRAM POINTER |
| $00CC-$00CD | 204-205 | PVL~PVH | BASIC END OF VARIABLES POINTER |
| $00CE-$00CF | 206-207 | ACL~ACH | BASIC ACC |
| $00D0-$00DF | 216-223 | . | ONERR POINTERS/SCRATCH |
| $00D0 | 216 | . | POKE 0 TOCLEAR ERROR FLAG |
| $00DE | 222 | . | WHEN ERROR OCCURS~ ERROR CODE APPEARS HERE |
| $00E0-$00E2 | 224-226 | . | HI-RES GRAPHICS X&Y COORDINATES |
| $00E4 | 228 | . | HI-RES GRAPHICS COLOR BYTE |
| $00E5-$00E7 | 229-231 | . | GENERAL USAGE FOR HI-RES GRAPHICS |
| $00E8-$00E9 | 232-233 | . | POINTER TO BEGINNING OF SHAPE TABLE |
| $00EA | 234 | . | COLLISION COUNTER FOR HI-RES GRAPHICS |
| $00F0-$00F3 | 240-243 | . | GENERAL USE FLAGS |
| $00F3 | . | SIGN | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'SIGN' |
| $00F4 | 244 | X2 | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'X2' (EXPONENT 2) |
| $00F4-$00F8 | 244-248 | . | ONERR POINTERS |
| $00F5 | 245 | M2 | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'M2' (MANTISSA 2) |
| $00F7 | 247 | S16PAG | SWEET-16 MEMORY LOCATION 'S16PAG' |
| $00F8 | 248 | X1 | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'X1' (EXPONENT 1) |
| $00F9 | 249 | M1 | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'M1' (MANTISSA 1) |
| $00FC | 252 | E | MONITOR & FLOATING POINT ROUTINES MEMORY LOC 'E' |
| $0100-$01FF | 256-511 | . | SUBROUTINE RETURN STACK |
| $0200 | 512 | IN | MONITOR & MINIASSEMBLER MEMORY LOCATION 'IN' |
| $0200-$02FF | 512-767 | . | KEYIN (INPUT) BUFFER |
| $0300-$03FF | 768-1023 | . | AREA CLOBBERED BY EITHER MASTER OR SLAVE DISKETTE BOOT |
| $0300-$03F7 | 768-1015 | . | OFTEN FREE SPACE. NOTE COMPETING USES OFTEN FREE SPACE CONSTRAINTS |
| $0300. $03AF | 768-943 | . | DECWRITER PRINTER OUTPUT (IF BLOADED FROM DISK) |
| $0320-$0321 | 800-801 | XOL~XOH | HI-RES GRAPHICS- PRIOR X-COORD SAVE AFTER HLIN OR HPLOT |
| $0322 | 802 | YO | HI-RES GRAPHICS YO - MOST RECENT Y-COORDINATE |
| $0323 | 803 | BXSAV | HI-RES GRAPHICS 'BXSAV' |
| $0324 | 804 | HCOLOR | HI-RES GRAPHICS COLOR FOR HPLOT~ HPOSN |
| $0325 | 805 | HNDX | HI-RES GRAPHICS HNDX - ON-THE-FLY BYTE INDEX FROM BASE ADDRESS |
| $0326 | 806 | HPAG | POKE 32 FOR HI-RES PG1 PLOTTING~ 64 FOR PAGE2 |
| $0326 | 806 | HPAG | HI-RES GRAPHICS MEM PAGE FOR PLOTTING GRAPHICS $20 FOR PG1 ~$40 FOR PG2 |
| $0327 | 807 | SCALE | ON-THE-FLY SCALE FACTOR FOR DRAW~ SHAPE~ MOVE |
| $0328-$0329 | 808-809 | SHAPXL~SHAPXH | START-OF-SHAPE-TABLE POINTER |
| $032A | 810 | COLLSN | COLLISION COUNT FROM DRAW~DRAW1 |
| $03D0 | 976 | . | DOS RE-ENTRY POINT (3D0G) |
| $03D0 | 976 | . | INITIALIZE OR RE-INITIALZE DOS (3D0G) |
| $03D3 | 979 | . | DOS 3.1 HARD ENTRY POINT |
| $03D6 | 982 | . | DOS 3.1 ENTRY POINT FOR I/O PACKAGE |
| $03D9 | 985 | . | DOS 3.1 ENTRY POINT FOR RWTS |
| $03DC | 988 | . | DOS 3.1 ENTRY POINT TO LOAD Y~A WITH ADDRESS AT END OF SYS BUFFER |
| $03E3 | 995 | 995 | DOS 3.1 ENTRY POINT TO LOAD Y~A WITH ADDRESS OF IOBLK |
| $03EA | 1002 | 1002 | DOS 3.2 ENTRY POINT FOR ROUTINE THAT UPDATES I/O HOOK TABLES |
| $03F8 | 1016 | USRADR | CTL-Y WILL CAUSE JSR HERE |
| $03FB | 1019 | NMI | NMI'S VECTORED TO THIS LOCATION |
| $03FE | 1022 | IRQADR | MONITOR MEMORY LOCATION 'IRQADR' |
| $03FE-$03FF | 1022-1023 | . | IRQ'S VECTORED TO ADDRESS WHOSE POINTER IS HERE |
| $0400-$07FF | 1024-2043 | . | SCREEN BUFFER (HARDWARE PAGES 4-7)(LOW-RES GRAPHICS & TEXT PAGE 1) |
| $0478+S | 1144+S | BRATE | SERIAL INTERFACE BAUD QUANTUM RATE. $1= 19200 BAUD; $40=300 BAUD |
| $0478+S | 1144+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $04F8+S | 1272+S | STBITS | SERIAL INTERFACE: CONTAIN NUMBER OF STOP BITS (INCLUDING 1 PARITY BIT) |
| $04F8+S | 1272+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $0578+S | 1400+S | STATUS | SERIAL INTERFACE: PARITY CHECKSUM OPTIONS (SEE MANUAL) |
| $0578+S | 1400+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT#S |
| $05F8+S | 1528+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $0678+S | 1656+S | BYTE | SERIAL INTERFACE INPUT OUTPUT BUFFER |
| $0678+S | 1656+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $06F8 | 1784+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $06F8+S | 1784+S | PWDTH | SERIAL INTERFACE PRINT LINE WIDTH (# CHARS PER LINE) |
| $0778+S | 1912+S | NBITS | SERIAL INTERFACE NUMBER OF DATA BITS PLUS 1 FOR START BIT |
| $0778+S | 1912+S | . | SCRATCHPAD MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $07F8+S | 2040+S | FLAGS | SERIAL INTERFACE OPERATION MODE |
| $07F8+S | 2040+S | . | INTERRUPT RETURN MEMORY BYTE FOR PERIPHERAL IN SLOT #S |
| $0800 | 2048 | . | DEFAULT INTEGER BASIC LOMEM |
| $0800-$09FF | 2048-2559 | . | AREA CLOBBERED BY EITHER MASTER OR SLAVE DISKETTE BOOT |
| $0800-$0BFF | 2048-3071 | . | SECONDARY SCREEN BUFFER (TEXT & LOW-RES GRAPHICS PAGE 2) |

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $0800-$C000 | 2048-49152 | . | RANGE OF POSSIBLE SETTINGS FOR HIMEM (DEPENDING UPON MEM SIZE~ DOS |
| $0800-LOMEM | 2048-LOMEM | . | PROGRAM STORAGE FOR ROM VERSION OF APPLESOFT |
| $0C00 | 3072 | . | DEFAULT LOCATION FOR START OF SHAPE TABLE AS SET BY HI-RES SHAPE LOAD |
| $0C00-$1FFF | 3072-8191 | . | OFTEN FREE SPACE |
| $0CF2 | 3314 | . | TO CNVRT A/S PROG FM ROM TO CASSETTE: LOAD PROG~ CALL 3314~LIST~SAVE |
| $1B00-$3FFF | 4000-16383 | . | THIS REGION OF MEMORY IS CLOBBERED BY A SLAVE DISKETTE BOOT |
| $1B00-$4000 | 6912-16384 | . | RAWDOS (VERSION OF DOS USED WITH MASTER.CREATE - FROM DISK) |
| $2000-$3FFF | 8192-16383 | . | HI-RES GRAPHICS PAGE 1 |
| $3000-LOMEM | 12288-LOMEM | . | PROGRAM STORAGE FOR RAM VERSION OF APPLESOFT |
| $3F3-$3F4 | 1011-1012 | . | DOS 3.1 - POKE TO ZEROS TO REBOOT HELLO PROGRAM |
| $4000-$4520 | 16384-17696 | . | NORMAL LOCATION FOR KAPOR'S HI RES TEXT SET |
| $4000-$5FFF | 16384-24575 | . | HI-RES GRAPHICS PAGE 2 |
| $4500 | 17664 | . | CALL FOR INVERSION BY KAPOR'S ROUTINE |
| $4500-4520 | 17664-17696 | . | S/R W/ KAPOR'S HI-RES TEXT SET TO INVERT WHITE TO BLACK & VICEVERSA |
| $5600-$8000 | 22016-32768 | . | DISK OPERATING SYSTEM (DOS3.1) |
| $9600-$9853 | -27136--26541 | . | DOS 3.1 USER BUFFER #1 |
| $9600-$9700 | -27136--26880 | . | DOS 3.1 USER BUFFER #1 DATA BUFFER |
| $9701-$9802 | -26879--26622 | . | DOS 3.1 USER BUFFER #1 - LIST OF SECTOR & TRACK NUMBERS USED |
| $9801-$9853 | -26623--26541 | . | DOS 3.1 USER BUFFER #1 - FILE NAME & MISC DATA |
| $9D10-? | -25328-? | . | STARTING ADDRESSES FOR VARIOUS DOS3.1 TASKS |
| $9D73-$A7DF | -25229--25561 | . | SYSTEM SECTION OF DOS 3.1 |
| $9DB9 | -29159 | . | INITIALIZE OR RE-INITIALIZE DOS |
| $9E4D | -25011 | . | ROUTINE WHICH HANDLES DOS INPUT HOOK |
| $9E7E | -24962 | . | ROUTINE WHICH HANDLES DOS OUTPUT HOOK |
| $A1B4 | -24140 | . | ADDRESS FOR DOS3.1 PR# COMMAND |
| $A1B9 | -24135 | . | ADDRESS FOR DOS 3.1 IN# COMMAND |
| $A1BE | -24130 | . | ADDRESS FOR DOS 3.1 MON COMMAND |
| $A1DC | -24100 | . | ADDRESS FOR DOS 3.1 MAXFILES COMMAND |
| $A1EE | -24082 | . | ADDRESS FOR DOS 3.1 DELETE COMMAND |
| $A1FC | -24068 | . | ADDRESS FOR DOS 3.1 LOCK COMMAND |
| $A200 | -24064 | . | ADDRESS FOR DOS 3.1 BSAVE COMMAND |
| $A200 | -24064 | . | ADDRESS FOR DOS 3.1 UNLOCK COMMAND |
| $A208 | -24056 | . | ADDRESS FOR DOS 3.1 VERIFY COMMAND |
| $A20C | -24052 | . | ADDRESS FOR DOS 3.1 RENAME COMMAND |
| $A223 | -24029 | . | ADDRESS FOR DOS 3.1 APPEND COMMAND |
| $A236 | -24010 | . | ADDRESS FOR DOS 3.1 OPEN COMMAND |
| $A278 | -23944 | . | ADDRESS FOR DOS 3.1 CLOSE COMMAND |
| $A2EC | -23828 | . | ADDRESS FOR DOS 3.1 BLOAD COMMAND |
| $A327 | -23769 | . | ADDRESS FOR DOS 3.1 BRUN COMMAND |
| $A330 | -23760 | . | ADDRESS FOR DOS 3.1 SAVE COMMAND |
| $A3A5 | -23643 | . | ADDRESS FOR DOS 3.1 LOAD COMMAND |
| $A476 | -23434 | . | ADDRESS FOR DOS 3.1 RUN COMMAND |
| $A48D | -23411 | . | ADDRESS FOR DOS 3.1 CHAIN COMMAND |
| $A4A5 | -23387 | . | ADDRESS FOR DOS3.1 WRITE COMMAND |
| $A4B0 | -23376 | . | ADDRESS FOR DOS 3.1 READ COMMAND |
| $A4E4 | -23324 | . | ADDRESS FOR DOS 3.1 INIT COMMAND |
| $A501 | -23295 | . | ADDRESS FOR DOS 3.1 NOMON COMMAND |
| $A50D | -23283 | . | ADDRESS FOR DOS 3.1 FP COMMAND |
| $A531 | -23247 | . | ADDRESS FOR DOS 3.1 INT COMMAND |
| $A54F | -23217 | . | ADDRESS FOR DOS 3.1 EXEC COMMAND |
| $A566 | -23210 | . | ADDRESS FOR DOS 3.1 POSITION COMMAND |
| $A7E0-$A863 | -22560--22439 | . | DOS COMMAND TABLE |
| $A8CD-$A980 | -22323--22144 | . | DOS ERROR MSG TABLE |
| $A996-$A997 | -22122--22121 | . | DOS INTERNAL HOOK ADDRESS TO OUTPUT A CHARACTER |
| $A998-$A999 | -22120--22119 | . | DOS INTERNAL HOOK ADDRESS TO INPUT A CHARACTER |
| $A9A3-$A9A4 | -22109--22108 | . | LENGTH OF BLOADED FILE |
| $A9B5-$A9B6 | -22091--22090 | . | STARTING ADDRESS OF BLOADED FILE |
| $AA0B | -22005 | . | START OF LIST OF POINTERS TO SECTIONS OF DOS 3.1 I/O PACKAGES |
| $AA3F-$B2CE | -21953--19762 | . | DOS 3.1 I/O PACKAGE |
| $B3EF-$B642 | -19473--18878 | . | DOS 3.1 SYSTEM BUFFER (FOR CATALOG ETC.) |
| $BD00 | -17152 | . | ROUTINE WHICH READS IN DIRECTORY OFF DISK |
| $BF6 | | . | VOL NO OF CURRENT DISK |
| $BFFF | -16384 | . | HIGHEST RAM MEMORY ADDRESS |
| $BFFF | -16384 | . | DEFAULT INTEGER BASIC HIMEM (W/O DOS~ 48K MACHINE) |
| $C000 | -16384 | KBD ~ IOADR | READ KEYBOARD. IF VAL>127 THEN KEY WAS PRESSED |
| $C000-$C00F | -16384--16369 | . | KEYBOARD INPUT SUBROUTINE |
| $C000-$CFFF | -16384--12289 | . | ADDRESSES DEDICATED TO HARDWARE FUNCTION |
| $C010 | -16368 | KBDSTB | CLEAR KEYBOARD STROBE. POKE 0 AWAYS AFTER READING KBD. |
| $C010-$C01F | -16368--16353 | . | CLEAR KEYBOARD STROBE SUBROUTINE |
| $C020 | -16352 | TAPEOUT | MONITOR MEMORY LOCATION 'TAPEOUT' |
| $C02X | -16352 | . | TOGGLE CASSETTE OUTPUT |
| $C030 | -16336 | SPKR | PEEK TO TOGGLE SPEAKER |
| $C04X | -16320 | . | OUTPUT STROBE TO GAME I/O CONNECTOR |
| $C050 | -16304 | TXTCLR | POKE TO 0 TO SET GRAPHICS MODE |
| $C051 | -16303 | TXTSET | POKE 0 TO SET TEXT MODE |
| $C052 | -16302 | MIXCLR | POKE 0 TO SET BOTTOM 4 LINES TO GRAPHICS |
| $C053 | -16301 | MIXSET | POKE=0 TO SELECT TEXT/GRAPHICS MIX (BOTTOM 4 LINES TEXT) |
| $C054 | -16300 | LOWSCR | POKE TO 0 TO DISPLAY PRIMARY PAGE (PAGE 1) |
| $C055 | -16299 | HISCR | POKE TO 0 TO DISPLAY SECONDARY PAGE (PAGE2) |
| $C056 | -16298 | LORES | POKE TO 0 TO SET LO-RES GRAPHICS |
| $C057 | -16297 | HIRES | POKE TO 0 TO SET HI-RES GRAPHICS |

