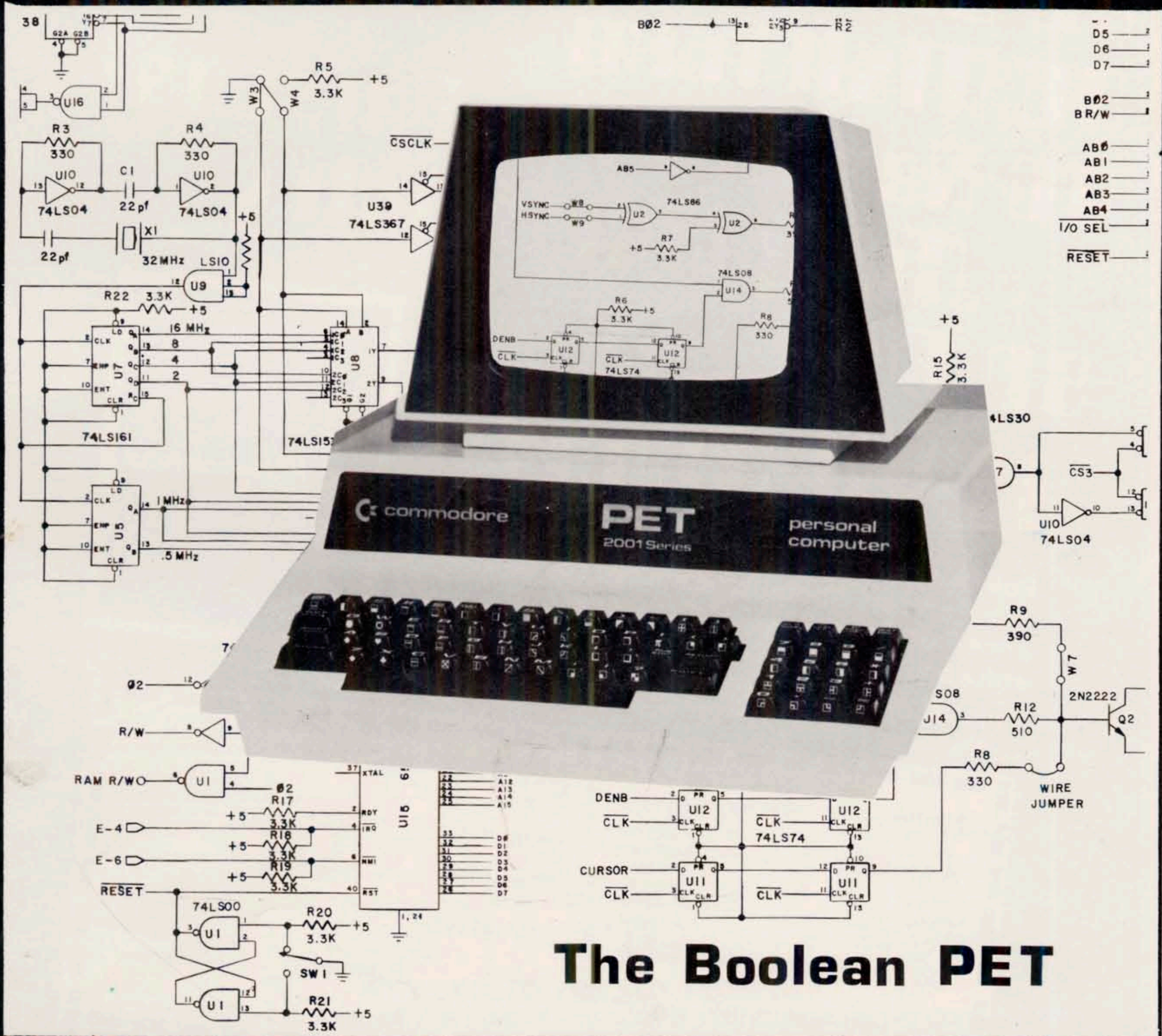
MICRO™

The Magazine of the APPLE, KIM, PET and Other 6502 Systems



The Boolean PET

NO 14 July 1979 $2.00

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 standalone 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

Software Products

Apple II is a registered trademark of Apple Computers, Inc. These professional products are available at your local computer dealer.

MICRO™

July 1979
Issue Number Fourteen

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

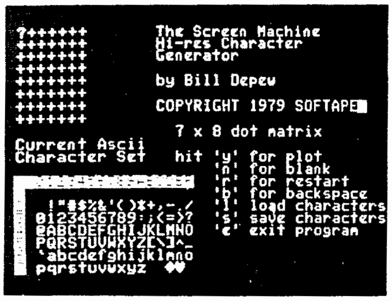
Micro-Systems Lab
James R. Witt, Jr.
Stephen L. Allen

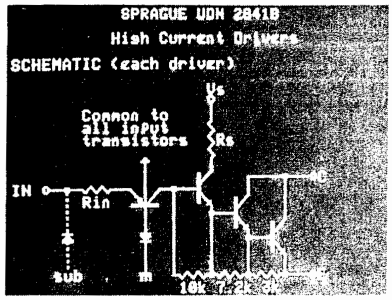
Comptroller
Donna M. Tripp

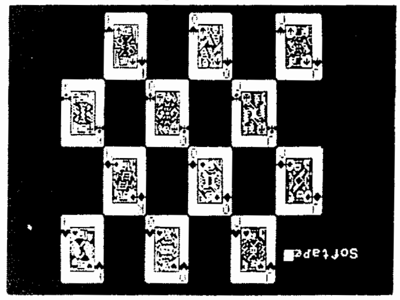
MICRO™ is published monthly by:
MICRO Ink, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515
Mailing address for all correspondence, subscriptions and address changes is:
MICRO
P. O. Box 6502
Chelmsford, MA 01824
Application to mail at second-class postage rates is pending at: Chelmsford,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

Advertiser's Index

APPLE HI-RES GRAPHICS: The Screen Machine by Softape







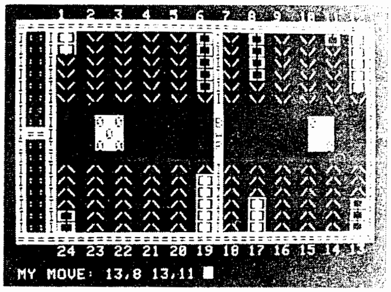
Open the manual and LOAD the cassette. Then get ready to explore the world of Programmable Characters' with the SCREEN MACHINE™. You can now create new character sets – foreign alphabets, electronic symbols and even Hi-Res playing cards, or, use the standard upper and lower case ASCII character set.

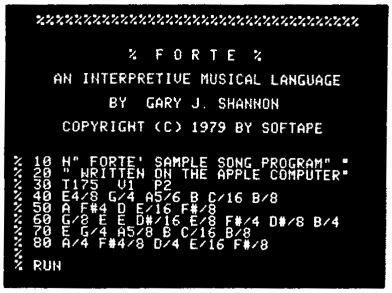
The "SCREEN MACHINE" lets you redefine any keyboard character. Just create any symbol using a few easy key strokes and the "SCREEN MACHINE" will assign that symbol to the key of your choice. For example: create a symbol, an upside down "A" and assign it to the keyboard 'A' key. Now every time you press the 'A' key or when the Apple prints an 'A' it will appear upside down. Any shape can be assigned to any key!

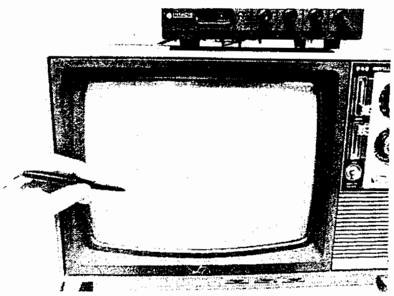
The "SCREEN MACHINE" gives you the option of saving your character symbols to disk or tape for later use. There is no complicated 'patching' needed. The SCREEN MACHINE is transparent to your programs. Just print the new character with a basic print statement. The "SCREEN MACHINE" is very easy to use.

Included on the cassette are Apple Hi-Res routines in SOFTAPES prefix format. You can use both Apple's, routines and the SCREEN MACHINE to create microcomputing's best graphics.

Cassette, and Documentation, a complete package **$19.95**







MICROGAMMON 1.0 Learn, practice and inhance your Backgammon ability with a true competitor **$14.95**

APPLE-LIS'NER Voice recognition Software. Create your own programs which 'listen' and understand 31 spoken words – English or Foreign. No hardware needed **$19.95**

APPLE TALKER Your Apple's voice. Create programs which talk to you in English or Spanish or any language **$15.95**

JUPITER EXPRESS Command your ship thru the hazards of the Asteroid belt between Mars and Jupiter **$9.95**

FORTE' A music language, written like basic, you use line numbers for your notes. You can trace line numbers or notes. You can even print the words of any song. Save your song to your Disk . **$19.95**

FORTH][Is the creation of **Wm. Graves.** This language gives you faster execution of programs than basic and is easier to program than machine language. Our 100 page manual will teach you everything you will need. **FORTH][** comes complete with demo programs on one Apple diskette **$49.95**

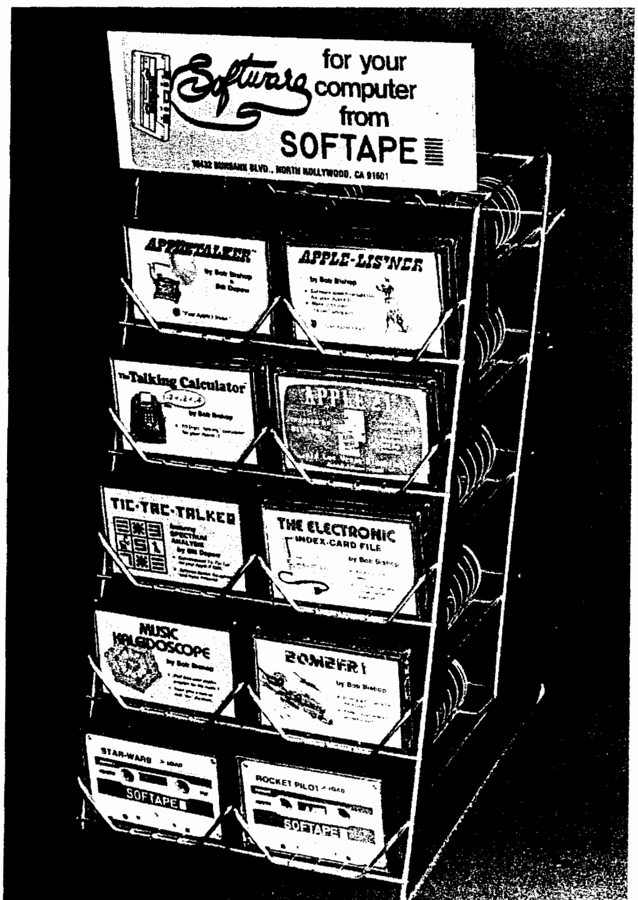
WHERE TO GET IT: Look for the SOFTAPE Software display in your local computer store. Apple dealers throughout the United States, Canada, South America, Europe and Australia carry the SOFTAPE Software line of quality products.

If your local dealer is sold out of SOFTAPE Software you can order it direct from us by check or Visa/Master Charge. If you have any questions please call us at:

VISA **1-213-985-5763** master charge THE INTERBANK CARD

Or mail your order to the address below. We'll add your name to our mailing list for free literature and announcements of new products.

BRIGHT PEN What is the difference between a light and a Bright Pen? Intelligent Software and extensive documentation **$34.95**



SOFTAPE™

10432 Burbank Blvd. • North Hollywood, CA 91601

A Baudot Teletype Driver for the APPLE II

Hard copy output can be economical if low cost surplus components are adapted to a 6502 system. Once the I/O interface has been achieved, character code incompatibility need not be a problem.

Lt. Robert Carlson, USN
NØAOT
3332 Crabapple Road
Virginia Beach, VA 23452

For many APPLE II owners, the investment in a high quality ASCII printer has to be deferred for a while and, in the interim, a printer of some sort is still highly desirable. One very inexpensive way to fill this need is to use the common Baudot Teletype. Typically, any of several models in good working order can be obtained for anywhere from $25 to $300. Large numbers of these units are made available as surplus by the telephone companies, the National Weather Service, and all branches of the Armed Forces.

As surplus, they sell for a small fraction of their original value. Of course, these Teletypes use an obsolete five bit character code, Baudot, but the following program performs the conversion from Baudot to ASCII automatically. If for some reason you need to use an ASCII character that does not convert directly to Baudot, such as the "=" sign, the program will print a space that you can fill in later. Alternatively, one could substitute some other Baudot character by changing the appropriate value in the lookup table. This problem is rarely encountered, except in certain BASIC program listings.

The program combines ideas from many other programs, but basically it is an adaptation of Chuck Carpenter's programs that appeared in MICRO 3:13 and 4:27. The program makes use of AN0, a one bit output port available on the paddle connector socket. There are no addresses used outside the program that can be "stepped on" by the system monitor or BASIC programs. While the printer is running, the characters will still appear on the video monitor normally, as they are printed.