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $C058 | -16296 | . | POKE 0 TO CLEAR GAME I/O OUTPUT AN0 |
| $C059 | -16295 | . | POKE 0 TO SET GAME I/O OUTPUT AN0 |
| $C05A | -16294 | . | POKE 0 TO CLEAR GAME I/O OUTPUT AN1 |
| $C05B | -16293 | . | POKE 0 TO SET GAME I/O OUTPUT AN1 |
| $C05C | -16292 | . | POKE 0 TO CLEAR GAME I/O OUTPUT AN2 |
| $C05D | -16291 | . | POKE 0 TO SET GAME I/O OUTPUT AN2 |
| $C05E | -16290 | . | POKE 0 TO CLEAR GAME I/O OUTPUT AN3 |
| $C05F | -16289 | . | POKE 0 TO SET GAME I/O OUTPUT AN3 |
| $C060 | -16288 | TAPEIN | MONITOR MEMORY LOCATION 'TAPEIN' |
| $C060/8 | -16288 | . | STATE OF 'CASSETE DATA IN' APPEARS IN BIT 7 |
| $C061 | -16287 | . | PEEK TO READ PDL(0). IF >127 SWITCH ON |
| $C062 | -16286 | . | PEEK TO READ PDL(1) PUSH BUTTON SWITCH |
| $C063 | -16285 | . | PEEK TO READ PDL(2) PUSH BUTTON SWITCH |
| $C064 | -16188 | PADDLO | MONITOR MEMORY LOCATION PADDLO |
| $C064/C | -16188 | . | STATE OF TIMER OUTPUT FOR PADDLE 1 APPEARS IN BIT 7 |
| $C065/D | -16187 | . | STATE OF TIMER OUTPUT FOR PADDLE 1 APPEARS IN BIT 7 |
| $C066/E | -16186 | . | STATE OF TIMER OUTPUT FOR PADDLE 2 APPEARS IN BIT 7 |
| $C067/F | -16185 | . | STATE OF TIMER OUTPUT FOR PADDLE 3 APPEARS IN BIT 7 |
| $C070 | -16272 | PTRIG | MONITOR MEMORY LOCATION 'PTRIG' (PADDLE TRIGGER) |
| $C07X | -16272 | PTRIG | TRIGGERS PADDLE TIMERS DURING PHI-2 |
| $C08X | -16256 | . | DEVICE SELECT 0 |
| $C09X | -16240 | . | DEVICE SELECT 1 |
| $C0AX | -16224 | DEVICE SELECT 2 | DEVICE SELECT 2 |
| $C0BX | -16208 | . | DEVICE SELECT 3 |
| $C0CX | -16192 | . | DEVICE SELECT 4 |
| $C0DX | -16176 | . | DEVICE SELECT 5 |
| $C0E8 | -16152 | . | ADDRESS TO POWER DOWN DISK IN SLOT 6 |
| $C0E9 | -16151 | . | ADDRESS TO POWER UP DISK IN SLOT 6 |
| $C0EX | -16160 | . | DEVICE SELECT 6 |
| $C0FX | -16144 | . | DEVICE SELECT 7 |
| $C100 | -16128 | . | CALL -16128 IS EQUIVALENT TO PR#1 FOR INITIALIZING SERIAL INTERFACE |
| $C100 | -16128 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #1 |
| $C200 | -15842 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #2 |
| $C300 | -15616 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #3 |
| $C400 | -15360 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #4 |
| $C500 | -15104 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #5 |
| $C600 | -14848 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #6 |
| $C700 | -14592 | . | STANDARD CHARACTER I/O SUBROUTINE ENTRY POINT FOR SLOT #6 |
| $C800-$CFFF | -14336--12289 | . | PIN 20 ON ALL PERIPH CONCTRS GOES LOW DURING PHI0 ON READ OR WRITE |
| $C93D | -14109 | . | SERIAL INTERFACE BATCH INPUT ROUTINE. A1&A2 SPECIFY MEMORY RANGE |
| $C941 | -14105 | . | SERIAL INTERFACE BATCH OUTPUT ROUTINE - A1 & A2 SPECIFY MEMORY RANGE |
| $CS00 | -16384+256*S | . | TRANSMIT ASCII CHAR IN ACCUMULATOR OUT VIA SERIAL INTERFACE IN SLOT S |
| $D000 | -12288 | SETHRL | HI-RES GRAPHICS INIT S/R CALL (ROM VERSION) |
| $D000-$D3FF | -12288--11265 | . | HI-RES GRAPHICS ROM |
| $D000-$D7FF | -12288--10241 | . | ROM SOCKET D0 |
| $D00E | -12274 | HCLR | HI-RES GRAPHICS CLEAR S/R CALL |
| $D010 | -12272 | BKGND0 | HI-RES GRAPHICS 'BKGND0 (HCOLOR1 SET FOR BLACK BKGND) |
| $D012 | -12270 | BKGND | HI-RES GRAPHICS MEMORY LOCATION 'BKGND' (ROM) |
| $D1FC | -11780 | . | HI-RES GRAPHICS FIND S/R CALL: PARAM=SHAPE~ROT~SCALE |
| $D2F9 | -11527 | . | HI-RES GRAPHICS POSN S/R CALL PARAM= X0~Y0~COLR |
| $D30E | -11506 | . | HI-RES GRAPHICS PLOT S/R CALL PARAM= X0~Y0~COLR |
| $D314 | -11500 | . | HI-RES GRAPHICS LINE S/R CALL PARAM= X0~Y0~COLR |
| $D331 | -11471 | . | HI-RES GRAPHICS BKGND S/R CALL PARAM= COLR |
| $D337 | -11465 | . | HI-RES GRAPHICS LINE S/R CALL: PARAM=X0~Y0~COLR |
| $D33A | -11462 | . | HI-RES GRAPHICS DRAW1 S/R CALL: PARAM= X0~Y0~COLR~SHAPE~ROT~SCALE |
| $D3B9 | -11335 | . | HI-RES GRAPHICS SHLOAD S/R CALL |
| $D4BC | -11076 | . | INTEGER BASIC PA#1 APPEND PROGRAM ENTRY |
| $D4F2 | -11022 | . | TO CONVERT A/S FM CASSETTE TO ROM- LD FM CASS~CALL -11022~LIST~SAVE |
| $D535 | -10955 | . | INTEGER BASIC PA#1 TAPE VERIFY PROG ENTRY |
| $D6DD | -10531 | . | INTEGER BASIC PA#1 RENUMBER PROG ENTRY (WHOLE PROG) |
| $D6E7 | -10521 | . | INTEGER BASIC PA#1 RENUMBER PROG ENTRY (PART PROG) |
| $D717 | -10473 | . | INTEGER BASIC PA#1 MUSIC PROG ENTRY |
| $D800-$DFFF | -10240--8193 | . | ROM SOCKET D8 |
| $DD67 | -8867 | . | FRMNUM S/R. EVALS FORMULA EXP. INTO FLOATING PT ACCUM |
| $DEC9 | -8503 | . | SNERR S/R. PRINTS "SYNTAX ERROR" AND HALTS PROG |
| $E000 | -8192 | BASIC | ENTER INTEGER BASIC |
| $E000-$E7FF | -8192--6145 | . | ROM SOCKET E0 (INTEGER BASIC) |
| $E003 | -8189 | BASIC2 | ENTRY 2 OF INTEGER BASIC |
| $E36B | -7317 | MEMFUL | INTEGER BASIC MEMORY FULL ERROR |
| $E51B | -6885 | . | INTEGER BASIC DECIMAL LPRINT S/R |
| $E6FB | -6408 | . | GETBYT S/R. EVALS FORMULA & CONVTS TO 1-BYT VAL IN X REG |
| $E800-$EFFF | -6144--4097 | . | ROM SOCKET E8 (INTEGER BASIC) |
| $EE68 | -4504 | RNGERR | INTEGER BASIC RANGE ERROR |
| $F000-$F7FF | -4096--2049 | . | ROM SOCKET F0 (1K INTEGER BASIC~ 1 K MONITOR) |
| $F11E | -3810 | ACADR | HI-RES GRAPHICS 2-BYTE TAPE READ SETUP |
| $F666 | -2458 | . | TURN ON MINIASSEMBLER |
| $F689 | -2423 | . | SWEET-16 INTERPRETER ENTRY |
| $F800 | -2048 | PLOT | MONITOR S/R PLOT A POINT (LO-RES) AC:Y-COORD Y:X-COORD |
| $F800 | -2048 | PLOT | MONITOR S/R PLOT A POINT. AC:Y-COORD~Y:X-COORD |
| $F800-$FFFF | -2048--1 | . | ROM SOCKET F8 (MONITOR) |

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $F80C | -2036 | RTMASK | MONITOR MEMORY LOCATION 'RTMASK' |
| $F80E | -2034 | PLOT1 | MONITOR MEMORY LOCATION 'PLOT1' |
| $F819 | -2023 | . | HLINE S/R (SEE CALL-APPLE NOV/DEC 78 PG4) |
| $F819 | -2023 | HLINE | MONITOR S/R TO DRAW A HORIZONTAL LINE (LO-RES) |
| $F81C | -2020 | HLINE1 | MONITOR MEMORY LOCATION 'HLINE1' |
| $F826 | -2010 | VLINEZ | MONITOR MEMORY LOCATION 'VLINEZ' |
| $F828 | -2008 | VLINE | DRAW A VERTICAL LINE |
| $F831 | -1999 | RTS1 | MONITOR MEMORY LOCATION 'RTS1' |
| $F832 | -1998 | CLRSCR | CLEAR SCREEN - GRAPHICS MODE |
| $F832 | -1998 | CLRSCR | CLEAR LOW RES GRAPHICS SCREEN1 |
| $F836 | -1994 | CLRTOP | MONITOR MEMORY LOCATION 'CLRTOP' |
| $F838 | -1992 | CLRSC2 | MONITOR MEMORY LOCATION 'CLRSC2' |
| $F83C | -1988 | CLRSC3 | MONITOR MEMORY LOCATION 'CLRSC3' |
| $F847 | -1977 | GBASCALC | MONITOR S/R TO CALCULATE GRAPHICS BASE ADDRESS |
| $F856 | -1962 | GBCALC | MONITOR MEMORY LOCATION 'GBCALC' |
| $F85F | -1953 | NXTCOL | MONITOR S/R - INCREMENT COLOR BY 3 |
| $F864 | -1948 | SETCOL | MONITOR S/R TO ADJUST COLOR BYTE FOR BOTH HALVES EQUAL |
| $F871 | -1935 | SCRN | SCRN S/R (LO-RES GRAPHICS)(SEE CALL-APPLE DEC78) |
| $F871 | -1935 | SCRN | MONITOR S/R TO GET SCREEN COLOR. AC:Y-COORD~Y:X-COORD |
| $F879 | -1927 | SCRN2 | MONITOR MEMORY LOCATION 'SCRN2' |
| $F87F | -1921 | RTMSKZ | MONITOR MEMORY LOCATION 'RTMSKZ' |
| $F882 | -1918 | INSDS1 | MONITOR MEMORY LOCATION 'INSDS1' |
| $F88E | -1906 | INSDS2 | MONITOR S/R - DISASSEMBLER ENTRY |
| $F89B | -1893 | IEVEN | MONITOR MEMORY LOCATION 'IEVEN' |
| $F8A5 | -1883 | ERR | MONITOR MEMORY LOCATION 'ERR' |
| $F8A9 | -1879 | GETFMT | MONITOR MEMORY LOCATION GETFMT |
| $F8BE | -1858 | MNNDX1 | MONITOR MEMORY LOCATION 'MNNDX1' |
| $F8C2 | -1854 | MNNDX2 | MONITOR MEMORY LOCATION 'MNNDX2' |
| $F8C9 | -1847 | MNNDX3 | MONITOR MEMORY LOCATION 'MNNDX3' |
| $F8D0 | -1840 | INSTDSP | MONITOR & MINIASSEMBLER MEMORY LOCATION 'INSTDSP' |
| $F8D4 | -1836 | PRNTOP | MONITOR MEMORY LOCATION 'PRNTOP' |
| $F8DB | -1829 | PRNTBL | MONITOR MEMORY LOCATION 'PRNTBL' |
| $F8F5 | -1803 | PRMN1 | MONITOR MEMORY LOCATION 'PRMN1' |
| $F8F9 | -1799 | PRMN2 | MONITOR MEMORY LOCATION 'PRMN2' |
| $F910 | -1776 | PRADR1 | MONITOR MEMORY LOCATION 'PRADR1' |
| $F914 | -1772 | PRADR2 | MONITOR MEMORY LOCATION 'PRADR2' |
| $F926 | -1754 | PRADR3 | MONITOR MEMORY LOCATION 'PRADR3' |
| $F92A | -1750 | PRADR4 | MONITOR MEMORY LOCATION 'PRADR4' |
| $F930 | -1744 | PRADR5 | MONITOR MEMORY LOCATION 'PRADR5' |
| $F938 | -1736 | RELADR | MONITOR MEMORY LOCATION 'RELADR' |
| $F940 | -1728 | PRNTYX | MONITOR S/R- PRINT CONTENTS OF Y AND X AS 4 HEX DIGITS |
| $F941 | -1727 | PRNTAX | MONITOR MEMORY LOCATION 'PRNTAX' |
| $F944 | -1724 | PRNTX | MONITOR MEMORY LOCATION 'PRNTX' |
| $F948 | -1720 | PRBLNK | MONITOR MEMORY LOCATION 'PRBLNK' |
| $F94C | -1716 | PRBL2 | MONITOR S/R- PRINT BLANKS: X REG CONTAINS NUMBER TO PRINT. |
| $F94C | . | PRBL3 | MONITOR MEMORY LOCATION 'PRBL3' |
| $F953 | -1709 | PCADJ | MINIASSEMBLER MEMORY LOCATION 'PCADJ' |
| $F954 | -1708 | PCADJ2 | MONITOR & MINIASSEMBLER MEMORY LOCATION 'PCADJ2' |
| $F956 | -1706 | PCADJ4 | MONITOR MEMORY LOCATION 'PCADJ4' |
| $F961 | -1695 | RTS2 | MONITOR MEMORY LOCATION 'RTS2' |
| $F962 | -1694 | FMT1 | MONITOR MEMORY LOCATION 'FMT1' |
| $F9A6 | -1626 | FMT2 | MONITOR MEMORY LOCATION 'FMT2' |
| $F9B4 | -1612 | CHAR1 | MONITOR & MINIASSEMBER MEMORY LOCATION 'CHAR1' |
| $F9BA | -1606 | CHAR2 | MONITOR & MINIASSEMBLER MEMORY LOCATION 'CHAR2' |
| $F9C0 | -1600 | MNEML | MONITOR & MINIASSEMBLER MEMORY LOCATION 'MNEML' |
| $FA00 | -1536 | MNEMR | MONITOR & MINIASSEMBER MEMORY LOCATION 'MNEMR' |
| $FA43 | -1469 | STEP | MONITOR S/R- PERFORM A SINGLE STEP |
| $FA4E | -1458 | XQINIT | MONITOR MEMORY LOCATION 'XQINIT' |
| $FA78 | -1416 | XQ1 | MONITOR MEMORY LOCATION 'XQ1' |
| $FA7A | -1414 | XQ2 | MONITOR MEMORY LOCATION 'XQ2' |
| $FA86 | -1402 | IRQ | MONITOR S/R- IRQ HANDLER |
| $FA92 | -1390 | BREAK | MONITOR S/R - BREAK HANDLER |
| $FA9C | -1380 | XBRK | MONITOR MEMORY LOCATION 'XBRK' |
| $FAA5 | -1371 | XRTI | MONITOR MEMORY LOCATION 'XRTI' |
| $FAA9 | -1367 | XRTS | MONITOR MEMORY LOCATION 'XRTS' |
| $FAAD | -1363 | PCINC2 | MONITOR MEMORY LOCATION 'PCINC2' |
| $FAAF | -1361 | PCINC3 | MONITOR MEMORY LOCATION 'PCINC3' |
| $FAB9 | -1351 | XJSR | MONITOR MEMORY LOCATION 'XJSR' |
| $FAC4 | -1340 | XJMP | MONITOR MEMORY LOCATION 'XJMP' |
| $FAC5 | -1339 | XJMPAT | MONITOR MEMORY LOCATION 'XJMPAT' |
| $FACD | -1331 | NEWPCL | MONITOR MEMORY LOCATION 'NEWPCL' |
| $FAD1 | -1327 | RTNJMP | MONITOR MEMORY LOCATION 'RTNJMP' |
| $FAD7 | -1321 | REGDSP | MONITOR S/R TO DISPLAY USER REGISTERS |
| $FADA | -1318 | RGDSP1 | MONITOR MEMORY LOCATION 'RGDSP1' |
| $FAE4 | -1308 | RDSP1 | MONITOR MEMORY LOCATION 'RDSP1' |
| $FAFD | -1283 | BRANCH | MONITOR MEMORY LOCATION 'BRANCH' |
| $FB0B | -1269 | NBRNCH | MONITOR MEMORY LOCATION 'NBRNCH' |
| $FB11 | -1263 | INITBL | MONITOR MEMORY LOCATION 'INITBL' |
| $FB19 | -1255 | RTBL | MONITOR MEMORY LOCATION 'RTBL' |
| $FB1E | -1250 | PREAD | MONITOR S/R TO READ PADDLE. X-REG CONTAINS PADDLE NUMBER 0-3 |
| $FB25 | -1243 | PREAD2 | MONITOR MEMORY LOCATION 'PREAD2' |

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $FB2E | -1234 | RTS2D | MONITOR MEMORY LOCATION 'RTS2D' |
| $FB2F | -1233 | INIT | MONITOR S/R- SCREEN INITIALIZATION |
| $FB39 | -1223 | SETTXT | MONITOR S/R- SET SCREEN TO TEXT MODE. CLOBBERS ACCUMULATOR |
| $FB40 | -1216 | SETGR | MONITOR S/R- SET GRAPHIC MODE (GR). CLOBBERS ACCUMULATOR |
| $FB4B | -1205 | SETWND | MONITOR S/R- SET NORMAL WINDOW |
| $FB5B | -1189 | TABV | MONITOR MEMORY LOCATION 'TABV' |
| $FB60 | -1184 | MULPM | MONITOR MEMORY LOCATION 'MULPM' |
| $FB63 | -1181 | MUL | MONITOR S/R- MULTIPLY ROUTINE |
| $FB65 | -1179 | MUL2 | MONITOR MEMORY LOCATION 'MUL2' |
| $FB6D | -1171 | MUL3 | MONITOR MEMORY LOCATION 'MUL3' |
| $FB76 | -1162 | MUL4 | MONITOR MEMORY LOCATION 'MUL4' |
| $FB78 | -1160 | MUL5 | MONITOR MEMORY LOCATION 'MUL5' |
| $FB81 | -1151 | DIVPM | MONITOR MEMORY LOCATION 'DIVPM' |
| $FB84 | -1148 | DIV | MONITOR S/R- DIVIDE ROUTINE |
| $FB86 | -1146 | DIV2 | MONITOR MEMORY LOCATION 'DIV2' |
| $FBA0 | -1120 | DIV3 | MONITOR MEMORY LOCATION 'DIV3' |
| $FBA4 | -1116 | MD1 | MONITOR MEMORY LOCATION 'MD1' |
| $FBAF | -1105 | MD2 | MONITOR MEMORY LOCATION 'MD2' |
| $FBB4 | -1100 | MD3 | MONITOR MEMORY LOCATION 'MD3' |
| $FBC0 | -1088 | MDRTS | MONITOR MEMORY LOCATION 'MDRTS' |
| $FBC1 | -1087 | BASCALC | MONITOR S/R- CALCULATE TEXT BASE ADDRESS |
| $FBD0 | -1072 | BSCLC2 | MONITOR MEMORY LOCATION 'BSCLC2' |
| $FBD9 | -1063 | BELL1 | MONITOR MEMORY LOCATION 'BELL1' |
| $FBE4 | -1052 | BELL2 | MONITOR S/R- SOUND BELL (BEEPER) |
| $FBEF | -1041 | RTS2B | MONITOR MEMORY LOCATION 'RTS2B' |
| $FBF0 | -1040 | STOADV | MONITOR MEMORY LOCATION 'STOADV' |
| $FBF4 | -1036 | ADVANCE | MONITOR S/R- MOVE CURSOR RIGHT |
| $FBFC | -1028 | RTS3 | MONITOR MEMORY LOCATION 'RTS3' |
| $FBFD | -1027 | VIDOUT | MONITOR S/R- OUTPUT A-REGISTER AS ASCII ON TEXT SCREEN 1 |
| $FC10 | -1008 | BS | MONITOR S/R TO MOVE CURSOR LEFT (BACKSPACE) |
| $FC1A | -998 | UP ~ CURSUP | MONITOR S/R TO CURSOR UP |
| $FC22 | -990 | VTAB | MONITOR S/R- PERFORM A VERTICAL TAB TO ROW SPECIFIED IN ACCUM ($0-$17) |
| $FC24 | -988 | VTABZ | MONITOR MEMORY LOCATION 'VTABZ' |
| $FC2B | -981 | RTS4 | MONITOR MEMORY LOCATION 'RTS4' |
| $FC2C | -980 | ESC1 | MONITOR S/R- PERFORM ESCAPE FUNCTIONS |
| $FC42 | -958 | CLREOP | MONITOR S/R TO CLEAR FROM CURSOR TO END OF PAGE. CLOBBERS ACC & Y-REG |
| $FC46 | -954 | CLEOP1 | MONITOR MEMORY LOCATION 'CLEOP1' |
| $FC58 | -936 | HOME | MONITOR S/R TO HOME CURSOR & CLEAR SCREEN. CLOBBERS ACCUM & Y-REG |
| $FC62 | -926 | CR | MONITOR S/R TO PERFORM A CARRIAGE RETURN |
| $FC66 | -922 | LF | MONITOR S/R TO TO PERFORM A LINE FEED |
| $FC70 | -912 | SCROLL | MONITOR S/R TO SCROLL UP 1 LINE. CLOBBERS ACCUM & Y-REG |
| $FC76 | -906 | SCRL1 | MONITOR MEMORY LOCATION 'SCRL1' |
| $FC8C | -884 | SCRL2 | MONITOR MEMORY LOCATION 'SCRL2' |
| $FC95 | -875 | SCRL3 | MONITOR MEMORY LOCATION 'SCRL3' |
| $FC9C | -868 | CLREOL | MONITOR S/R TO CLEAR TO END OF LINE |
| $FC9E | -866 | CLEOLZ | MONITOR MEMORY LOCATION 'CLEOLZ' |
| $FCA0 | -864 | CLEOL2 | MONITOR MEMORY LOCATION 'CLEOL2' |
| $FCA8 | -856 | WAIT | CALL FOR WAIT LOOP |
| $FCA9 | -855 | WAIT2 | MONITOR MEMORY LOCATION 'WAIT2' |
| $FCAA | -854 | WAIT3 | MONITOR MEMORY LOCATION 'WAIT3' |
| $FCB4 | -844 | NXTA4 | MONITOR S/R TO INCREMENT A4 (16 BITS) THEN DO NXTA1 |
| $FCBA | -838 | NXTA1 | MONITOR S/R TO INCREMENT A1 (16 BITS). SETT CARRY IF RESULT >=A2. |
| $FCC8 | -824 | RTS4B | MONITOR MEMORY LOCATION 'RTS4B' |
| $FCC9 | -823 | HEADR | MONITOR MEMORY LOCATION 'HEADR' |
| $FCD6 | -810 | WRBIT | MONITOR MEMORY LOCATION 'WRBIT' |
| $FCDB | -805 | ZERDLY | MONITOR MEMORY LOCATION 'ZERDLY' |
| $FCE2 | -798 | ONEDLY | MONITOR MEMORY LOCATION 'ONEDLY' |
| $FCE5 | -795 | WRTAPE | MONITOR MEMORY LOCATION 'WRTAPE' |
| $FCEC | -798 | RDBYTE | MONITOR MEMORY LOCATION 'RDBYTE' |
| $FCEE | -786 | RDBYT2 | MONITOR MEMORY LOCATION 'RDBYT2' |
| $FCFA | -774 | RD2BIT | MONITOR TWO-EDGE TAPE SENSE |
| $FCFD | -771 | RDBIT | MONITOR MEMORY LOCATION 'RDBIT' |
| $FD0C | -756 | RDKEY | GET KEY INPUT FROM THE KEYBOARD. CLOBBERS ACC ~ Y-REG |
| $FD1B | -741 | KEYIN | MONITOR S/R- MONITOR KEYIN ROUTINE |
| $FD21 | -735 | KEYIN2 | MONITOR MEMORY LOCATION KEYIN2 |
| $FD2F | -721 | ESC | MONITOR MEMORY LOCATION 'ESC' |
| $FD35 | -715 | RDCHAR | CALL TO READ KEY & PERFORM ESCAPE FUNCTION IF NECESSARY. |
| $FD3D | -707 | NOTCR | MONITOR MEMORY LOCATION 'NOTCR' |
| $FD5F | -673 | NOTCR1 | MONITOR MEMORY LOCATION 'NOTCR1' |
| $FD62 | -670 | CANCEL | MONITOR S/R TO PERFORM A LINE CANCEL (\) |
| $FD67 | -665 | GETLNZ | MONITOR S/R TO PERFORM CARRIAGE RETURN AND GET A LINE OF TEXT |
| $FD6A | -662 | GETLN | MONITOR S/R TO GET LINE OF TEXT FROM KEYBD. X RETND W/ # OF CHARS |
| $FD71 | -655 | BCKSPC | MONITOR MEMORY LOCATION 'BCKSPC' |
| $FD75 | -651 | NXTCHAR | MONITOR MEMORY LOCATION 'NXTCHAR' |
| $FD7E | -642 | CAPTST | MONITOR MEMORY LOCATION 'CAPTST' |
| $FD80 | -640 | INSTDSP | MONITOR S/R TO DISASSEMBLE INSTRUCTION AT PCH/PCL |
| $FD84 | -636 | ADDINP | MONITOR MEMORY LOCATION 'ADDINP' |
| $FD8E | -626 | CROUT | MONITOR S/R TO PRINT A CARRIAGE RETURN. CLOBBERS ACC~ Y-REG |
| $FD92 | -622 | PRA1 | MONITOR MEMORY LOCATION 'PRA1' |
| $FD96 | -618 | PRYX2 | MONITOR MEMORY LOCATION 'PRYX2' |

| HEXLOC | DECLOC | NAME | USE |
|---|---|---|---|
| $FDA3 | -605 | XAM8 | MONITOR MEMORY LOCATION 'XAM8' |
| $FDAD | -595 | MOD8CHK | MONITOR MEMORY LOCATION 'MOD8CHK' |
| $FDB3 | -589 | XAM | MONITOR MEMORY LOCATION 'XAM' |
| $FDB6 | -586 | DATAOUT | MONITOR MEMORY LOCATION 'DATAOUT' |
| $FDC5 | -571 | RTS4C | MONITOR MEMORY LOCATION 'RTS4C' |
| $FDC6 | -570 | XAMPM | MONITOR MEMORY LOCATION 'XAMPM' |
| $FDD1 | -559 | ADD | MONITOR MEMORY LOCATION 'ADD' |
| $FDDA | -550 | PRBYTE | MONITOR S/R TO PRINT CONTENTS OF ACC AS 2 HEX DIGITS |
| $FDE3 | -541 | PRHEX | MONITOR S/R TO PRINT A HEX DIGIT |
| $FDE5 | -539 | PRHEXZ | MONITOR MEMORY LOCATION 'PRHEXZ' |
| $FDED | -531 | COUT | MONITOR S/R TO OUTPUT CHAR IN ACC. CLOBBERS ACC~Y-REG~COUT. |
| $FDF0 | -528 | COUT1 | MONITOR S/R TO GET MONITOR CHARACTER OUTPUT |
| $FDF6 | -522 | COUTZ | MONITOR MEMORY LOCATION 'COUTZ' |
| $FE00 | -512 | BL1 | MONITOR & MINIASSEMBLER MEMORY LOCATION 'BL1' |
| $FE04 | -508 | BLANK | MONITOR MEMORY LOCATION 'BLANK' |
| $FE0B | -501 | STOR | MONITOR MEMORY LOCATION 'STOR' |
| $FE17 | -489 | RTS5 | MONITOR MEMORY LOCATION 'RTS5' |
| $FE18 | -488 | SETMODE | MONITOR MEMORY LOCATION 'SETMODE' |
| $FE1D | -483 | SETMDZ | MONITOR MEMORY LOCATION 'SETMDZ' |
| $FE20 | -480 | LT | MONITOR MEMORY LOCATION 'LT' |
| $FE22 | -478 | LT2 | MONITOR MEMORY LOCATION 'LT2' |
| $FE2C | -468 | MOVE | MONITOR S/R TO PERFORM A MEMORY MOVE (A1-A2 TO A4) |
| $FE36 | -458 | VFY | MONITOR S/R TO PERFORM A MEMORY VERIFY |
| $FE58 | -424 | VFYOK | MONITOR MEMORY LOCATION 'VFYOK' |
| $FE5E | -418 | LIST | CALL TO DISASSEMBLE 20 INSTRUCTIONS |
| $FE63 | -413 | LIST2 | MONITOR MEMORY LOCATION 'LIST2' |
| $FE78 | -392 | A1PCLP | MONITOR & MINIASSEMBLER MEMORY LOCATION 'A1PCLP' |
| $FE7F | -385 | A1PCRTS | MONITOR MEMORY LOCATION 'A1PCRTS' |
| $FE80 | -384 | SETINV | MONITOR MEMORY LOCATION 'SETINV' |
| $FE84 | -380 | SETNORM | MONITOR MEMORY LOCATION 'SETNORM' |
| $FE86 | -378 | SETIFLG | MONITOR MEMORY LOCATION 'SETIFLG' |
| $FE89 | -375 | SETKBD | MONITOR MEMORY LOCATION 'SETKBD' |
| $FE8B | -373 | INPORT | MONITOR MEMORY LOCATION 'INPORT' |
| $FE8D | -371 | INPRT | MONITOR MEMORY LOCATION 'INPRT' |
| $FE93 | -365 | SETVID | MONITOR MEMORY LOCATION'SETVID' |
| $FE95 | -363 | OUTPORT | MONITOR MEMORY LOCATION 'OUTPORT' |
| $FE97 | -361 | OUTPRT | MONITOR MEMORY LOCATION 'OUTPRT' |
| $FE9B | -357 | IOPRT | MONITOR MEMORY LOCATION 'IOPRT' |
| $FEA7 | -345 | IOPRT1 | MONITOR MEMORY LOCATION 'IOPRT1' |
| $FEA9 | -343 | IOPRT2 | MONITOR MEMORY LOCATION 'IOPRT2' |
| $FEB0 | -336 | XBASIC | MONITOR S/R TO JUMP TO BASIC |
| $FEB3 | -333 | BASCONT | MONITOR S/R TO CONTINUE BASIC |
| $FEB6 | -330 | GO | MONITOR MEMORY LOCATION 'GO' |
| $FEBF | -321 | REGZ | MONITOR MEMORY LOCATION 'REGZ' |
| $FEC2 | -318 | TRACE | CALL TO PERFORM MONITOR TRACE |
| $FEC4 | -316 | STEPZ | MONITOR MEMORY LOCATION 'STEPZ' |
| $FECA | -310 | USR | MONITOR MEMORY LOCATION 'USR' |
| $FECD | -307 | WRITE | MONITOR S/R TO WRITE TO CASSETTE TAPE |
| $FED4 | -300 | WR1 | MONITOR MEMORY LOCATION 'WR1' |
| $FEED | -275 | WRBYTE | MONITOR MEMORY LOCATION 'WRBYTE' |
| $FEEF | -273 | WRBYT2 | MONITOR MEMORY LOCATION 'WRBYT2' |
| $FEF6 | -266 | CRMON | MONITOR MEMORY LOCATION 'CRMON' |
| $FEFD | -259 | READ | CALL TO READ FROM TAPE - LIMITS A1 & A2 |
| $FF02 | -254 | READX1 | HI-RES GRAPHICS - READ WITHOUT HEADER |
| $FF0A | -246 | RD2 | MONITOR MEMORY LOCATION 'RD2' |
| $FF16 | -234 | RD3 | MONITOR MEMORY LOCATION 'RD3' |
| $FF2D | -211 | PRERR | MONITOR S/R TO PRINT "ERR" AND SOUND BELL. CLOBBERS ACC & Y-REG |
| $FF3A | -198 | BELL | MONITOR S/R TO PRINT BELL. CLOBBERS ACC~ Y-REG |
| $FF3A | -198 | BELL | CALL HERE TO OUTPUT BELL |
| $FF3F | -193 | RESTORE | MONITOR & SWEET-16 MEMORY LOCATION 'RESTORE' |
| $FF44 | -188 | RESTR1 | MONITOR MEMORY LOCATION 'RESTR1' |
| $FF4A | -182 | SAVE | MONITOR & SWEET-16 MEMORY LOCATION 'SAVE' |
| $FF4C | -180 | SAV1 | MONITOR MEMORY LOCATION 'SAV1' |
| $FF59 | -167 | RESET | CALL HERE HAS SAME EFFECT AS PUSHING RESET BUTTON |
| $FF65 | -155 | MON | MONITOR S/R- NORMAL ENTRY TO 'TOP' OF MONITOR WHEN RUNNING |
| $FF69 | -151 | MONZ | MONITOR S/R TO RESET AND ENTER MONITOR |
| $FF73 | -141 | NXTITM | MONITOR MEMORY LOCATION 'NXTITM' |
| $FF7A | -134 | CHRSRCH | MONITOR MEMORY LOCATION 'CHRSRCH' |
| $FF7C | -132 | ZMODE | MONITOR & MINIASSEMBLER MEMORY LOCATION 'ZMODE' |
| $FF8A | -118 | DIG | MONITOR MEMORY LOCATION 'DIG' |
| $FF90 | -112 | NXTBIT | MONITOR MEMORY LOCATION 'NXTBIT' |
| $FF98 | -104 | NXTBAS | MONITOR MEMORY LOCATION 'NXTBAS' |
| $FFA2 | -94 | NXTBS2 | MONITOR MEMORY LOCATION 'NXTBS2' |
| $FFA7 | -89 | GETNUM | MONITOR & MINIASSEMBLER MEMORY LOCATION 'GETNUM' |
| $FFAD | -83 | NXTCHR | MONITOR MEMORY LOCATION 'NXTCHR' |
| $FFBE | -66 | TOSUB | MONITOR & MINIASSEMBLER MEMORY LOCATION 'TOSUB' |
| $FFC7 | -57 | ZMODE | MONITOR MEMORY LOCATION 'ZMODE' |
| $FFCC | -52 | CHRTBL | MONITOR & MINIASSEMBLER MEMORY LOCATION 'CHRTBL' |
| $FFE3 | -29 | SUBTBL | MONITOR MEMORY LOCATION 'SUBTBL' |
| ~1979/06/25 | VERSION | COPYRIGHT | PREPARED BY PROF W. F. LUEBBERT DARTMOUTH COLLEGE~ HANOVER N. H. |

# THE MICRO SOFTWARE CATALOG: XI

**Mike Rowe**
**P.O. Box 6502**
**Chelmsford, MA 01824**

Name: **APPLE-80**
System: **APPLE II**
Memory: **16K**
Language: **Integer BASIC (manual), Machine Language (APPLE-80 interpreter)**
Hardware: **Standard APPLE II, 16K,** game paddles for variable speed trace.