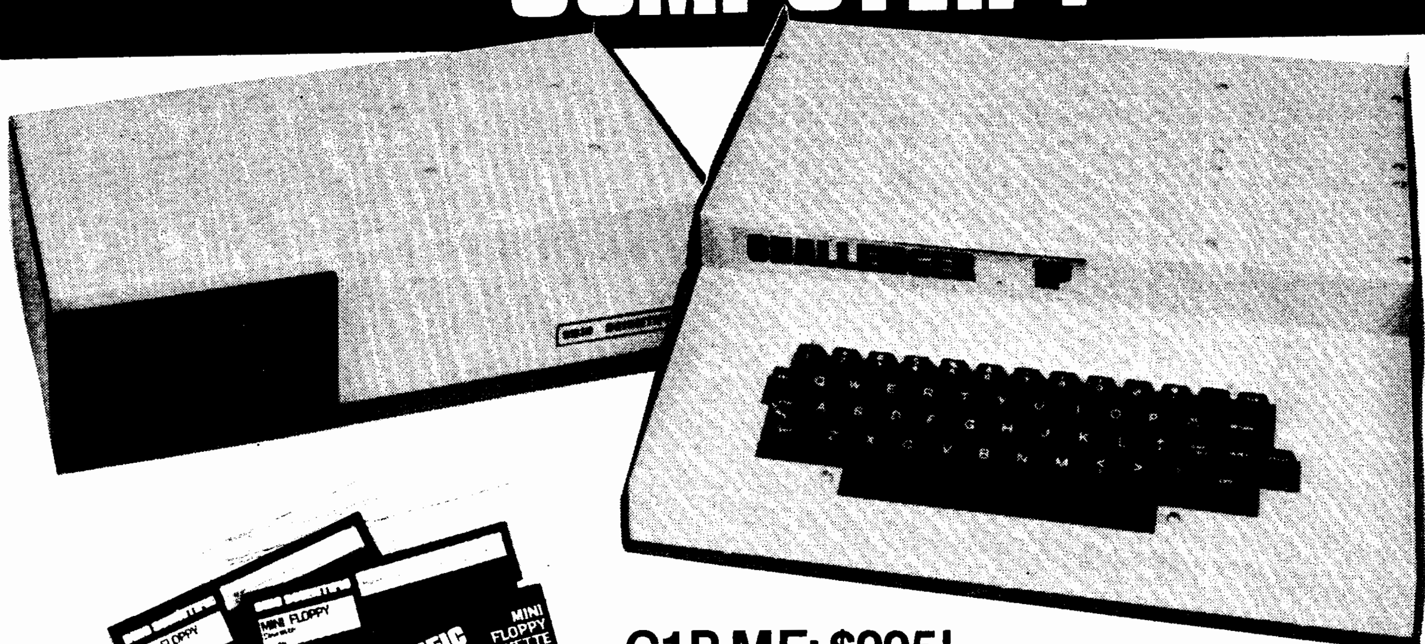
Enter the program from the monitor at $300. From Integer BASIC use a "CALL 768," and from AppleSoft use something like A = USR 768. To exit while in the monitor, hit RESET and, when in either BASIC, use "PR#0."

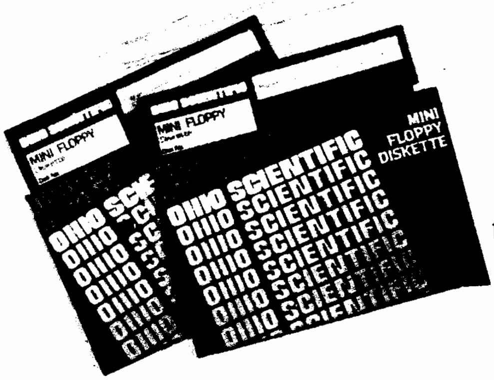
```
10 CALL 768
20 PRINT "TESTING BAUDOT DRIVER
1234567890."
30 PR#0
40 END
```

To change from 60 WPM to 100 WPM operation, change the timing value at $377 from #$5F to #$48. The output can be inverted by exchanging the values at $36F and $374.

```
                    ORG    $0300
0300 A9 09          LDAIM  $09
0302 85 36          STA    $0036
0304 A9 03          LDAIM  $03
0306 85 37          STA    $0037
0308 60             RTS
0309 8C C2 03       STY    $03C2
030C 8E C3 03       STX    $03C3
030F 48             PHA
0310 20 2D 03       JSR    $032D
0313 68             PLA
0314 C9 8D          CMPIM  $8D
0316 D0 0C          BNE    $0324
0318 48             PHA
0319 A9 00          LDAIM  $00
031B 20 2D 03       JSR    $032D
031E A9 8A          LDAIM  $8A
0320 20 2D 03       JSR    $032D
0323 68             PLA
0324 AC C2 03       LDY    $03C2
0327 AE C3 03       LDX    $03C3
032A 4C F0 FD       JMP    $FDF0
032D 29 7F          ANDIM  $7F
032F A2 3F          LDXIM  $3F
0331 DD 81 03       CMPX   $0381
0334 F0 07          BEQ    $033D
0336 CA             DEX
0337 10 F8          BPL    $0331
0339 A9 04          LDAIM  $04
033B D0 01          BNE    $033E
033D 8A             TXA
033E C9 20          CMPIM  $20
0340 B0 15          BCS    $0357
0342 2C C4 03       BIT    $03C4
0345 10 0C          BPL    $0353
0347 48             PHA
0348 A9 00          LDAIM  $00
034A 8D C4 03       STA    $03C4
034D A9 1F          LDAIM  $1F
034F 20 66 03       JSR    $0366
0352 68             PLA
0353 20 66 03       JSR    $0366
0356 60             RTS
0357 2C C4 03       BIT    $03C4
035A 30 F7          BMI    $0353
035C 48             PHA
035D A9 80          LDAIM  $80
035F 8D C4 03       STA    $03C4
0362 A9 1B          LDAIM  $1B
0364 D0 E9          BNE    $034F
0366 A0 07          LDYIM  $07
0368 18             CLC
0369 09 E0          ORAIM  $E0
036B 48             PHA
036C B0 05          BCS    $0373
036E 8D 59 C0       STA    $C059
0371 90 03          BCC    $0376
0373 AD 58 C0       LDA    $C058
0376 A9 5F          LDAIM  $5F
0378 20 A8 FC       JSR    $FCA8
037B 68             PLA
037C 6E A8 FC       ROR
037F 88             DEY
0380 D0 E9          BNE    $036B
0382 60             RTS
```

"$995 FOR A FLOPPY DISK BASED COMPUTER ?"





C1P MF: $995! First floppy disk based computer for under $1000! A dramatic breakthrough in price and performance. Features OSI's ultra-fast BASIC-in-ROM, full graphics display capability, and large library of software on disk, including entertainment programs, personal finance, small business, and home applications. It's a complete programmable computer system ready to go. Just plug-in a video monitor or TV through an RF converter, and be up and running. Can be expanded to 32K static RAM and a second mini-floppy. It also supports a printer, modem, real time clock, and AC remote interface, as well as OS-65D V3.0 development disk operating system.

VISA AND MASTER-CHARGE ORDERS ARE BOTH ACCEPTED.

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

Structured BASIC Editor and Pre-processor

Enter, list, modify and resequence BASIC programs with this versatile pre-processor for the OSI Challenger. Here is one editor that you can modify because it is written in BASIC. What's more, you can modify it in structured BASIC because the structured BASIC syntax is implemented as a bonus.

Robert Abrahamson
5533 25th Avenue
Kenosha, WI 53140

This program is a line editor and pre-processor which converts a structured BASIC program into executable BASIC statements. It is written in Microsoft BASIC and takes up about 10K of memory. Using only string operations, it changes IF THEN ELSE, DO WHILE, CASE, REPEAT UNITL, and REPEAT FOREVER structures into their equivalent forms.

Besides these constructs, it also allows the use of subroutine names. The editor portion of the program can add lines, delete single lines, delete blocks of lines, modify existing lines, print out a single line, print out a block of lines, print out the complete text, and resequence all of the lines. Table I is a list of editor commands.

The editor works by first reading in a string and comparing this string to a list of commands (see Figure 1). If it matches the string to a command, it then branches to the appropriate routine. Without a match, the program assumes that the string is a line of text. It next compares each character to a pound sign and a backwards slash. These characters are immediately changed to a comma or colon, respectively. Since BASIC does not accept commas or colons in an input string, this is a necessary inconvenience.

After this, the program tries to parse out the line number and checks for at least one non-numeric character after the line number. A missing line number initiates an error message. Thus, an illegal command would cause a message stating that one forgot the line number. On the other hand, a line number without following text would be interpreted as a request to delete that line number.

Upon finding a line number and text, it strips the line number from the text and stores the line number, separately, in a doubly linked circular list with a head node at an index of zero (see Figure 3).

The preprocessor alters the text received by the editor and returns control to the editor when processing is finished or an error is detected. First the preprocessor (see Figure 2) resequences the line numbers, insuring enough room to add lines later. The next step is to parse out the first token in the first line. This token is then compared with "SUBROUTINE." A match tells the program that this is a statement which declares a subroutine; to save the subroutine name and line number in the subroutine name table.

Matching with CASE, THEN, DO, REPEAT, ELSE, or a semi-colon requires the program to parse out the arithmetic expression, if it exists, and store it, along with a structure type code and line index, on the stack. A match with "END" causes a record to be popped from the stack, and a branch to a routine which converts that type of structure into standard BASIC statements.

If no match is found for any of these keywords, each character thereafter is compared with the ampersand, which is reserved for use only as the first character in a subroutine name. Finding an ampersand, the program parses out the subroutine name and stores it in the subroutine call table, along with line index, line length, and start and stop positions of the name. This same procedure is then repeated for every line of text. After finishing this, the subroutine call table is read, and every subroutine

Table I — Editor Command Summary

RESEQ	—	Renumbers all lines in multiples in ten.
LIST	—	Prints out entire text.
LIST X	—	X is a valid line number. Prints out only line number X. The space between LIST and X is optional.
LIST X Y	—	X is a valid line number, and Y can be any number. Prints out all lines from X to Y. There must be at least one non-numeric character between X and Y.
DEL X	—	Same restrictions as LIST X. Deletes only line number X.
DEL X Y	—	Same restrictions as LIST X Y. Deletes all lines from X to Y.
MOD X	—	Same restrictions as LIST X. Allows you to modify line number X. Program asks for a stop character and repetition.
NEW	—	Has the effect of clearing the text by breaking links.
BASIC	—	Command to start pre-processing.

name in the text is changed to a line number. This completes the pre-processing.

There are a few things to keep in mind when using this pre-processor. You should be very careful when coding GOTO statements, because the line numbers are resequenced before processing. The structured input text is altered, and so the structured text for all practical purposes is lost. As for using the structured statements, following the examples in the printout should help. Remember that in all of the structured statements spaces are necessary between words, and spaces must not be used within an arithmetic or logical expression. This is because the program uses the space, colon, and end of line to identify an expression or word ending. Multiple structured statements per line cannot be used because the program sees only the first one.

This pre-processor is relatively easy to use with a cassette interface. First enter the structured program using the editor, then convert it to BASIC with the Basic command. When you see the message stating that pre-processing is finished, type in "LIST" but do not hit return. Turn on your cassette, and then hit return. You now have the program on tape and can load it like any other program.