Description: With APPLE-80, your APPLE II RAM from 1000 HEX up becomes 8080 programming space. Single-Step or Trace with all 8080 registers dynamically displayed on APPLE's screen. When your 8080 program is fully de-bugged, let it run — you have full access to all APPLE I/O routines via the special C65 instruction, which also lets you call user-written 6502 subroutines directly from your 8080 program. 8080 I/O ports are arranged in a table for ease of user modification. Up to 8 non-destructive breakpoints may be set to facilitate program debugging. 8080 routines may also be imbedded in the middle of 6502 programs, saving tedious translation. Educators and students will benefit from APPLE-80's clear illustration of the inner workings of the 8080. APPLE-80 is suitable for all but time-dependent applications.

Copies: **45 +**
Price: **$20.00 + $1.50** Shipping & handling. California residents must add 6% sales tax.

Includes: APPLE-80 manual and APPLE-80 program on cassette, 8080 time-of-day clock demonstration program (illustrates use of APPLE II I/O from 8080 programs), and APPLE-80 ready reference card. Source NOT INCLUDED.

Order Info: Send Check or Money Order
Author: **Dann McCreary**
Available from:
Dann McCreary
Box 16435-M
San Diego, California 92116

Name: **FLEET**
System: **PET**
Memory: **8K**
Language: **Machine Language**
Hardware: **Standard Pet**

Description: FLEET is a game where the object is to find and destroy all of the enemy's ships. The program is designed to make optimum use of the features of the Commodore Pet, such as its graphic and sound producing capabilities. FLEET is written in machine language but has been specially recorded so that it can be loaded with the LOAD command, and it automatically runs after being loaded.

Copies: **Just Released**
Price: **$7.95**

Includes: Cassette with two versions of FLEET (one with sound effects and one without), manual, and instructions on how to hook up a music box to your PET.

Author: **William Robinson**

Available from:
PETRONICS
18431 Kingsport
Malibu, Ca. 90265

Name: **APPLESHIFT**
System: **APPLE II**
Memory: **16K** for tape version, **24K** for Disk II version
Language: **Integer BASIC and 6502 machine language**
Hardware: **APPLE II, tape recorder or Disk II, and printer**

Description: A package allowing conventional use of the APPLE II keyboard shift keys, containing instructions for hardware modification, machine language subroutines for input and display, an Integer BASIC demonstration program called TEXTPAGE, and complete documentation.

TEXTPAGE allows you to enter, edit, store on disk, and print (using your own printer driver) a page of text (55 lines of 80 characters each). The primary purpose of the package is to show you how to modify your apple and use our subroutines with your programs but TEXTPAGE functions nicely as a "mini" word processor. A complete word processor called APPLETEXT (the only complete word processor for the APPLE II to allow normal use of the shift keys!) is also available. Registered APPLESHIFT packages may be returned for complete credit toward APPLETEXT packages. Both products may be used with Dan Paymar's lower case adaptor or as stand-alone products, with lower case appearing on the screen as upper case in normal mode and upper case appearing as upper case in inverse mode.

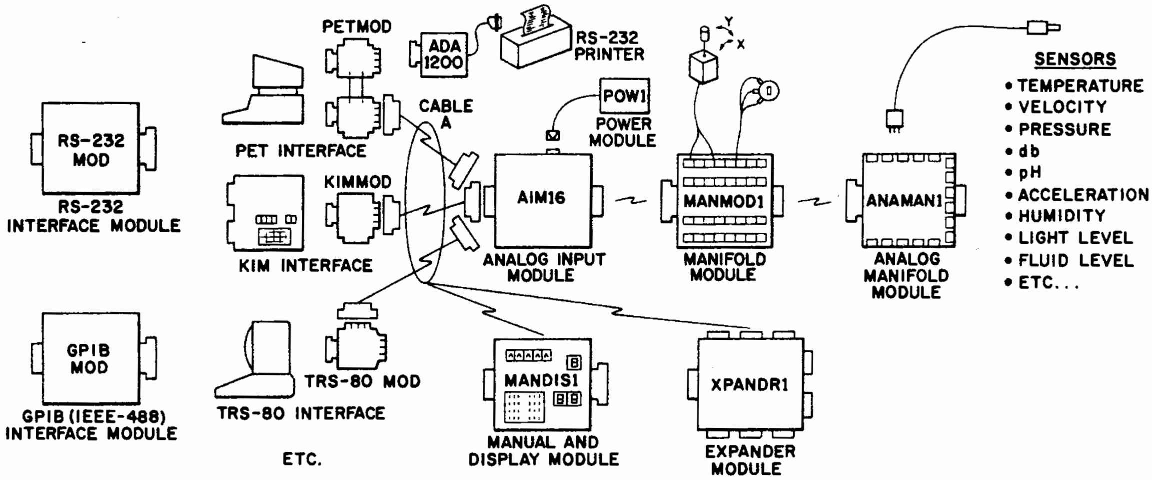
Copies: **Proprietary**
Price: **$29.95**
Includes: Complete documentation with listings, discussion, and instructions for hardware modification. Disk II version includes disk.

Author: **C&H Micro**
Available from:
C&H Micro
P.O. Box 2161
Glen Ellyn, Illinois 60137

# CONNECTICUT microCOMPUTER, Inc.

*150 POCONO ROAD · BROOKFIELD, CONNECTICUT 06804*

*(203) 775-9659*


DAM SYSTEMS by CmC
A complete system of modules to let your computer listen to the real world.

## DAM SYSTEMS PRICE LIST

### DAM SYSTEMS components

**AIM161 - Analog Input Module** — $179.00
16 8-bit analog inputs - 100 microsecond conversion time - 3 state output - requires one 8-bit computer output port for control and one 8-bit computer input port for data.

**AIM162 - Analog Input Module** — $249.00
As above plus: greater accuracy - gold plated contacts - pilot light - switch selectable start, enable and ready polarities.

**POW1 - Power Module** — $14.95
Supplies power for one AIM16 module.

**ICON - Input Connector** — $9.95
For connecting analog inputs to the AIM16 - 20 pin card edge connector - solder eyelets.

**OCON - Output Connector** — $9.95
For connecting the AIM16 to a computer - 20 pin card edge connector - solder eyelets.

**MANMOD1 - Manifold Module** — $59.95
Use in place of ICON. Screw terminal barrier strips for connecting joysticks, potentiometers, voltage sources, etc. Eliminates the need for soldering. Plugs into the AIM16.

**ANAMAN1 - Analog Manifold Module** — TBA
Use in place of ICON. Connects DAM SYSTEMS SENSORS to the AIM16 without soldering - sensor cables just plug in. Plugs into the AIM16 or the MANMOD1.

**SENSORS** — TBA
Sensors for temperature, pressure, flow, humidity, level, pH, motion, etc.

**COMPUTER INTERFACES** — TBA
For the PET, KIM, TRS-80, etc. Use in place of OCON. Eliminates the need for soldering or special construction.

**PETMOD - PET Interface Module** — $49.95
Gives two IEEE ports, one user port and one DAM SYSTEMS interface port. Saves wear and tear on the PET's printed circuit board. Also called the PETSAVR.

**KIMMOD - KIM Interface Module** — $39.95
Gives one application connector port and one DAM SYSTEMS interface port.

**CABLE "A" - Interconnect Cables** — TBA
Connects computer interface to AIM16, MANDIS1, XPANDR1, etc.

**CABLE A24 - Interconnect Cable** — $19.95
24 inch cable with interface connector on one end and an OCON equivalent on the other.

**MANDIS1 - Manual and Display Module** — TBA
Connects between the AIM16 and the computer interface. Allows manual or computer control of the AIM16. Displays channel number and data.

**GPIB MOD - GPIB (IEEE-488) Interface** — TBA
Allows the DAM SYSTEMS MODULES to be used with the GPIB bus instead of a computer's other I/O ports.

**RS232 MOD - RS232 Interface Module** — TBA
Allows the DAM SYSTEMS MODULES to be used with an RS-232 port or terminal.

**XPANDR1 - Expander Module** — TBA
Allows up to 128 8-bit analog inputs (8 AIM16 Modules) to be connected to one system.

### DAM SYSTEMS sets

**AIM161 Starter Set** — $189.00
Includes one AIM161, one POW1, one ICON and one OCON.

**AIM162 Starter Set** — $259.00
Includes one AIM162, one POW1, one ICON and one OCON.

**PETSET1a** — $295.00
Includes one PETMOD, one CABLE A24, one AIM161, one POW1 and one MANMOD1.

**KIMSET1a** — $285.00
Includes one KIMMOD, one CABLE A24, one AIM161, one POW1 and one MANMOD1.

# Interfacing the Analog Devices 7570J A/D Converter

Dr. Marvin L. De Jong
Department of Mathematics and Physics
The School of the Ozarks
Point Lookout, MO 65726

**Complete interfacing information, including a demonstration program, will make real time applications responsive to external events when you add this top of the line analog-to-digital converter to a 6502 system.**

If you want to go first class in analog-to-digital converters, you ought to consider the AD-7570J marketed by Analog Devices, 1 Industrial Park, Box 280, Norwood, MA 02062. It is a 28 pin, monolithic CMOS 8-bit successive approximation A/D converter specifically designed for interfacing with microprocessors. The data lines are three-state lines, and consequently may be connected to the data bus of a microcomputer.

An interface between a 6530 PIA and the 7570 is described in this article. In the near future, I hope to describe an interface directly to the data bus of a 6502 system. A demonstration program to control the A/D converter is also given. The interface circuitry and program should be applicable to any 6502 system with a MOS PIA, such as the 6530, or a VIA such as the 6522.

The circuit is shown in the figure. It differs from the one given on the 7570 specification sheet, supplied with the chip, only in the comparator which was used. I used an LM318 op amp simply because I did not have a 311 comparator handy. The AD311 or LM311 is recommended because it was designed for voltage comparisons, whereas the LM318 is a high-class op amp.

The 7570 has an internal clock which can be used by adding a resistor-capacitor network, but I chose to use the clock signal from the 6502 (either $0_2$ or $0_1$) which was divided by ten using the 7490. This arrangement gives the necessary phase relationship between the CLK and the STRT signals on the 7570.

A Zener diode provided the necessary reference voltage. STRT, BSEN, LBEN and HBEN are active high control signals. Since the 6502/6530 "comes up" with highs on the output ports, I used a 74004 inverter between the control port PB0-2 and the control inputs on the 7570. The CMOS version of the 7404 is not necessary; a 74L04, LS04, or just a plain old 7404 may be used. The CMOS version of the 7490 should not be used in the divide-down circuit because of propogation delays which might destroy the necessary phase relationships.

So much for the circuit. The reader is urged to study the 7570 spec sheet for additional details. Bipolar operation is possible, for example, and details regarding settling time, layout, and grounding are also quite important.

Conversion is initiated by applying a positive pulse to the STRT pin. The pulse must be at least 500 nanoseconds in duration, and conversion begins on the trailing edge of the pulse. The BSEN pin next receives a logical 1 from the computer. This is an interrogation signal. If the converter is still busy, the $\overline{\text{BUSY}}$ pin is low, putting a zero on the PA7 line. If the conversion is complete, a one will appear on the PA7 line. If the BSEN pin is low, the $\overline{\text{BUSY}}$ pin is in its high impedance state.

Once the conversion is complete, BSEN is brought low, and HBEN and LBEN are placed at logic 1 by the microcomputer. This results in the conversion data appearing at pins DB2-9 to be read by the A-port on the 6530. While HBEN and LBEN are low, the data pins are in their high impedance state.

The reason for having both LBEN and HBEN is simply that a ten bit version of the same chip (7507L) is available, and HBEN puts the two highest bits on the bus, while LBEN puts the low order bits on the bus. This also explains why DB0 and DB1 are not used. The ten bit version is also more expensive.

The program, while written for the KIM-1, demonstrates how the 6502 microprocessor and 6530 PIA control the A/D converter. The comments cover the details quite well. Clearly, the machine language details will be different for a system other than the KIM-1, but the mnemonics will remain the same.

What might you do with an A/D converter? If you are a game nut, you might attach the ANALOG IN signal to the center tap of a pot and call it a joy stick, I think. You want two, three, four joysticks? Don't get four of these expensive A/D converters; get an analog multiplexer such as the 4052.

Use the same device and the same reference (Lancaster) to build a programmable digital voltmeter. Speech recognition circuits convert the filtered and rectified voice signal to a digital value using A/D converters. Here is a real opportunity to help the seriously handicapped person.

Get a pressure transducer and use your A/D converter to monitor pulse rates and measure blood pressure automatically. Processing analog signals with digital techniques, averaging, filtering, etc. is also an interesting area for experimentation. Finally, document your experiment and send it away to be published in one of the hobby magazines, such as MICRO, so the rest of us can benefit from your work.

Reference:


Lancaster, D., *CMOS Cookbook*, Howard W. Sams & Co., Inc., Indianapolis, 1977.

SPEECHLAB™, Heuristics, Inc., 900 N. San Antonio Rd., Los Altos, CA 94022.

*Pressure Transducer Handbook*, National Semiconductor Corp., Santa Clara, CA 95051.