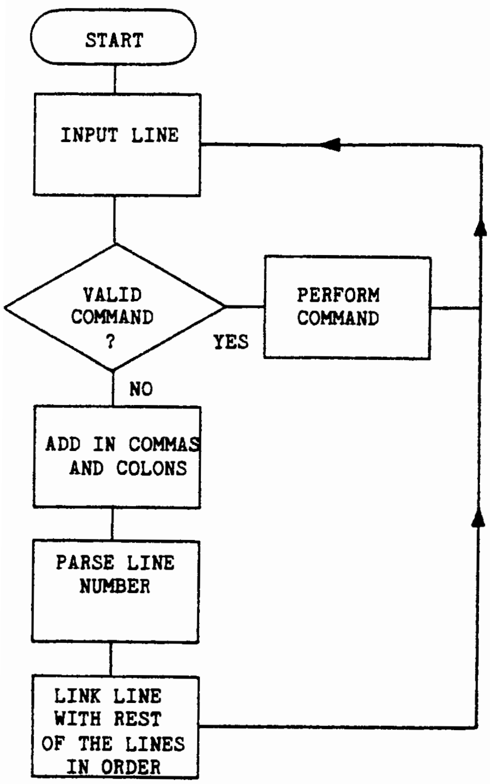


Figure 1: Editor Flow Chart

```
1 REM..........................................................
2 REM.... PRE-PROCESSOR TO CONVERT STRUCTURED BASIC TO .......
3 REM....                    BASIC                       .......
4 REM....              BY ROBERT ABRAHAMSON              .......
5 REM....                   4 MAY 79                     .......
6 REM..........................................................
10 DIM T$(100),LL(101),RL(101),LN(101),SC$(20),ST(20,4)
20 DIM SD$(20),SU(20),AR$(10),SR(10),IN(10)
30 REM... INITIATE AVAILABLE POOL OF NODES .....
40 FORI=1TO99:RL(I)=I+1:NEXTI
50 RL(100)=0:AV=1:RL(0)=0:LL(0)=0
60 INPUT S$
70 REM... DECODE COMMANDS .....
80 IFLEFT$(S$,3)="NEW"THEN30
90 IFLEFT$(S$,3)="DEL"THEN860
100 IFLEFT$(S$,3)="MOD"THEN960
110 IFLEFT$(S$,4)="LIST"THEN730
120 IFLEFT$(S$,5)="RESEQ"THENGOSUB370:GOTO60
130 IFLEFT$(S$,5)="BASIC"THEN1790
140 REM... ASSUME LINE OF TEXT .....
150 GOSUB1320:GOSUB450
160 IFP<>0THEN190
170 PRINT"OK, WHERE'S THE LINE NUMBER?"
180 GOTO60
190 IFLG>ITHEN220
200 GOSUB640:IFGN=0THEN60
210 GOSUB1220:GOTO60
220 S$=RIGHT$(S$,LG-I)
230 REM... LOCATE WHERE TO ADD IN NEW LINE .....
240 GN=LL(0)
250 IFGN=0THENAN=0:GOTO340
255 IFLN>LN(GN)THENAN=GN:GOTO340
260 IFLN<LN(RL(0))THENAN=0:GOTO340
270 GN=0
280 GN=RL(GN):IFGN=0THEN320
290 IFLN=LN(GN)THENAN=LL(GN):GOTO330
300 IFLN>LN(GN)ANDLN<LN(RL(GN))THENAN=GN:GOTO340
310 GOTO280
320 PRINT"I CAN'T FIND A SPOT FOR THE NEW LINE.":GOTO60
330 GOSUB1220
340 GOSUB1160
350 IFGN=0THENPRINT"OUT OF TEXT SPACE":GOTO60
360 GOSUB1270:GOTO60
370 REM
380 REM...RESEQUENCE ROUTINE .....
390 REM
400 GN=0:LN=10
410 GN=RL(GN)
420 IFGN=0THENPRINT:RETURN
430 LN(GN)=LN:LN=LN+10
440 GOTO410
450 REM
460 REM... FIND START OF LINE NUMBER, PARSE IT OUT .....
470 REM... INPUTS: S$=STRING TO PARSE
480 REM... OUTPUTS: P,I=START AND END OF LINE NUMBER
490 REM...               LN =LINE NUMBER
500 REM...               LG =LENGTH OF S$
510 X=1
520 LG=LEN(S$)
525 IFX>LGTHENP=0:RETURN
530 FORP=XTOLG
540 ::A=ASC(MID$(S$,P,1))
550 ::IFA>=48ANDA<=57THEN580
560 NEXTP
570 P=0:RETURN
580 FORI=PTOLG
590 ::A=ASC(MID$(S$,I,1))
600 ::IFA<48ORA>57THENI=I-1:GOTO630
610 NEXTI
620 I=I-1
630 LN=VAL(MID$(S$,P,I-P+1)):RETURN
640 REM
650 REM... SUBROUTINE TO FIND LINE NUMBER .....
660 REM...    INPUT   LN=LINE NUMBER        .....
670 REM...    OUTPUT  GN=INDEX              .....
```

```
680 REM
690 GN=0
700 GN=RL(GN):IFGN=0THENRETURN
710 IFLN=LN(GN)THENRETURN
720 GOTO700
730 REM
740 REM... LIST ROUTINE .....
750 REM
760 GOSUB450
770 IFP=0THEN820
780 GOSUB640
790 IFGN=0THENPRINT:GOTO60
800 X=I+1:GOSUB520:IFP=0THENPRINTLN(GN);T$(GN):GOTO60
801 PRINTLN(GN);T$(GN):GN=RL(GN)
802 IFLN(GN)<=LNANDGN<>0THEN801
810 GOTO60
820 GN=0
830 GN=RL(GN):IFGN=0THENPRINT:GOTO60
840 PRINTLN(GN);T$(GN)
850 GOTO830
860 REM
870 REM... DELETE COMMAND PROCEDURE .....
880 REM
890 GOSUB450
900 IFP<>0THEN920
910 PRINT"WHERE'S THE LINE NUMBER?":GOTO60
920 GOSUB640
930 IFGN=0THENPRINT"LINE NOT FOUND":GOTO60
940 X=I+1:GOSUB520:IFP=0THENGOSUB1220:GOTO950
941 G1=RL(GN):GOSUB1220:GN=G1
942 IFLN(GN)<=LNANDGN<>0THEN941
950 PRINT"DELETED":GOTO60
960 REM
970 REM... MODIFY COMMAND PROCEDURE .....
980 REM
990 GOSUB450
1000 IFP=0THENPRINT"NO LINE NUMBER":GOTO60
1010 GOSUB640
1020 IFGN=0THENPRINT"NOT FOUND":GOTO60
1030 PRINTLN(GN);T$(GN)
1040 PRINT"STOP CHARACTER";:INPUTST$
1050 PRINT"REPETITION";:INPUTF
1060 PRINTLN(GN);
1070 FORP=1TOLEN(T$(GN))
1080 ::PRINTMID$(T$(GN),P,1);
1090 ::IFMID$(T$(GN),P,1)=ST$THENF=F-1
1100 ::IFF=<0THEN1120
1110 NEXTP
1120 INPUTST$
1130 S$=LEFT$(T$(GN),P)+ST$
1140 GOSUB1320:T$(GN)=S$
1150 GOTO800
1160 REM
1170 REM... SUBROUTINE GETNODE GN FROM POOL .....
1180 REM
1190 IFAV<>0THEN1210
1200 PRINT"OUT OF NODES":GN=0:RETURN
1210 GN=AV:AV=RL(AV):RETURN
1220 REM
1230 REM... SUBROUTINE DELETE NODE GN FROM LIST .....
1240 REM
1250 RL(LL(GN))=RL(GN):LL(RL(GN))=LL(GN)
1260 RL(GN)=AV:AV=GN:RETURN
1270 REM
1280 REM... SUBROUTINE ADD NODE GN TO RIGHT OF AN .....
1290 REM
1300 RL(GN)=RL(AN):LL(GN)=AN:LL(RL(AN))=GN
1310 RL(AN)=GN:LN(GN)=LN:T$(GN)=S$:RETURN
1320 REM
1330 REM... SUBROUTINE ADD IN COMMAS/COLONS TO TEXT .....
1340 REM
1350 LG=LEN(S$)
1360 FORI=1TOLG
```

INDEX	ARITHMETIC EXPRESSION	STRUCTURE TYPE CODE
IN(Q)	AR$(Q)	SR(Q)
Q POINTS TO THE TOP OF THE STACK		

STACK RECORD

LEFT LINK	LINE NUMBER	TEXT	RIGHT LINK
LL(I)	LN(I)	T$(I)	RL(I)
CIRCULAR DOUBLE LINKED LIST WITH HEAD NODE AT I=0			

LINE OF TEXT

NAME OF SUBROUTINE	START POS. OF NAME	END POS. OF NAME	LINE LENGTH	LINE INDEX
SC$(SC)	ST(SC,1)	ST(SC,3)	ST(SC,4)	ST(SC,2)
SC POINTS TO THE LAST TABLE ENTRY				

SUBROUTINE CALL TABLE

NAME OF SUBROUTINE	LINE NUMBER
SD$(SD)	SU(SD)
SD POINTS TO THE LAST TABLE ENTRY	

SUBROUTINE DEFINITION OR NAME TABLE

```
1370 ::IFMID$(S$,I,1)="#"THENST$=",":GOTO1400
1380 ::IFMID$(S$,I,1)="\"THENST$=":":GOTO1400
1390 ::GOTO1430
1400 ::S1$=LEFT$(S$,I-1)+ST$
1410 ::IFLG>ITHENS1$=S1$+RIGHT$(S$,LG-I)
1420 ::S$=S1$
1430 NEXTI
1440 RETURN
1450 REM
1460 REM... PARSE SUBROUTINE .....
1470 REM... INPUTS:     S$=STRING TO PARSE
1480 REM...             P1=START POSITION
1490 REM... OUTPUTS:    LG=LENGTH OF S$
1500 REM...             P1=START OF TOKEN
1510 REM...             P2=END OF TOKEN + 1
1520 REM...             TK$=TOKEN
1530 LG=LEN(S$):TK$="......"
1540 IFMID$(S$,P1,1)=" "THENP1=P1+1:GOTO1540
1550 FORP2=P1TOLG
1560 ::TP$=MID$(S$,P2,1)
1570 ::IFTP$=" "THEN1610
1580 ::IFTP$="&"ANDP2>P1THEN1610
1590 ::IFTP$=":"THEN1610
1600 NEXTP2
1610 TK$=MID$(S$,P1,P2-P1)
1620 RETURN
1630 REM
1640 REM...SUBROUTINE PUSH ONTO STACK
1650 REM...INPUTS: TK$=ARITHMETIC EXPRESSION
1660 REM...           SR=STRUCTURE TYPE CODE
1670 REM...           IN=INDEX
1680 Q=Q+1:IFQ>10THENPRINT"STACK OVERFLOW ERROR":STOP
1690 AR$(Q)=TK$:SR(Q)=SR:IN(Q)=IN
1700 RETURN
1710 REM
1720 REM...SUBROUTINE POP OFF OF STACK
1730 REM...OUTPUTS: TK$=ARITHMETIC EXPRESSION
1740 REM...            SR=STRUCTURE TYPE CODE
1750 REM...            IN=INDEX
1760 IFQ=0THENPRINT"STACK UNDERFLOW ERROR":STOP
1770 TK$=AR$(Q):SR=SR(Q):IN=IN(Q)
1780 Q=Q-1:RETURN
1790 REM
1800 REM... CONVERT STRUCTURED TO BASIC .....
1810 REM
1820 GOSUB370
1830 NL=0:SD=0:SC=0:Q=0:G$="GOTO":G1$="REM"
1840 G2$="THEN":G3$="IF"
1850 NL=RL(NL):IFNL=0THEN3150
1860 S$=T$(NL):P1=1:GOSUB1450
1870 IFTK$="SUBROUTINE"THEN1890
1880 GOTO1960
1890 P1=P2:GOSUB1450
1900 IFLEFT$(TK$,1)="&"THEN1930
1910 PRINT"ERROR IN SUBROUTINE NAME, NO &"
1920 PRINTLN(NL);T$(NL):GOTO60
1930 T$(NL)=G1$+T$(NL):SD=SD+1
1940 IFSD>20THENPRINT"OUT OF SUB TABLE SPACE":GOTO60
1950 SD$(SD)=TK$:SU(SD)=LN(NL):GOTO1850
1960 IFTK$="DO"THEN1980
1970 GOTO2040
1980 P1=P2:GOSUB1450
1990 IFTK$="WHILE"THEN2010
2000 PRINT"ERROR IN DO WHILE STATEMENT SYNTAX":GOTO1920
2010 P1=P2:GOSUB1450
2020 SR=1:IN=NL:GOSUB1630
2030 GOTO1850
2040 IFTK$="REPEAT"THEN2060
2050 GOTO2150
2060 P1=P2:GOSUB1450
2070 IFTK$="UNTIL"THEN2110
2080 IFTK$="FOREVER"THEN2100
2090 PRINT"ERROR IN REPEAT STRUCTURE SYNTAX":GOTO1920
2100 IN=NL:SR=3:TK$="":GOTO2130
```

```
2110 P1=P2:GOSUB1450
2120 SR=2:IN=NL
2130 GOSUB1630:T$(NL)=G1$+T$(NL)
2140 GOTO1850
2150 IFTK$="CASE"THEN2170
2160 GOTO2220
2170 T$(NL)=G1$+T$(NL)
2180 IN=NL:SR=4:TK$=""
2190 GOSUB1630:GOTO1850
2220 IFTK$=":"THEN2240
2230 GOTO2270
2240 P1=P2:GOSUB1450
2250 SR=5:IN=NL:GOSUB1630
2260 GOTO1850
2270 IFTK$="THEN"THEN2290
2280 GOTO2370
2290 P1=P2:GOSUB1450
2300 IFTK$="DO"THEN2320
2310 PRINT"ERROR IN IF-THEN DO STATEMENT SYNTAX":GOTO1920
2320 NM=LL(NL):P1=1:S$=T$(NM):GOSUB1450
2330 IFTK$<>"IF"THENNL=NM:GOTO2310
2340 P1=P2:GOSUB1450
2350 SR=6:IN=NM:GOSUB1630
2360 GOSUB1850
2370 IFTK$="ELSE"THEN2390
2380 GOTO2420
2390 SR=7:IN=NL:TK$="":GOSUB1630
2400 T$(NL)=G1$+T$(NL)
2410 GOTO1850
2420 IFTK$="END"THEN2440
2430 GOTO2470
2440 IFQ>0THENGOSUB1710:GOTO2450
2445 PRINT"TOO MANY END STATEMENTS":GOTO60
2450 ON SR GOTO 2570,2720,2670,2970,2820,2980,3040
2470 FORP1=P2TOLG
2480 ::IFMID$(S$,P1,1)="&"THEN2510
2490 NEXTP1
2500 GOTO1850
2510 GOSUB1450
2520 SC=SC+1
2530 IFSC>20THENPRINT"OUT OF SUB CALL SPACE":GOTO60
2540 ST(SC,1)=P1:ST(SC,3)=P2:ST(SC,4)=LG
2550 ST(SC,2)=NL:SC$(SC)=TK$
2560 GOTO2470
2570 REM
2580 REM... CONVERT DO/WHILE STRUCTURE .....
2590 REM
2600 EN=LN(NL):DW=LN(IN)
2610 T$(NL)=G1$+T$(NL)
2620 T$(IN)=G3$+TK$+G2$+STR$(DW+10)
2630 LN=DW+1:S$=G$+STR$(EN):AN=IN
2640 GOSUB1160:GOSUB1270
2650 LN=EN-1:S$=G$+STR$(DW):AN=LL(NL)
2660 GOSUB1160:GOSUB1270:GOTO1850
2670 REM
2680 REM... CONVERT REPEAT FOREVER STRUCTURE .....
2690 REM
2700 T$(NL)=G$+STR$(LN(IN))
2710 GOTO1850
2720 REM
2730 REM... CONVERT REPEAT UNTIL STRUCTURE .....
2740 REM
2750 EN=LN(NL):DW=LN(IN)
2760 T$(NL)=G3$+TK$+G2$+STR$(EN+2)
2770 LN=EN+1:S$=G$+STR$(DW):AN=NL
2780 GOSUB1160:GOSUB1270
2790 LN=EN+2:S$=G1$:AN=GN
2800 GOSUB1160:GOSUB1270
2810 GOTO1850
2820 REM