*Analog-Digital Conversion Handbook*, Analog Devices, Norwood, MA 02062, 1972.


```
MICRO-WARE ASSEMBLER 65XX-1.0 PAGE 01

                       * A/D CONVERTER DEMONSTRATION PROGRAM
                       * MODIFIED 7/4/79 BY MICRO STAFF
032D                   SCANDS *     $1F1F
032D                   PAD    *     $1700
032D                   PBD    *     $1702
032D                   PBDD   *     $1703
032D                   INH    *     $00F9
0300                          ORG   $0300
0300 A9 07             START  LDAIM $07    A/D CONTROL PINS SET TO
0302 8D 02 17                 STA   PBD    LOGICAL 0 VIA PB0-2 WHEN
0305 8D 03 17                 STA   PBDD   DIRECTION REGISTER IS ALSO SET
0308 CE 02 17          AGN    DEC   PBD    TOGGLE STRT PIN TO INITIATE
030B EE 02 17                 INC   PBD    CONVERSION
030E A9 05                    LDAIM $05    ACTIVATE BSEN TO CHECK BUSY
0310 8D 02 17                 STA   PBD
0313 AD 00 17          BACK   LDA   PAD    CHECK BIT 7 ON PAD (BUSY) TO
0316 10 FB                    BPL   BACK   SEE IF CONVERSION IS COMPLETE
0318 A9 03                    LDAIM $03    SET HBEN & LBEN TO LOGIC 1 TO
031A 8D 02 17                 STA   PBD    PUT DATA ON THE LINES
031D AD 00 17          FINISH LDA   PAD    DIGITAL DATA IS NOW IN
0320 85 F9                    STA   INH    ACCUMULATOR KIM-1 USERS MAY
0322 20 1F 1F                 JSR   SCANDS WISH TO DISPLAY THE RESULT
0325 A9 07                    LDAIM $07    INITIALIZE CONTROL PINS TO ZERO
0327 8D 02 17                 STA   PBD    AND THEN
032A 4C 08 03          PRGEND JMP   AGN    START ANOTHER CONVERSION
ID=

-T

   SYMBOL TABLE 2000 203C
   AGN    0308   BACK  0313   FINISH 031D   INH    00F9
   PAD    1700   PBDD  1703   PBD    1702   PRGEND 032A
   SCANDS 1F1F   START 0300
```

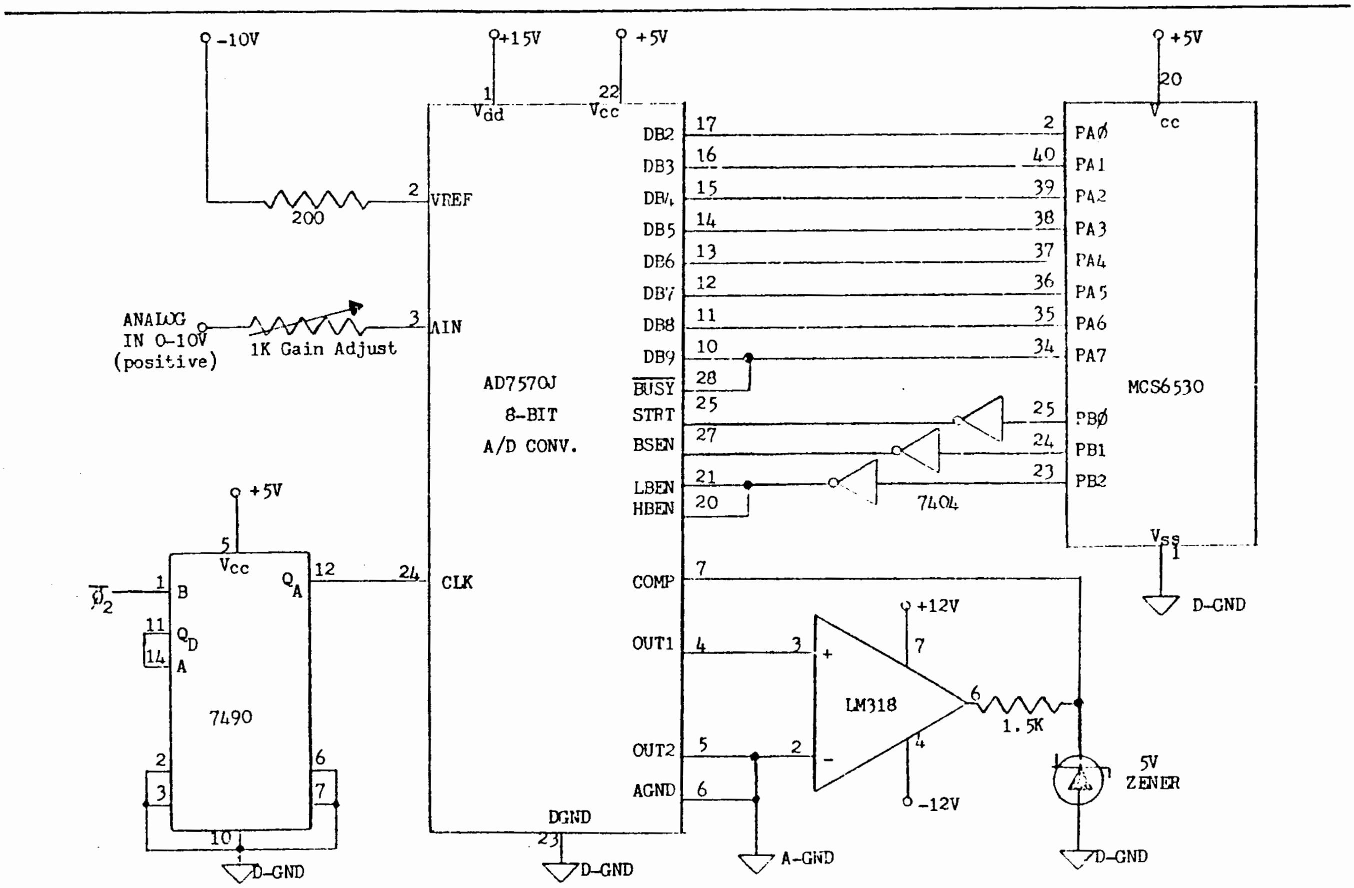



***Figure 1: Interface circuit. An LM311 voltage comparator is recommended instead of the LM318 op amp. D-GND is short for digital ground, and A-GND stands for analog ground. The 6530 is assumed to be part of a microprocesor.***

# SYMple Memory Expansion

John M. Blalock
3054 West Evans Drive
Phoenix, AZ 85023

**An 8K SYM from a board small enough to fit in the Synertek logo area of a standard enclosure? This interesting modification may violate good engineering practices, but it is difficult to argue with the designer's result.**

Synertek states in their SYM-1 manual, "it is believed that most users of the SYM-1 will ultimately use a TTY". I disagree. Most users, like me, will probably use some type of CRT terminal. The full power of the SYM monitor is not really appreciated until you connect it to a CRT or TTY. No wonder that Synertek made such a statement in the manual. The addition of a terminal turns the SYM into quite a little computer!

There is only one drawback to adding the terminal. Once you have it connected, you'll need to expand the SYM's memory to keep up with the larger programs, interpreters, and assemblers that you are sure to come up with!

### Tiny Basic

One of the easiest and least expensive additions that can be made to the SYM, after the addition of a TTY or CRT, is Tom Pittman's Tiny Basic. It is only $5.00 in paper tape format from him at Itty Bitty Computers, PO Box 23189, San Jose, CA 95153. Several ASK dealers sell it on cassette for $10.00. Get Version V.1K for the 6502 that starts at 0200 hex. It will fit from $0200 to $0AFF, leaving $0B00 to $0FFF available for programs. Since the SYM already includes a Break Test routine in its monitor, it is even simpler to interface Tiny Basic to the SYM than to the KIM. Make the following patches:

```
0206  4C  1B  8A     JMP INCHR
0209  4C  47  8A     JMP OUTCHR
020C  4C  3C  8B     JMP TSTAT
```

I also made the following optional changes to my copy:

| | | |
|---|---|---|
| 020F | 08 | Changes the character correction code to the ASCII backspace code. |
| 0210 | 40 | Changes the line cancel code to the "@" sign. |
| 0971 | 2A | Changes the prompt character from "colon" to "asterisk". |

### Memory Limitations

Tiny Basic is a very good interpreter, for its size, but only 1024 bytes are left out of the SYM's 4K RAM for Tiny Basic programs. I had an extra pair of 2114s on hand after I got Tiny up and running, and decided to see if there wasn't some way that I could make use of them.

I removed 2114s U12 and U13 from their sockets, mounted the extra two 2114s on top of them in the so called "piggyback" fashion, and soldered all pins of the extra 2114s to the same pins on the originals, except that the pin 8s were left unconnected.

I attached 30 GA wire to these pins on the two added 2114s, making sure that they were well insulated from the pin 8s of the original 2114s. The original ICs were then plugged back into the SYM and a memory test was run. So far, so good.

U1, a 74LS138, is a decoder that divides the first 8K of the SYM's memory into 1K blocks. The signals from it that correspond to the first four 1K blocks are used as the chip select signals for the original 2114s. The wires from pin 8 of the two added 2114s were wired to the fifth signal from U1, which is at pin 11 of its package.

Repeating the memory test, I had 5K of memory! I had just doubled the memory space available for Tiny Basic! Could it be expanded further? Perhaps, but not this way. The 2114s were too close together and got hotter than I would like to see them get.

### Bumble Bees Can't Fly

The address and data lines from the 6502 are only guaranteed to drive up to 1 TTL load and 130 pf of capacitance. No buffers exist on the SYM to reduce the loading. Adding up the capacitance of all the devices already on the SYM that are wired to the data and address buses, and adding a conservative figure for the capacitance of all the PC traces themselves, shows that the 6502 is being pushed to its limit already.

But those values of capacitance from the spec sheets are maximum values, while the 130 pf is a minimum. Let's try! The goal is to fit it in over the logo and Synertek name.

I built up a small perf board with IC sockets and wired them together using a wiring pencil and 36 GA solder strippable wire. Nine sockets were on the board, and an 18-pin homemade DIP plug plugged into the SYM's U19 socket to pick up most of the required connections.

Additional wires were run to the data lines at U12, and to the chip select signals from U1. It worked! I had an 8K SYM! And the board was small enough to fit in the area of the Synertek logo and name, between U1 and the original memory chips.

Several other SYM owners were very interested in my design, even though it violates good engineering practices. Enough interest was shown to commit the schematic of Figure 1 to an artwork and make up a few dozen copies of the board. This version is much neater than the prototype.

The board is double sided and has plated through holes. Two 16-pin DIP jumpers connect from it to the SYM's U12 and U19 sockets. (Ever try to buy an 18-pin jumper?) Four wires run from the board to pins of U1. U12, U19, and eight other 2114s mount on the final board.

None of the copies built to date have failed to work satisfactorily, nor does an oscilloscope show any degradation of the 6502's signals. My SYM has U20, U21, U22, U23, and U28 installed, so it is close to a worst case. I have had several dozen blank PC boards made which I will make available to other SYM owners for $5.00 each, with instructions. Please include a self addressed stamped envelope.

### Results

I will have to admit that the added board is an expansion to the SYM, but it certainly doesn't expand its size by much, does it? Tiny Basic now has 5K for its programs, a pretty respectable amount of memory. Synertek's BASIC, which is excellent, has 7679 bytes free, at initialization, instead of 3585. Many of the applications that I had only considered running on my KIM (29+ RAM!) system are now being run on the SYM, due to the faster tape interface, sufficient memory, BASIC in ROM, and the capabilities of the SYM's monitor.

It was certainly worth the trouble to try, even if bumble bees can't fly!

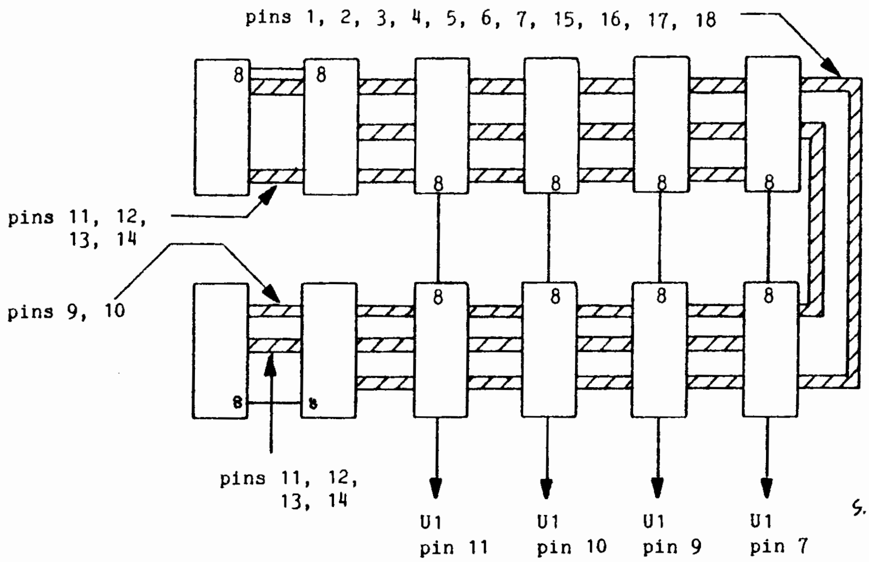



***Figure 1: W7AAY Sym-1 Memory Expansion***

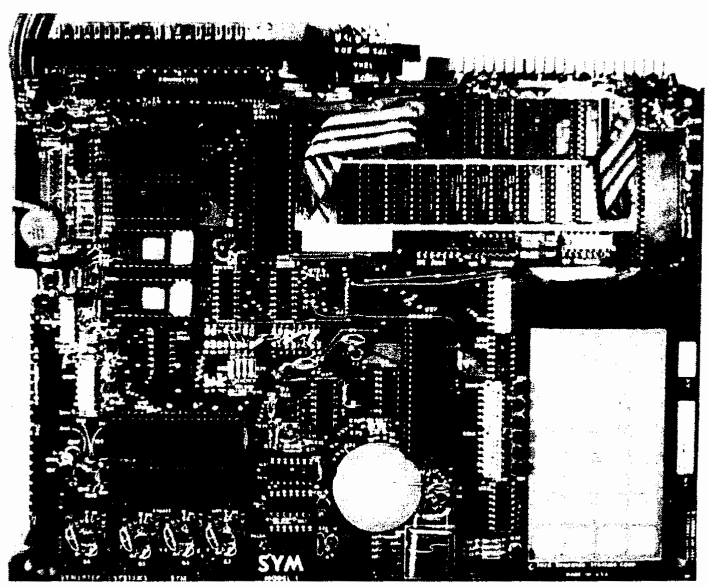


***Figure 2: The 8K SYM.***

# Define HI-RES Characters for the APPLE II

**This program makes it easy to generate and modify HI-RES characters on the APPLE II.**


**Robert F. Zant**
**Department of Accounting and Information Systems**
**North Texas State University**
**Denton, TX 76203**


The user contributed library of programs, Volumes 3, 4, and 5, recently released by the Apple Computer Company, contains a machine language routine for generating characters using the HI-RES features of the APPLE II. The package also includes a character table that contains 128 predefined characters.

The characters are represented in the table in a coded, reverse image format. The code is based on a 7 by 8 dot matrix representation for each character. The format for an "L" is depicted below. Note that a border is left at the top and side so that characters will be separated on the screen.

```
.....*.
.....*.
.....*.
.....*.
.....*.
*....*.
******.
.......
```

The coded table entry is derived from the format by substituting a zero for each dot and a one for each asterisk. Each line of the matrix is thereby coded into one byte. The high order bit is set to zero in each byte. Eight bytes are required to encode each character. The code for the "L" depicted above would be

02, 02, 02, 02, 02, 42, 7E, 00

The following program assists in defining characters and substituting them into the character table. Each character is defined in a regular dot matrix format, rather than in reverse image. The program automatically calculates the binary code for the equivalent rotated version. The letter "L" would be entered as:

```
.*.....
.*.....
.*.....
.*.....
.*.....
.*....*
.******
.......
```

Note that the dot matrix must remain intact, and must contain only dots and asterisks. The command to store the character, the CTRL S, must be entered after the matrix, on the ninth line. A carriage return is required after each command.

At the beginning of the run, the operator specifies the table position (0 to 127) for the first character to be defined. Thereafter, characters are automatically stored at succeeding locations in the table. Separate runs of the program can be used to define characters in non-contiguous table locations.