2830 REM... CONVERT CASE STRUCTURE .....
2840 REM
2850 ED=LN(NL):S1=LN(IN):PC=ED
2860 T$(NL)=G1$+T$(NL)
2870 LN=S1+1:S$=G$+STR$(PC):AN=IN
```

```
2880 GOSUB1160:GOSUB1270
2890 T$(IN)=G3$+TK$+G2$+STR$(S1+10)
2900 IFSR(0)<>5THEN2950
2910 LN=S1-1:S$=G$+STR$(ED):AN=LL(IN)
2920 GOSUB1160:GOSUB1270
2930 GOSUB1710:PC=S1:S1=LN(IN)
2940 GOTO2870
2950 GOSUB1710:IFSR<>4THENPRINT"CASE ERROR":NL=IN:GOTO1920
2960 GOTO1850
2970 PRINT"CASE ERROR":NL=IN:GOTO1920
2980 REM
2990 REM••• CONVERT IF/THEN DO STRUCTURE •••••
3000 REM
3010 T$(RL(IN))=G$+STR$(LN(NL)):T$(NL)=G1$+T$(NL)
3020 T$(IN)=T$(IN)+G2$+STR$(LN(IN)+20)
3030 GOTO1850
3040 REM
3050 REM••• CONVERT IF THEN ELSE STRUCTURE •••••
3060 REM
3070 ED=LN(NL):T$(NL)=G1$+T$(NL):EL=LN(IN)
3080 LN=EL-1:S$=G$+STR$(ED):AN=LL(IN)
3090 GOSUB1160:GOSUB1270
3100 GOSUB1710
3110 IFSR<>6THENPRINT"IF THEN ELSE ERROR":NL=IN:GOTO1920
3120 T$(RL(IN))=G$+STR$(EL)
3130 T$(IN)=T$(IN)+G2$+STR$(LN(IN)+20)
3140 GOTO1850
3150 REM
3160 REM••• SUBSTITUTE NUMBERS FOR SUBROUTINE NAMES •••••
3170 REM
3180 IFSC=0THEN3320
3190 IFSD=0THENPRINT"ERROR-NO SUBROUTINES DEFINED":GOTO3320
3200 FORI=1TOSC
3210 ::FORJ=1TOSD
3220 ::::IFSC$(I)=SD$(J)THEN3260
3230 ::NEXTJ
3240 ::PRINT"ERROR-SUBROUTINE ";SC$(I);" NOT DEFINED"
3250 ::GOTO3310
3260 ::S$=T$(ST(I,2)):LG=LEN(S$)
3270 ::F=LG-ST(I,4):P1=ST(I,1)+F:P2=ST(I,3)+F
3280 ::TK$=LEFT$(S$,P1-1)+STR$(SU(J))
3290 ::IFP2<=LGTHENTK$=TK$+RIGHT$(S$,LG-P2+1)
3300 ::T$(ST(I,2))=TK$
3310 NEXTI
3320 PRINT"END OF PRE-PROCESSING":PRINT:GOTO60
```

```
RUN
? 10 REM EXAMPLE OF DO WHILE STRUCTURE
? 20 REM
? 30 DO WHILE X<>0ANDY<>0ANDZ<>0
? 40      FIRST STATEMENT
? 50      SECOND STATEMENT
? 60      LAST STATEMENT
? 70 END
? LIST
 10  REM EXAMPLE OF DO WHILE STRUCTURE
 20  REM
 30  DO WHILE X<>0ANDY<>0ANDZ<>0
 40      FIRST STATEMENT
 50      SECOND STATEMENT
 60      LAST STATEMENT
 70  END

? BASIC

END OF PRE-PROCESSING

? LIST
 10  REM EXAMPLE OF DO WHILE STRUCTURE
 20  REM
 30 IFX<>0ANDY<>0ANDZ<>0THEN 40
 31 GOTO 70
 40      FIRST STATEMENT
 50      SECOND STATEMENT
 60      LAST STATEMENT
 69 GOTO 30
 70 REM END
```

```
? LIST
 10  REM•••EXAMPLE OF IF THEN DO STRUCTURE
 20  REM
 30  IF X<>0
 31      THEN DO
 40     FIRST STATEMENT
 50     SECOND STATEMENT
 60     LAST STATEMENT
 70  END

? BASIC•••••LIST
 10  REM•••EXAMPLE OF IF THEN DO STRUCTURE
 20  REM
 30  IF X<>0THEN 50
 40 GOTO 90
 50     FIRST STATEMENT
 60     SECOND STATEMENT
 70     N TH STATEMENT
 80     LAST STATEMENT
 90 REM END

? LIST
 10  REM EXAMPLE OF IF THEN ELSE STRUCTURE
 20  REM
 30  IF NUMBER=0THEN 50
 40 GOTO 60
 50      PRINT"THE NUMBER IS ZERO"
 59 GOTO 80
 60 REM    ELSE
 70      PRINT"THE NUMBER IS NON-ZERO"
 80 REM END
```

```
? LIST
 10  REM EXAMPLE OF REPEAT UNTIL STRUCTURE
 20  REM
 30  REPEAT UNTIL A=0
 40      FIRST STATEMENT:SECOND STATEMENT
 50      N-1 TH STATEMENT
 60      N TH STATEMENT
 70  END

? BASIC

END OF PRE-PROCESSING

? LIST
 10  REM EXAMPLE OF REPEAT UNTIL STRUCTURE
 20  REM
 30 REM REPEAT UNTIL A=0
 40      FIRST STATEMENT:SECOND STATEMENT
 50      N-1 TH STATEMENT
 60      N TH STATEMENT
 70 IFA=0THEN 72
 71 GOTO 30
 72 REM


? LIST
 10  REM EXAMPLE USING SUBROUTINES
 20  REM
 30  GOSUB &INPUT:GOSUB&OUTPUT
 40  GOSUB &OUTPUT
 50  STOP
 60  REM NOTE THAT LINE 50 IS NOT NECESSARY
 70  REM
 80  SUBROUTINE &INPUT
 90     BODY OF SUB
 100  RETURN
 110  REM
 120  SUBROUTINE &OUTPUT
 130    BODY OF SUB &OUTPUT
 140  RETURN

? BASIC

END OF PRE-PROCESSING

? LIST
 10  REM EXAMPLE USING SUBROUTINES
 20  REM
 30  GOSUB  80:GOSUB 120
 40  GOSUB  120
 50  STOP
 60  REM NOTE THAT LINE 50 IS NOT NECESSARY
 70  REM
 80 REM SUBROUTINE &INPUT
 90     BODY OF SUB
 100  RETURN
 110  REM
 120 REM SUBROUTINE &OUTPUT
 130    BODY OF SUB  120
 140  RETURN


? LIST
 10  REM EXAMPLE OF REPEAT FOREVER STRUCTURE
 20  REM
 30  REPEAT FOREVER
 40    FIRST STATEMENT
 50    ...............
 60    LAST STATEMENT
 70  END   (REMEMBER THAT END CONCLUDES EACH

? BASIC

END OF PRE-PROCESSING

? LIST
 10  REM EXAMPLE OF REPEAT FOREVER STRUCTURE
 20  REM
 30 REM REPEAT FOREVER
 40    FIRST STATEMENT
 50    ...............
 60    LAST STATEMENT
 70 GOTO 30
```