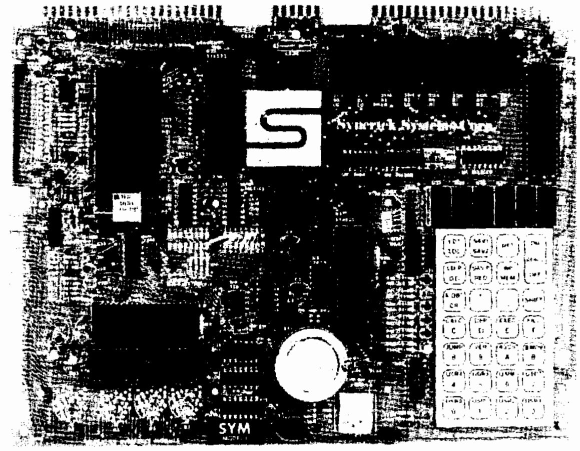
```
ASSEMBLE LIST

                    0100 ;MOVE TBL 1 TO TBL2
                    0110             .BA  $400
0400— A/  0B        0120 LOOP        LDY  #00
0402— B9  0B  04    0130             LDA  TBL1,Y
0405— 89  0B  05    0140             STA  TBL2,Y
0408— C8            0150             INY
0409  D0  F7        0160             BNE  LOOP
                    0170 ;
040B                0180 TBL1        .DS  256
050B                0190 TBL2        .DS  256
                    0200 ;
                    0210              .EN

LABEL FILE    1 = EXTERNAL

START = 0400        LOOP = 0402         TBL1 = 040B
TBL2 = 050B
110000.060B.060B
```

```
  50 REM
  60 REM  ASSUMES CHARACTER TABLE
  70 REM  BEGINS AT $6800
  80 REM
  90 REM
 100 TEXT : CALL -936
 200 VTAB 5: PRINT "ENTER DECIMAL EQUIVALENT"
 300 PRINT "OF FIRST 'ASCII' CHARACTER"
 350 PRINT "(MAXIMUM VALUE OF 127)"
 400 INPUT B
 425 IF B>=0 AND B<128 THEN 450: PRINT "RE-ENTER": GOTO 400
 450 B=26624+B*8
 500 CALL -936
 600 PRINT "CHANGE THE DOTS IN THE FOLLOWING MATRIX"
 700 PRINT "TO ASTERISKS TO DESCRIBE A FIGURE."
 750 PRINT "USE 'ESC C', 'ESC D', '->' AND '<-' TO  EDIT "
 775 PRINT "(LEAVE DOTS THAT ARE NOT REPLACED.)"
 800 PRINT "ENTER A 'CTRL S' TO STORE THE FIGURE."
 900 PRINT "ENTER A 'CTRL Q' TO QUIT."
1000 REM  PRINT MATRIX
1100 VTAB 9
1200 FOR I-0 TO 7
1300 PRINT "......."
1400 NEXT I
1500 VTAB 9
2000 REM  GET INPUT CHARACTER
2100 CALL -657
2200 IF PEEK (512)=147 THEN 3000
2300 IF PEEK (512)=145 THEN 9000
2500 GOTO 2000
3000 REM  ENCODE CHARACTER
3050 A=B: REM  SAVE BEGINNING OF CHARACTER
3100 REM  LOOK THRU MATRIX
3200 FOR I=1064 TO 1960 STEP 128
3250 C=0
3300 FOR J=0 TO 6
3400 IF PEEK (I+J)=174 THEN 3700
3500 IF PEEK (I+J)<>170 THEN 4000
3600 C=C+2 ^ J
3700 NEXT J
3800 POKE B,C:B=B+1
3900 NEXT I
3950 GOTO 1000
4000 REM  ERROR IN MATRIX
4100 VTAB 20
4200 PRINT "MATRIX CONTAINS INVALID CHARACTER"
4250 PRINT "RE-ENTER":B=A
4300 FOR I=0 TO 1000: NEXT I
4400 VTAB 20: CALL -958
4500 GOTO 1500
9000 END
```

## SYM-1, 6502-BASED MICROCOMPUTER

- FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to-use
- ALL LSI IC'S ARE IN SOCKETS
- 28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS
- EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY
- KIM-1* HARDWARE COMPATIBILITY

  The powerful 6502 8-Bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.
- THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user, expandable to five on-board.
- 4K BYTE ROM RESIDENT MONITOR and Operating Programs.
- Single 5 Volt power supply is all that is required.
- 1K BYTES OF 2114 STATIC RAM onboard with sockets provided for immediate expansion to 4K bytes onboard, with total memory expansion to 65, 536 bytes.
- USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs
- ENHANCED SOFTWARE with simplified user interface
- STANDARD INTERFACES INCLUDE:
  - —Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1* compatible, Hi-Speed 1500 Baud)
  - —Full duplex 20mA Teletype Interface
  - —System Expansion Bus Interface
  - —TV Controller Board Interface
  - —CRT Compatible Interface (RS-232)
- APPLICATION PORT: 15 Bi-directional TTL Lines for user applications with expansion capability for added lines
- EXPANSION PORT FOR ADD-ON MODULES (51 I/O Lines included in the basic system)
- SEPARATE POWER SUPPLY connector for easy disconnect of the d-c power
- AUDIBLE RESPONSE KEYPAD




Synertek has enhanced KIM-1* software as well as the hardware. The software has simplified the user interface. The basic SYM-1 system is programmed in machine language. Monitor status is easily accessible, and the monitor gives the keypad user the same full functional capability of the TTY user. The SYM-1 has everything the KIM-1* has to offer, plus so much more that we cannot begin to tell you here. So, if you want to know more, the SYM-1 User Manual is available, separately.

| | |
|---|---|
| **SYM-1 Complete w/manuals** | **$269.00** |
| **SYM-1 User Manual Only** | **7.00** |
| **SYM-1 Expansion Kit** | **75.00** |

Expansion includes 3K of 2114 RAM chips and 1-6522 I/O chip.

SYM-1 Manuals: The well organized documentation package is complete and easy-to-understand.

SYM-1 CAN GROW AS YOU GROW. Its the system to BUILD-ON. Expansion features that are soon to be offered:

| | |
|---|---|
| ***BAS-1 8K Basic ROM (Microsoft)** | **$159.00** |
| ***KTM-2 TV Interface Board** | **349.00** |

**We do honor Synertek discount coupons*

## QUALITY EXPANSION BOARDS DESIGNED SPECIFICALLY FOR KIM-1, SYM-1 & AIM 65

These boards are set up for use with a regulated power supply such as the one below, but, provisions have been made so that you can add onboard regulators for use with an unregulated power supply. But, because of unreliability, we do not recommend the use of onboard regulators. All I.C.'s are socketed for ease of maintenance. *All boards carry full 90-day warranty.*

All products that we manufacture are designed to meet or exceed industrial standards. All components are first qualtiy and meet full manufacturer's specifications. All this and an extended burn-in is done to reduce the normal percentage of field failures by up to 75%. To you, this means the chance of inconvenience and lost time due to a failure is very rare; but, if it should happen, we guarantee a turn-around time of less than forty-eight hours for repair.

*Our money back guarantee:* If, for any reason you wish to return any board that you have purchased directly from us within ten (10) days after receipt, complete, in original condition, and in original shipping carton; we will give you a complete credit or refund less a $10.00 restocking charge per board.

### VAK-1 8-SLOT MOTHERBOARD

This motherboard uses the KIM-4* bus structure. It provides eight (8) expansion board sockets with rigid card cage. Separate jacks for audio cassette, TTY and power supply are provided. Fully buffered bus.

**VAK-1 Motherboard** **$129.00**

### VAK-2/4 16K STATIC RAM BOARD

This board using 2114 RAMs is configured in two (2) separately addressable 8K blocks with individual write-protect switches.

| | |
|---|---|
| **VAK-2 16K RAM Board with only 8K of RAM (½ populated)** | **$239.00** |
| **VAK-3 Complete set of chips to expand above board to 16K** | **$175.00** |
| **VAK-4 Fully populated 16K RAM** | **$379.00** |

### VAK-5 2708 EPROM PROGRAMMER

This board requires a +5 VDC and ±12 VDC, but has a DC to DC multiplyer so there is no need for an additional power supply. All software is resident in on-board ROM, and has a zero-insertion socket.

**VAK-5 2708 EPROM Programmer** **$269.00**

### VAK-6 EPROM BOARD

This board will hold 8K of 2708 or 2758, or 16K of 2716 or 2516 EPROMs. *EPROMs not included.*

**VAK-6 EPROM Board** **$129.00**

### VAK-7 COMPLETE FLOPPY-DISK SYSTEM (May '79)

### VAK-8 PROTYPING BOARD

This board allows you to create your own interfaces to plug into the motherboard. Etched circuitry is provided for regulators, address and data bus drivers; with a large area for either wire-wrapped or soldered IC circuitry.

**VAK-8 Protyping Board** **$49.00**

## POWER SUPPLIES

ALL POWER SUPPLIES are totally enclosed with grounded enclosures for safety, AC power cord, and carry a full 2-year warranty.

### FULL SYSTEM POWER SUPPLY

This power supply will handle a microcomputer and up to 65K of our VAK-4 RAM. ADDITIONAL FEATURES ARE: Over voltage Protection on 5 volts, fused, AC on/off switch. Equivalent to units selling for $225.00 or more.

**Provides +5 VDC @ 10 Amps & ±12 VDC @ 1 Amp**

**VAK-EPS Power Supply** **$125.00**

**KIM-1* Custom P.S. provides 5 VDC @ 1.2 Amps and +12 VDC @ .1 Amps**

**KCP-1 Power Supply** **$41.50**

**SYM-1 Custom P.S. provides 5 VDC @ 1.4 Amps**

**VCP-1 Power Supply** **$41.50**

*KIM is a product of MOS Technology

RNB ENTERPRISES INCORPORATED

2967 W. Fairmount Avenue
Phoenix AZ 85017
(602)265-7564

master charge VISA

# Common Variables on the APPLE II

**Modular software designs rely on common variables to pass data between interrelated programs. Two short subroutines emulate the DOS CHAIN capability by allowing use of common variables under Integer or Applesoft BASIC, without a disk.**

**Professor Robert F. Zant**
**Department of Accounting and Information Systems**
**North Texas State University**
**Denton, TX 76203**

The solution of complex problems often leads to the writing of several interrelated programs. Furthermore, the programs usually use several of the same variables — called common variables. This is accomplished in most systems by not destroying the common variables when a new program is loaded. Thus, the value of a variable can be defined in one program and used in subsequent programs.

There is no true facility with the APPLE II for using common variables. The CHAIN command in DOS comes close to providing the capability, but it saves all variables instead of just saving designated common variables. Also, it can only be used with Integer BASIC programs run under DOS. No facility for common variables is provided for non-disk systems or for AppleSoft programs.

The attached machine language routines can be used to pass all variables to succeeding programs. Integer BASIC and AppleSoft versions are provided. Both versions are used as follows:

1. Load the machine language routine before the first BASIC program is executed.
2. In each BASIC program except the last program, "CALL 774" immediately before termination or before the DOS command to RUN the next program.
3. In each BASIC program except the first program, "CALL 770" before executing any statement that affects or uses variables. Do not reDIMension variables in subsequent programs.

Since all variables are saved whether they are needed or not, main storage is used most efficiently if the same set of variable names is used in all programs. This, of course, is required for the variables that are intended to be common for all programs. Other main storage is reclaimed by the reuse of the names of "non-common" variables.

String variables will not always be saved correctly in AppleSoft. If the string value was read from disk, tape or keyboard, the value will be saved. If the string value is defined in an assignment statement (e.g. A$ = "XXX"), the value will not be available to subsequent programs.

The routine for Integer BASIC is very simple. The variable table pointer is simply saved and restored. The AppleSoft version, however, is a little more complex. The AppleSoft version of the routine moves all non-string variables to high RAM, just under the strings. Then, when called at the beginning of the next program, via "CALL 770", the routine moves the variables back down to the end of the new program.

```
                    * ROUTINE TO SAVE AND RECALL
                    * COMMON VARIABLES FOR APPLESOFT II BASIC
                    * PROGRAMS ON THE APPLE II
                    *
                    * WRITTEN 03/16/79 BY ROBERT F. ZANT
                    *
03A7                DL     *      $0018
03A7                DH     *      $0019
03A7                CL     *      $001A
03A7                CH     *      $001B
03A7                EL     *      $001C
03A7                EH     *      $001D
03A7                A1L    *      $003C
03A7                A1H    *      $003D
03A7                A2L    *      $003E
03A7                A2H    *      $003F
03A7                A4L    *      $0042
03A7                A4H    *      $0043
0302                       ORG    $0302
0302 4C 56 03              JMP    RECALL ***ENTRY 770
0305 00                    BRK
0306 38                    SEC           ***ENTRY 774 - SAVE NUMERICS
0307 A5 6F                 LDA    $006F  COMPUTE ADDRESSES FOR MOVE
0309 85 18                 STA    DL     SAVE START OF STRING ADDRESS
030B E5 6D                 SBC    $006D  END OF NUMERICS
030D 85 1A                 STA    CL     TEMPORARY STORAGE
030F A5 70                 LDA    $0070
0311 85 19                 STA    DH
0313 E5 6E                 SBC    $006E
0315 85 1B                 STA    CH     TEMPORARY STORAGE
0317 18                    CLC
0318 A5 1A                 LDA    CL
031A 65 69                 ADC    $0069  START OF NUMERICS
031C 85 1A                 STA    CL     TEMP STORAGE
031E A5 1B                 LDA    CH
0320 65 6A                 ADC    $006A
0322 85 1B                 STA    CH
0324 A6 1A                 LDX    CL     SUBTRACT ONE
0326 D0 02                 BNE    A1
0328 C6 1B                 DEC    CH     START OF COMMON
032A CA             A1     DEX
032B 86 1A                 STX    CL
032D 86 42                 STX    A4L    SET UP MOVE
032F A5 1B                 LDA    CH
0331 85 43                 STA    A4H
0333 A5 69                 LDA    $0069  START OF VARIABLES
0335 85 3C                 STA    A1L
0337 A5 6A                 LDA    $006A
0339 85 3D                 STA    A1H
033B A5 6D                 LDA    $006D  END OF VARIABLES
```

```
0343 A0 00            LDYIM $00
0345 20 2C FE         JSR   $FE2C   USE MONITOR MOVE ROUTINE
0348 38               SEC           COMPUTE DISPLACEMENT
0349 A5 6B            LDA   $006B   TO ARRAYS
034B E5 69            SBC   $0069
034D 85 1C            STA   EL
034F A5 6C            LDA   $006C
0351 E5 6A            SBC   $006A
0353 85 1D            STA   EH
0355 60               RTS           BACK TO BASIC
                *
0356 A5 1A      RECALL LDA  CL      ***ENTRY 770 - RECALL
0358 85 3C            STA   A1L     SET UP MOVE
035A A5 1B            LDA   CH
035C 85 3D            STA   A1H
035E A5 18            LDA   DL
0360 85 6F            STA   $006F   START OF STRINGS
0362 85 3E            STA   A2L
0364 A5 19            LDA   DH
0366 85 70            STA   $0070
0368 85 3F            STA   A2H
036A A5 69            LDA   $0069   START OF NUMERICS
036C 85 42            STA   A4L
036E A5 6A            LDA   $006A
0370 85 43            STA   A4H
0372 A0 00            LDYIM $00
0374 20 2C FE         JSR   $FE2C   USE MONITOR MOVE ROUTINE
0377 18               CLC           COMPUTE START
0378 A5 69            LDA   $0069   OF ARRAYS
037A 65 1C            ADC   EL
037C 85 6B            STA   $006B
037E A5 6A            LDA   $006A
0380 65 1D            ADC   EH
0382 85 6C            STA   $006C
0384 38               SEC           COMPUTE END OF NUMERICS
0385 A5 6F            LDA   $006F
0387 E5 1A            SBC   CL
0389 85 6D            STA   $006D
038B A5 70            LDA   $0070
038D E5 1B            SBC   CH
038F 85 6E            STA   $006E   TEMP STORAGE
0391 18               CLC
0392 A5 6D            LDA   $006D
0394 65 69            ADC   $0069
0396 85 6D            STA   $006D   TEMP VALUE
0398 A5 6E            LDA   $006E
039A 65 6A            ADC   $006A
039C 85 6E            STA   $006E   TEMP VALUE
039E A5 6D            LDA   $006D   SUBTRACT ONE
03A0 D0 02            BNE   A2
03A2 C6 6E            DEC   $006E   END OF NUMERICS
03A4 C6 6D      A2    DEC   $006D
03A6 60               RTS           BACK TO BASIC
```

T

```
SYMBOL TABLE 2000 205A
AQ   032A   AQH  003D   AQL    003C   AR   03A4
ARH  003F   ARL  003E   ATH    0043   ATL  0042
CH   001B   CL   001A   DH     0019   DL   0018
EH   001D   EL   001C   RECALL 0356
```

```
                        * ROUTINE TO SAVE AND RECALL
                        * COMMON VARIABLES FOR INTEGER BASIC
                        * PROGRAMS ON THE APPLE II
                        *
                        * WRITTEN 03/16/79 BY ROBERT F. ZANT
                        * MODIFIED 7/4/79 BY MICRO STAFF
                        *
0318                    CL     *     $001A
0318                    CH     *     $001B
0302                           ORG   $0302
0302 4C 0F 03                  JMP   RECALL ***ENTRY 770
0305 00                        BRK
0306 A5 CC                     LDA   $00CC  ***ENTRY 774 - SAVE VARIABLES
0308 85 1A                     STA   CL     SAVE END OF
030A A5 CD                     LDA   $00CD  VARIABLE TABLE
030C 85 1B                     STA   CH
030E 60                        RTS          BACK TO BASIC

030F A5 1A              RECALL LDA   CL     ENTRY 770 - RECALL VARIABLES
0311 85 CC                     STA   $00CC  RESET END OF
0313 A5 1B                     LDA   CH     VARIABLE TABLE
0315 85 CD                     STA   $00CD
0317 60                        RTS          BACK TO BASIC
```

# Classified Ads



## BAD REVIEW

What's worse than getting a complaint about MICRO that is not valid? Getting one that is! I received a telephone call from Dr. Rodney Zaks the other day concerning a review which was published about his book Programming the 6502 in an earlier issue of MICRO. His complaint was not that the review was unfavorable to his book, but that the "review" went beyond the boundaries of a review and made a number of unwarranted accusations about the techniques, motivations and values of the entire product line offered by SYBEX, the publisher of Dr. Zaks' book. I told Dr. Zaks that I didn't really remember the review, that it was against MICRO's basic policy to print anything of that nature, but that I would look into the matter and if he was correct, I would print an apology and try to rectify the matter as much as possible.

Well, when I read the "review" I was surprised. I agree with Dr. Zaks. While the first part of the review is critical of the book, it is within the rights of a reviewer. The second part of the "review" should not have been printed. It does not provide any useful information to the reader and its negative assertions are unjustified. Since I was both Editor and Publisher at the time the review was printed, I take full blame for its appearance in MICRO, and apologize to Dr. Zaks and SYBEX for its appearance.

Since it is against MICRO's policy to print such material, how did it get printed? All I can figure is that it "fell through the crack". With the very small staff we had at the time, most of our efforts were spent on getting the major articles into shape for publication: technical editing, typesetting, proofing, pasting-up, and so forth. Very little time was left for a careful analysis or review of the small "filler" material, and the "review" never got the attention it should have, and so "slipped in". I suggest that all readers ignore the negative implications of the second half of the review. With the enlargement of the MICRO staff to include a full time editor as well as other support personnel, we have more time and similar problems should not occur.

MICRO has printed very few reviews to date: three book reviews and only a couple hardware or software reviews. The reason for this is that we feel that unsolicited reviews tend to be biased. The author is writing because he either loves or hates a product. We are working on a plan by which MICRO can establish a panel of reviewers and actively start doing product reviews which are both fair and thorough. Information about this plan will appear in MICRO shortly.

# Classified Ads



## MANUSCRIPT COVER SHEET

MICRO™

Please enter all of the information requested on this cover sheet:

**Date Submitted:** ______________________

**Author(s) Name(s)** ______________________ **Telephone:** ____________
(To be published exactly as entered) (This will NOT be published)

**Mailing Address:**
(This will be published) ______________________

______________________

**AUTHOR'S DECLARATION OF OWNERSHIP OF MANUSCRIPT RIGHTS:** This manuscript is my/our original work and is not currently owned or being considered for publication by another publisher and has not been previously published in whole or in part in any other publication. I/we have written permission from the legal owner(s) to use any illustrations, photographs, or other source material appearing in this manuscript which is not my/our property. If required, the manuscript has been cleared for publication by my/our employer(s). Note any exceptions to the above (such as material has been published in a club newsletter but you still retain ownership) here:

**Signature(s):** ______________________ **Date:** __________

______________________

Any material for which you are paid by Micro Ink, Incorporated, whether or not it is published in MICRO, becomes the exclusive property of Micro Ink, Incorporated, with all rights reserved.

### A FEW SUGGESTIONS

All text should be typewritten using double or triple spacing and generous left and right margins. Figures and illustrations should be drawn neatly, either full size or to scale, exactly as they will appear in MICRO. Photographs should be high contrast glossy prints, preferably with negatives, and program listings should be machine generated hard copy output in black ink on white paper. Assembly language program listings need not be of especially high quality, since these are normally re-generated in the MICRO Systems Lab, but they must include object code as a check against typographical errors.

Since other MICRO readers will be copying your program code, please try to test your program thoroughly and ensure that is is as free from errors as possible. MICRO will pay for program listings, figures and illustrations, in addition to the text of an article; however, MICRO does not normally pay for figures that must be re-drawn or for programs that must be re-keyboarded in order to obtain a high contrast listing. Any program should include a reasonable amount of commentary, of course, and the comments should be part of the source code rather than explanations added to the listing.

Send your manuscripts to:

**MICRO, P.O. Box 6502, Chelmsford, MA 01824, U.S.A.**

# 6502 Bibliography: Part XII

Dr. William R. Dial
438 Roslyn Avenue
Akron, OH 44320

**449. Road and Track Magazine (May, 1977)**
Dinkel, John "Computerized Road Testing", pg. 60.
Using a KIM-1 based system to gather road test data.

**450. Personal Computing 3 No 3 (Mar., 1979)**
Anon, "Tom Pittman, Tiny Basic and Cosmac", pg. 20-22.
Comments on merits of various microprocessors from the viewpoint of writing higher level languages.

Zimmerman, Mark "Line Renumbering on the PET", pg. 24-29.
Both Basic and Assembly Language versions of a renumbering program.

**451. SES Newsletter Issue 7 (Mar., 1979)**
Romano, Nicholas A. "Billboard", pg. 2.
A horizontal scrolling message ala Goodyear Blimp for Apple.

Romano, Nicholas A. "Circle Graphics" pg. 2-3.
Several interesting circle programs including SEWER PIPE. For Apple.

McClelland, Geo. "Machine and Assembly language Programming", pg. 4-6.
A tutorial leading you by the hand thru the mysterious machine language of Apple.

McClelland, Geo. "The & Command", pg. 6.
How to use & in the Apple.

**452. Cider Press 1 No 11 (Feb., 1979)**
Nareff, Max J. "R&R for Decimal Dumps", pg. 5.
How to round off those long decimal strings on the Apple.

Kamins, Scot "FP Disk Trace", pg. 5.
Tracing an Applesoft program with disk booted.

Hoag, C.G. "Page Flip", pg. 6.
Procedure to move Page 1 to page 2 from Basic on the Apple.

Hertzfeld, Andy "Free Sector Program", pg. 8.
Calculates the amount of free space available on an Apple II diskette.

Nareff, M.J. "Space Commanders-SPC(X) and TAB(X)", pg. 10.
Useful commands for formatting tables, etc. for the Apple.

Kamins, Scot "Son of N", pg.
A novel HELLO program for the Apple Disk.

Anon, "February Disk of the Month", pg. 11.
List of programs.

**453. MICRO No 10 (Mar., 1979)**
DeJong, Marvin L. "A Simple 24 Hour Clock for the AIM 65", pg. 5.
Displays time in hours: min: sec on the AIM display.

Hill, Alan g. "Apple II-Trace List Utility", pg. 9-14.
The utility presented here will list each Basic program source statement line by line in the order executed.

Rowe, Mike "The MICRO Software Catalog: VI", pg. 15-16.
Review of 9 6502 software packages.

Kosinski, John T. and Suitor, Richard F. "6522 Chip Setup Time", pg. 17.
One more article on the 6522 I/O problems that should put this controversy to rest.

Sherburne, John R. "High-Resolution Plotting for the PET", pg. 19-23.
Some very interesting graphics programs for the PET.

Zuber, Jim "Using Tiny Basic to Debug Machine Language Programs", pg. 25-30.
Debugging on the KIM-1 and other 6502 machines.

Tripp, Robert M., Ph.D. "Ask the Doctor: An ASK EPROM Programmer", pg. 31-35.
This EPROM program will run on AIM, SYM or KIM systems.

Herman, Harvey B. " 'Thanks for the Memories' A PET Machine Language Memory Test", pg. 37-40.
A very efficient memory test for the PET.

Jones, Robert E. "The OSI Flasher: Basic-Machine Language Interfacing", pg. 41-42.
Tutorial 6502 program.

Dial, Wm. R. "6502 Bibliography—Part IX", pg. 47-48.
the literature on this most popular of microporcessors continues to grow.

**454. Cider Press 1 No 12 (Mar., 1979)**
Anon, "March Disk of the Month", pg. 2.
21 New programs, for Apple.

Uhley, John "HIRES" Using its Commands and Saving 'Still Life' Pictures", pg. 4-5.
Hires Tutorial article, for Apple.

Nareff, Max and Kamins, Scot "V.X. Defeat", pg. 6.
Two methods of avoiding the volume mismatch message on the Apple disk.

Anon, "MAT Functions with the Apple—Part I", pg. 6.
Simple Matrix operations can be accomplished on the Apple.

Garrigues, Chris "Simple Animation in One Easy Lesson", pg. 7.
Demonstration of the use of PageI/Page 2.

Rahl, Robert R. "Great Grand Nephew of N", pg. 7.
Yet another modification in the development of this Hello program for the Apple disk.

**455. Interface Age 5 Iss3 (Mar., 1979)**
Margolin, Jed "A Musical Synthesizer for the KIM-1", pg. 65-67.
Plays two tunes or you can key in your own tune.

**456. Kilobaud No 28 (April, 1979)**
Lindsay, Len "PET-Pourri", pg. 8-10.
New Accessories for PET, Software Review, How to protect programs, etc.

Grina, Jim "RePROM", pg. 19.
A completely revised version of a PROM program for the KIM.

Luffman, Frederick E. "Bar-Graph Generator", pg. 90-92.
A useful program for PET owners who want graphing information without learning statistics.

Schwartz, Marc "Starship Attack", pg. 106-107.
A game for the Apple.

**457. Byte 4 No 3 (March, 1979)**
Meushaw, Robert V. "The Standard Data Encryption Algorithm—Part 1: An Overview", pg. 66-74.
KIM is used to demonstrate the advantages and disadvantages of the 6502 in handling the data algorithm.

**458. Creative Computing 5 No 3 (Mar., 1979)**
Palenik, Les "PET Machine—Language Programming", pg. 49.
Here is a low level monitor for expanding the PET's programming capabilities.

Anon, "Personal Electronic Transactions", pg. 33-37.
Notes on disk for the PET, machine language commands PEEK, POKE, SYS, USR, etc. and Music programs.

Owens, Dr. James "Teachers! A Social Science Survey Program", pg. 68-72.
An OSI computer program (6502) for analysis of survey questionaires.

**459. The Paper 1 Iss 10 (Dec., 1978)**
Sparks, Paul W. "Tape Head Alignment on the PET", pg. 4.
Simple instructions for a critical and important adjustment.

Anon, "More on Alien Basics", pg. 9.
Hints to help translate programs into PET Basic.

Julich, Paul M. "Delete", pg. 12.
Code to be used to delete a group of statements from any program on the PET.

Bunker, W. Marvin "The Great Circle Route", pg. 20.
Determining distance between two points on the Earths surface.

Strasma, James "Saving Time and Space", pg. 23-26.
How to condense your PET programs into less space.

Anon, "Automatic Repeating Keys", pg. 27.
How to make any key on the PET auto-repeating.

**460. EDN 23 No 15 (Aug. 20, 1978)**
Conway, John "Serial I/O Thrusts INDECOMP into Asynchronous communications", pg. 89-97.
The successful conclusion of project INDECOMP using the Apple II and a PIA interface.

**461. EDN 23 No 17 (Sept., 1978)**
Patstone, Walt "Apple II—No PIA Problem", pg. 17.
The Editor of EDN Magazine reports that the controversy about the questionable compatibility of the APPLE II with the PIA was all a mistake, and efforts to duplicate the "problem" have met failure.

**462. Creative Computing 5 NO 3 (Mar., 1979)**
Swenson, Carl "Disk Power: How to Use It—Apple's New Disk System", pg. 124-127.
Helpful hints for Apple Disk users.

**463. The Paper 1 Iss 9 (Nov., 1978)**
Connely, R. Dale "Fix for the Disappearing Cursor", pg. 3.
A simple diode fix for this common problem in the PET.

Morehead, James C. "Cursor Problems", pg. 3-4.
The cursor problem in the PET was alleviated by a fan.

Baltay, Michael "Limitation in the Dimension Statement", pg. 8.
This program for the PET checks the limits in the DIM statement.

Anon, "ROM Test", pg. 17-18.
A program to test the ROM on your PET.

Busdiecker, Roy "The PET Symbol and Data Formats", pg. 19-22.
Explore your RAM to get interesting information on the PET's management of variables.

**464. Personal Computing 3 No 4 (Apr., 1979)**
Anon, "Apple Slices the Grovery Bills", pg. 9-10.
A description of one practical use of the APPLE II.

Vizzone, Raymon T. "Artist Extraordinaire", pg. 58-60.
Create pop-art images on your color TV.

**465. Call—Apple 2 No 3 (Mar., 1979)**
Golding, Val J. "Applesoft from Bottom to Top", pg. 3-10.
Includes several useful utility programs: Print current value of all Applesoft pointers; Program to store Applesoft programs at 3072 so 2nd text screen may be used; program to display Applesoft tokens; appending Applesoft programs; examining variables in memory.

Golding, Val J. "A Note or Three About BINADR 3.2", pg. 11.
A BINADR program for the forthcoming DOS Version 3.2

Cross, Mark "Hi-Res Colors", pg. 12.
Program to Clear the GR screen to any color.

Aldrich, Ron "Illegal Control Characters in REM lines',, pg. 13-17.
Program to insert illegal control characters into programs.

Aldrich, Darrell "The Mystery of Text Files", pg. 15.
A short tutorial article.

Paymar, Dan "Disk Access Utility", pg. 16-17.
This program dumps a whole disk or a track at a time to the printer or screen.

Sedgewick, Dick "Applelock", pg. 21.
An integer Basic program to add to the end of a program to "write protect" or "lock" the entire program.

Paymar, Dan "Keyboard Modifications to Get "[", "/" and "—"", pg. 23.
At the risk of voiding the warranty, changes can be made in the Apple circuit board to provide the extra characters.

Aldrich, Darrell "The Apple Doctor", pg. 25.
A number of useful tips including how to save variables in Integer basic to disk and how to uncover control characters used in catalog titles to lock programs.

**466. The Paper 1 Iss 8 (Oct., 1978)**
Anon, "Intro to Basic", pg. 12-14.
Discusses MAT READ and MAT PRINT and other alien commands giving the appropriate PET translation.

Maier, Gary A. "Resequence", pg. 19-20.
A renumbering program for the PET.

Butterfield, Jim "Some PET Routines", pg. 23-24.
Index to some useful routines.

**467. Recreational Computing 7 No 5 Iss 38 (Mar./Apr., 1979)**
Carpenter, Chuck "Easy Pokeing with Applesoft Basic", pg. 46-47.
An easy way to enter in machine language problems.

Saal, Harry "SPOT", pg. 52-54.
New models of the PET are said to be on the way, with improved keyboard, external cassette, etc. More on music programs for the PET.

Day, Jim "Apple-Rose", pg. 55.
Program for the Apple plots rose-leaf patterns. A lot of fun can be had by changing parameters in this program.

**468. Rainbow 1 Iss 3, (Mar., 1979)**
Watson, Allen III "Don't Ignore Integer Basic", pg. 2.
Some real advantages of Integer Basic are discussed.

Anon, "Apple II Memory Map, Showing Areas Over-written

when Booting DOS", pg. 6.
Helpful in diagnosing 'What Happened?'

Wozniak, Steve "Auto Repeat for Apple II Monitor Commands", pg. 7.
How to automatically repeat Monitor Commands.

Watson, Allen III "Add a Color-Killer for Clearer Text Display", pg. 9-12.
This simple modification is now being incorporated in production Apples.

Dubas, Andy "Hires Graphics Plotting Program", pg. 14-16.
A plotting program that will solve and plot almost any polynamial equation. For an Apple (48K) with AS ROM and DOS.

Watson, Allen III "Integer Basic Square Root", pg. 18.
Simple program for this omission from Integer Basic.

**469. The Paper 1 Iss 7 (Sept., 1978)**
Sokel, Ralph J. "Looking at Basic ROM", pg. 6.
How to examine the Commodore PET Basic ROM.

Garst, John F. "Renumber", pg. 7.
A modified program.

Anon, "Intro to Basic: Strings", pg. 8-12.
Tutorial article with examples on PET basic.

Smith, Ron "Cassette I/O", pg. 20-21.
All about PET Cassette format.

**470. The Paper 1 Iss 6 (Aug., 1978)**
Alexander, Frank "Demo for ARCOS(X(", pg. 6.
A short demo for PET.

Schwartz, Glenn "Tone on the PET", pg. 11.
A short program and hardware for producing tones.

Martin, Russell "Interfacing an Audio Cassette Deck to the Cassette I/O Port", pg. 16.
How to hook-up two cassette recorders to the PET.

McCarthy, Charles A. "PET Basic Documentation", pg. 18-21.
Discussion of floating point numbers in PET Basic.

**471. Dr. Dobbs Journal 3 Iss 6 No 26 (June/July, 1978)**
Herzfeld, Andy "Lazarus", pg. 31-33.
A program to resurrect BASIC programs on the Apple II.

**477. Softalk 1 Issue 1 (Apr., 1979)**
Smith, Wm. V.R. and Depew, Wm. H. "Transferring Appletalker to Disk", pg. 1.
Detailed instructions from Softape on modifying their tapes for disk. SOFTALK is a newsletter published by Softape, 12 issues $5.00.

Anon, "Talksaver, a Disk Save for Appletalker", pg. 2-3.
Procedure and software listing to allow Disk II owners to save the data tables created by Appletalker to a named disk file.

Anon, "Append Procedure for Prefix Programs", pg. 4.
Detailed procedure for appending prefix programs to tape or disk programs.

Anon, "How to Save Any Program in the Apple's Memory", pg. 6.
Many programs contain subroutines which interface with saving the progrm to tape. Here is a way to overcome this.

Anon, "Apple Memory Map", pg. 5.
Map showing just how Apple Talker and Apple-lis'ner are situated in memory.

**473. Dr. Dobbs Journal 4 Issue 4 No 34 (Apr., 1979)**
Prigot, Jonathon M. "OSI Basic for the KIM-1", pg. 37-39.
How to adapt the OSI Basic to KIM.

Lentezner, Mark "Improve Your OSI Resident Editor", pg. 46.
A simple program to fix a problem with OSI's resident.

**475. Creative Computing 5 No 4 (Apr., 1979)**
Milewski, Richard A. "Apple-Cart", pg. 22-23.
All about the EXEC command of the Apple DOS. With examples.

Yob, Gregory "Personal Electronic Transactions", pg. 28-32.
Discussion of the PET Clock with example, PET files, etc.

Zorn, Michael d. "Superose", pg. 98-99.
A rose program for the PET.

**475. MICRO No 11 (Apr., 1979)**
Hill, Alan G. "An Apple II Program Edit Aid", pg. 5-7.
A basic program to locate all occurences of a variable name, character string or Basic statements.

Stelly, J. "Lifesaver", pg. 9-11.
This program makes it easy to save LIFE patterns to cassette, run Life at different rates, etc.

Vrtis, Nicholas J. "Corrected KIM Format Loader for SYM-1", pg. 12-14.
Program helps overcome the SYM-1's KIM tape "2F" problem with a corrected loader.

Hoyt, Bruce "A Close Look at the Superboard II", pg. 15-18.
In addition to an overview report on the Superboard II, there is presented a cassette save/hex memory dump program and a very useful table of memory usage.

Sensicle, Andrew V.W. "SKIM or MAXI-KIM", pg. 19-20.
An extended monitor supports a PC decrement function, as well as "open up" and "close up" modes to move blocks of data to make room for adding code and a branch calculator to help determine the relative branch addresses.

Stein, Robert A., Jr. "A Cassette Operating System for the Apple II", pg. 21-23.
Program makes it possible to load programs into the Apple by typing the name of the program and the cassette operating program goes looking for it and if it is found it is loaded into the Apple.

Tripp, Robert M., PhD "Ask the Doctor—Part III—Bits and Bytes", pg. 25-26.
Problems and fixes discussed this month include a corrected AIM SYNC program, a patch for the AIM-Disassembler, Sym Tape evaluation, and comments on Synertek Basic (8K) V1.1.

Rowe, Mike (Micro staff) "The Micro Software Catalog: VII", pg. 29-30.
Ten more entries.

Gieryic, John "SYM-1 6522-Based Timer", pg. 31-32.
A tutorial article on the timer and the working of the 6522 versatile interface adapter.

Chalfin, Edward "The TVT-6; A User's Report", pg. 34.
A user's impression of this inexpensive method of getting a video signal out of the KIM-1.

Dial, Wm. R. "6502 Bibliography—Part X", pg. 36.
Forty more references to the 6502 literature.

Rindsberg, Don "The Ultimate PET Renumber", pg. 37-47.
A major program for the PET.

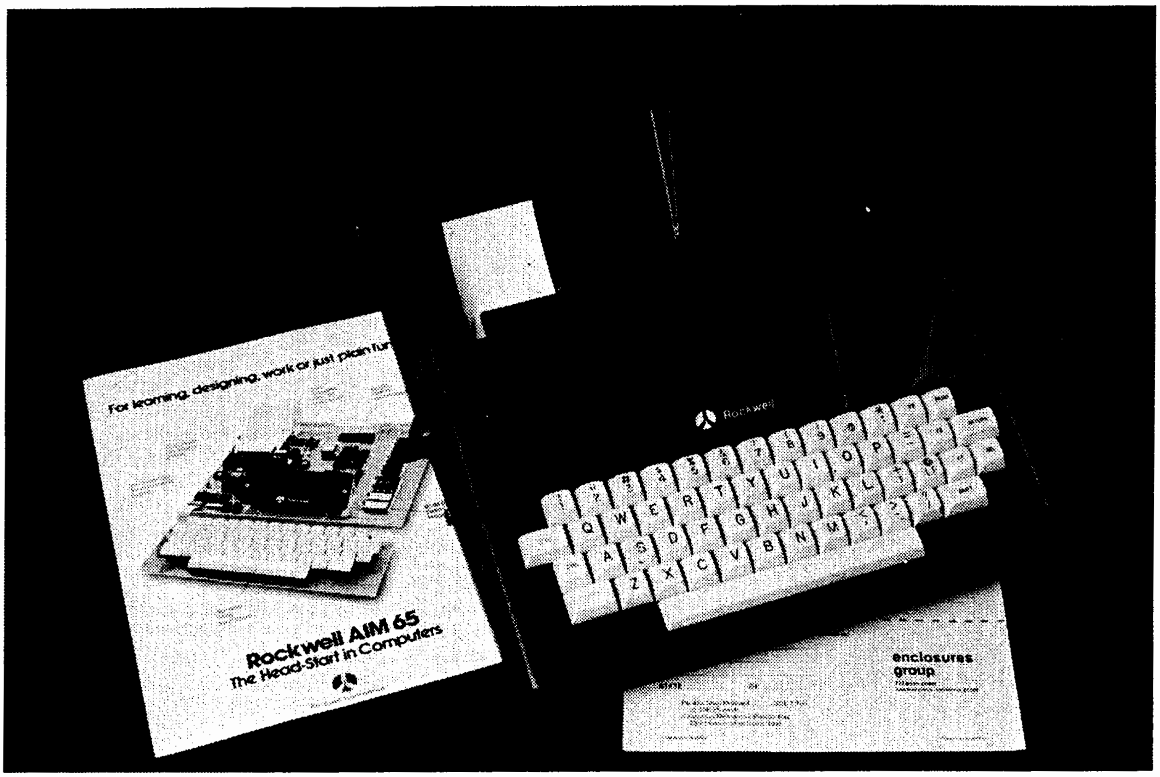
**476. The Paper 1 Iss 5 (July, 1978)**
Oakes, Peter L.A. "Routines for Finding Arcsin and Arccos", pg. 4.
Improved routines for the PET.

Laudereau, Terry "PET Files", pg. 5.
Discussion of PET files and the commands, OPEN, CLOSE, INPUT#, GET#, etc.

VanDusseldorp, Dean "Pause Routins", pg. 7.
PET program to provide pauses in a program.

Anon, "Simple Memory Test for PET", pg. 9.
Program runs until a bad Ram is found.

# PERFECT AIM




**ATTRACTIVE FUNCTIONAL PACKAGING FOR YOUR AIM-65 MICROCOMPUTER**
- Professional Appearance
- Striking Grey and Black Color Combination
- Protects Vital Components

**ENGINEERED SPECIFICALLY FOR THE ROCKWELL AIM-65**
- All Switches Accessible
- Integral Reset Button Actuator
- Easy Paper Tape Replacement

**EASILY ASSEMBLED**
- Absolutely No Alteration of AIM-65 Required
- All Fasteners Provided
- Goes Together in Minutes

**MADE OF HIGH IMPACT STRENGTH THERMOFORMED PLASTIC**
- Kydex 100*
- Durable
- Molded-In Color
- Non-Conductive

**AVAILABLE FROM STOCK**
- Allow Three to Four Weeks for Processing and Delivery
- No COD's Please
- Dealer Inquiries Invited

---

TO ORDER: 1. Fill in this Coupon (Print or Type Please)
2. Attach Check or Money Order and Mail to:

NAME ____________________

STREET ____________________

CITY ____________________

STATE ____________ ZIP ________

Please Ship Prepaid ____ SAE 1-1(s)
@ $43.50 each
California Residents Please Pay
$46.33 (Includes Sales Tax)

**enclosures group**

**753 bush street**
**san francisco, california 94108**

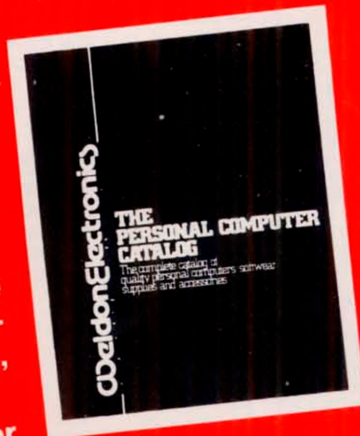
*TM Rohm & Haas

Patent Applied For

# RUN THIS PROGRAM

## 10 Enter data in form below
## 20 Goto mailbox
## 30 Mail form
## 40 Recieve the Personal Computer Catalog
## 50 End

# Well Done!




**Follow this simple program and you will receive The Personal Computer Catalog. The one reference book to fine quality personal computers, software, supplies and accessories.**

**This valuable catalog is FREE so mail your order today.**

Name __________

Address __________ State ______ Zip ______

City __________ What type? ______

Do you own a computer? ______

Do you use your computer for: Business? ______

Personal? ______ Education? ______ Other? ______

**Mail this form to:**

WeldonElectronics

COMPUTER INDUSTRY

SERVING THE PERSONAL

**Or phone: (612) 884-1475**

**Weldon Electronics**
**4150 Hillcrest Road**
**Wayzata, MN 55391**

# APPLE II® PROFESSIONAL SOFTWARE

## PIE TEXT EDITOR

PIE (PROGRAMMA IMPROVED EDITOR) is a two-dimensional cursor-based editor designed specifically for use with memory-mapped and cursor-based CRT's. It is totally different from the usual line-based editors, which were originally designed for Teletypes. The keys of the system input keyboard are assigned specific PIE Editor function commands. Some of the features included in the PIE system are: Blinking Cursor; Cursor movement up, down, right, left, plus tabs; Character insert and delete; String search forwards and backwards; Page scrolling; GOTO line number, plus top or bottom of file; Line insert and delete anywhere on screen; Move and copy (single and multiple lines); Append and clear to end of line; Efficient memory usage. The following commands are available in the PIE Text Editor and each is executed by depressing the systems argument key simulataneously with the command key desired:

| | |
|---|---|
| [LEFT] | Move cursor one position to the left |
| [RGHT] | Move cursor one position to the right |
| [UP] | Move cursor up one line |
| [DOWN] | Move cursor down one line |
| [BHOM] | Home cursor in lower left left hand corner |
| [HOME] | Home cursor in upper left hand corner |
| [-PAG] | Move up (toward top of file) one "page" |
| [+PAG] | Move down (toward bottom of file) one "page" |
| [LTAB] | Move cursor left one horizontal tab |
| [RTAB] | Move cursor right one horizontal tab |
| [GOTO] | Go to top of file (line 1) |
| [ARG] n[GOTO] | Go to line 'n' |
| [BOT] | Go to bottom of file (last line + 1) |
| [-SCH] | Search backwards (up) into file for the next occurence of the string specified in the last search command |
| [ARG] t[-SCH] | Search backwards for string 't' |
| [+SCH] | Search forwards (down) into the file for the next occurence of the string specified in the last search command |
| [ARG] t[+SCH] | Search forward for string 't' |
| [APP] | Append -move cursor to last character of line +1 |
| [INS] | Insert a blank line beforere the current line |
| [ARG] n[INS] | Insert 'n' blank lines before the current line |
| [DEL] | Delete the current line, saving it in the "push" buffer |
| [ARG] n[DEL] | Delete 'n' lines and save the first 20 in the "push" buffer |
| [DBLK] | Delete the current line as long as it is blank |
| [PUSH] | Save current line in "push" buffer |
| [ARG] n[PUSH] | Save 'n' lines in the "push" buffer |
| [POP] | Copy the contents of the "push" buffer before the current line |
| [CINS] | Enable character insert mode |
| [CINS] [CINS] | Turn off character insert mode |
| [BS] | Backspace |
| [GOB] | Gobble - delete the current character and pull remainder of characters to right of cursor left one position |
| [EXIT] | Scroll all text off the screen and exit the editor |
| [ARG] [HOME] | Home Line - scroll up to move current line to top of screen |
| [APP] [APP] | Left justify cursor on current line |
| [ARG] [GOB] | Clear to end of line |

| | | |
|---|---|---|
| **Apple PIE Cassette** | **16K** | **$19.95** |
| **TRS-80PIE Cassette** | **16K** | **19.95** |
| **Apple PIE Disk** | **32K** | **24.95** |

## 6502FORTH · Z-80FORTH 6800 FORTH

FORTH is a unique threaded language that is ideally suited for systems and applications programming on a micro-processor system. The user may have the interactive FORTH Compiler/Interpreter system running stand-alone in 8K to 12K bytes of RAM. The system also offers a built-in incremental assembler and text editor. Since the FORTH language is vocabulary based, the user may tailor the system to resemble the needs and structure of any specific application. Programming in FORTH consists of defining new words, which draw upon the existing vocabulary, and which in turn may be used to define even more complex applications. Reverse Polish Notation and LIFO stacks are used in the FORTH system to process arithmetic expressions. Programs written in FORTH are compact and very fast.

### SYSTEM FEATURES & FACILITIES

Standard Vocabulary with 200 words
Incremental Assembler
Structured Programming Constructs
Text Editor
Block 1/0 Buffers
Cassette Based System
User Defined Stacks
Variable Length Stacks
User Defined Dictionary
Logical Dictionary Limit
Error Detection
Buffered Input

### CONFIGURATIONS

| | |
|---|---|
| **AppleFORTH Cassette 16K** | **$34.95** |
| **AppleFORTH Disk 32K** | **49.95** |
| **PetFORTH Cassette 16K** | **34.95** |
| **TRS-80FORTH Cassette 16K** | **34.95** |
| **SWTPCFORTH Cassette 16K** | **34.95** |

## ASM/65 EDITOR ASSEMBLER

ASM/65 is a powerful, 2 pass disk-based assembler for the Apple II Computer System. It is a compatible subset of the FORTRAN cross-assemblers which are available for the 6500 family of micro-processors. ASM/65 features many powerful capabilities, which are under direct control of the user. The PIE Text Editor co-resides with the ASM/65 Assembler to form a comprehensive development tool for the assembler language programmer. Following are some of the features available in the ASM/65 Editor Assembler.

PIE Text Editor Command Repetoire
Disk Based System
Decimal, Hexadecimal, Octal, & Binary Constants
ASCII Literal Constants
One to Six character long symbols
Location counter addressing "*"
Addition & Subtraction Operators in Expressions
High-Byte Selection Operator
Low-Byte Selection Operator
Source statements of the form:
[label] [opcode] [operand] [;comment]
56 valid machine instruction mnemonics
All valid addressing modes
Equate Directive
BYTE Directive to initialize memory locations
WORD Directive to initialize 16-bit words
PAGE Directive to control source listing
SKIP Directive to control source listing
OPT Directive to set select options
LINK Directive to chain multiple text files
Comments
Source listing with object code and source statements
Sorted symbol table listing

### CONFIGURATION

**Apple II 48K/Disk $69.95**

## LISA INTERACTIVE ASSEMBLER

LISA is a totally new concept in assembly language programming. Whereas all other assemblers use a separate or co-resident text editor to enter the assembly language program and then an assembler to assemble the source code, LISA is fully interactive and performs syntax/addressing mode checks as the source code is entered in. This is similar in operation to the Apple II Integer BASIC Interpreter. All error messages that are displayed are in plain, easy to understand English, and not simply an Error Code. Commands in LISA are structured as close as possible to those in BASIC. Commands that are included are: LIST, DELETE, INSERT, PR #n, IN #n, SAVE, LOAD, APPEND, ASM, and a special user-defineable key envisioned for use with "dumb" peripherals. LISA is DISK II based and will assemble programs with a textfile too long to fit into the Apple memory. Likewise, the code generated can also be stored on the Disk, hence freeing up memory for even larger source programs. Despite these Disk features, LISA is very fast; in fact LISA is faster than most other commercially available assemblers for the Apple II. Not only is LISA faster, but also, due to code compression techniques used LISA requires less memory space for the text file. A full source listing containing the object and source code are produced by LISA, in addition to the symbol table

**Apple II 32K/Disk $34.95**


**PROGRAMMA INTERNATIONAL, INC.**

3400 Wilshire Blvd.
Los Angeles, CA 90010

(213) 384-0579 · 384-1116 · 384-1117




PROGRAMMA Software Products