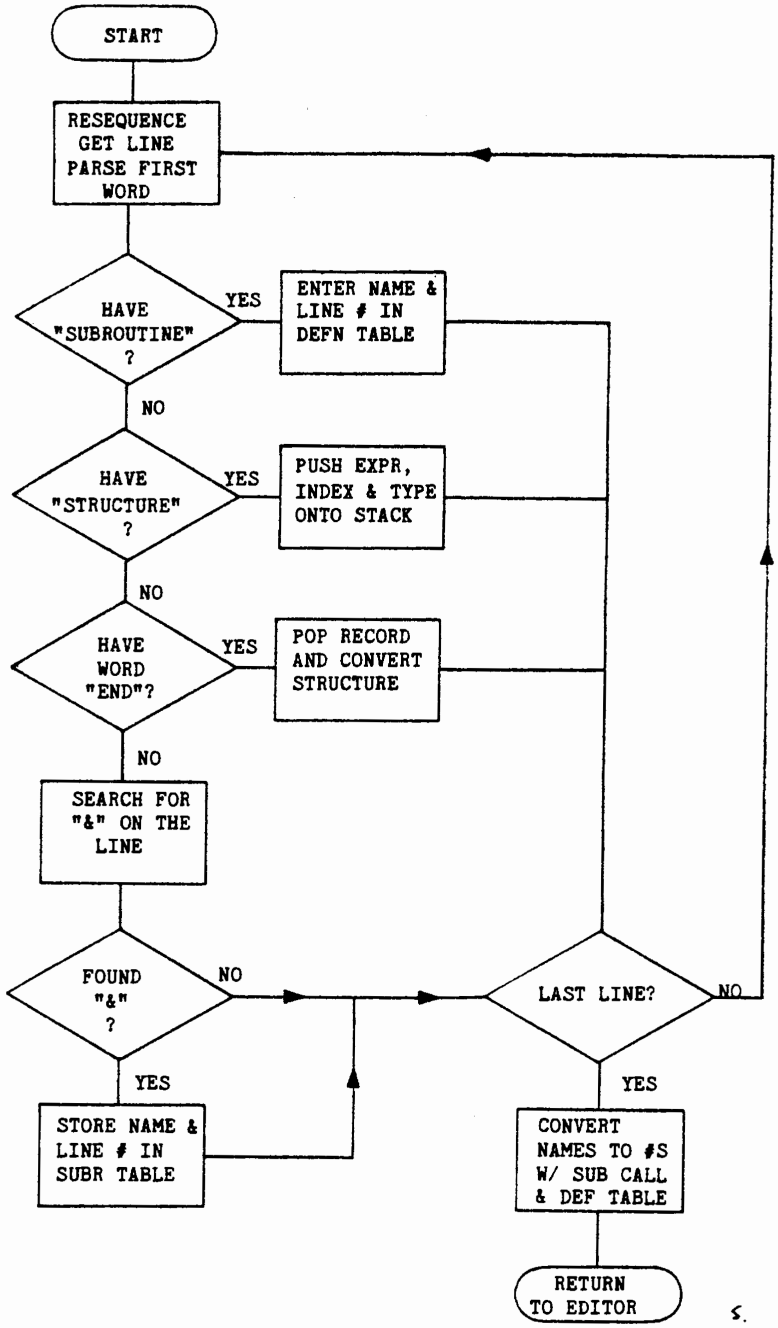


Figure 2: Pre-Processor Flow Chart

```
? LIST
 10  REM EXAMPLE OF CASE STRUCTURE, YOU CAN HAVE AS MANY CONDITIONS
 20  REM AS YOU WANT. THERE MUST BE AT LEAST ONE.
 30  REM
 40     CASE
 50      ; X=>0
 60        STATEMENT 1
 70        .   .   .
 80        STATEMENT N
 90      ; X<>0
 100       STATEMENT 1A
 110       - - - - - -
 120       STATEMENT P
 130     ; X1<>2
 140       STATEMENT
 150       STATEMENT, LAST
 160    END
```

LETTERS

Just received my May issue of MICRO today — it's getting better with every issue.

I have two 6502 systems, KIM and SYM. My KIM has an additional 28K of memory added to it, a homebrew CRT terminal, and a Selectric I/O typewriter used as output only. I used open collector TTL to interface my terminal with the KIM TTY port, but due to terminal problems, I was not able to get reliable communication until I cut the run from U15-11 to U26-10 as you described in MICRO 12:40. It does work.

I have Micro-Z's 9K + BASIC for the KIM. Bob Kurtz was very helpful in changing the data save/load routines to also include string data — I highly recommend his version. I have interfaced BASIC to the Selectric, so it is a pretty complete system.

My other system is a SYM-1 with 8K RAM and Synertek's BASIC in ROM. I use the same terminal to communicate with it as with the KIM. Their BASIC is almost the same as my KIM version, with the exclusion of the data save/load routines. Trig functions are not included but can be added with a routine that they have supplied. The trig routine occupies 313 bytes of RAM. It's handy to have BASIC in ROM but sure wish that I could change their character delete from an underline to an ASCII backspace!

I also received from Synertek an advance copy of their new monitor. The cassette problems I was having were greatly helped by it, but were not completely cleared up until I added reverse parallel diode pairs across my recorder's MIC IN and EAR lines to the SYM. I used Aud Out Hi to the recorder MIC IN with the diodes tied from Aud Out Hi to ground. The waveform generated by the SYM in HS format is non-symetrical. This caused a low frequency AC ripple to be generated by my recorder, probably due to capacitative coupling in the recorder's circuits. The diodes act as a clamp and eliminate this ripple which was quite severe for some data patterns. The cassette interface is rock-solid now.

I didn't get any listing of the new monitor, either, but the only monitor routines that I found relocated are those dealing with the cassette. I use the paper tape format to downline and upline load programs from a Honeywell L66 computer at work, and so have had the opportunity to test the changes there. They work as stated, as does the Break key on Verify. The latest info I have from Synertek says that the new monitor will be available on ROM in early July for $15.00.

```
? LIST
 10  REM EXAMPLE OF CASE STRUCTURE, YOU CAN HAVE AS MANY CONDITIONS
 20  REM AS YOU WANT. THERE MUST BE AT LEAST ONE.
 30  REM
 40 REM     CASE
 50 IFX=>0THEN 60
 51 GOTO 90
 60          STATEMENT 1
 70          .   .   .
 80          STATEMENT N
 89 GOTO 100
 90 IFX<>0THEN 100
 91 GOTO 130
 100         STATEMENT 1A
 110         - - - - - -
 120         STATEMENT P
 129 GOTO 160
 130 IFX1<>2THEN 140
 131 GOTO 160
 140         STATEMENT
 150         STATEMENT, LAST
 160 REM     END
? LIST
 10  REM SMALL PROGRAM USING SOME OF THE STRUCTURES
 20  REM
 30  PRINT:PRINT
 40  GOSUB &INPUT
 50  REPEAT UNTIL NUM=0
 60      CASE
 70        : NUM>50
 80           PRINT"THE NUMBER IS MORE THAN 50"
 90        : NUM<=50ANDNUM>10
 100          PRINT"THE NUMBER IS LESS THAN OR EQUAL TO 50";
 110          PRINT"AND GREATER THAN 10"
 120       : NUM>0ANDNUM<=10
 130          PRINT"THE NUMBER IS GREATER THAN ZERO";
 140          PRINT"AND LESS THAN OR EQUAL TO 10"
 150       : NUM<0
 160          PRINT"THE NUMBER IS NEGATIVE"
 170     END
 180     GOSUB &INPUT
 190  END
 200  STOP
 210  REM
 220  SUBROUTINE &INPUT
 230    PRINT"TYPE IN A NUMBER, TYPE ZERO TO STOP";
 240    INPUT NUM
 250  RETURN

? BASIC

END OF PRE-PROCESSING

? LIST
 10  REM SMALL PROGRAM USING SOME OF THE STRUCTURES
 20  REM
 30  PRINT:PRINT
 40  GOSUB  220
 50 REM REPEAT UNTIL NUM=0
 60 REM     CASE
 70 IFNUM>50THEN 80
 71 GOTO 90
 80           PRINT"THE NUMBER IS MORE THAN 50"
 89 GOTO 170
 90 IFNUM<=50ANDNUM>10THEN 100
 91 GOTO 120
 100          PRINT"THE NUMBER IS LESS THAN OR EQUAL TO 50";
 110          PRINT"AND GREATER THAN 10"
 119 GOTO 170
 120 IFNUM>0ANDNUM<=10THEN 130
 121 GOTO 150
 130          PRINT"THE NUMBER IS GREATER THAN ZERO";
 140          PRINT"AND LESS THAN OR EQUAL TO 10"
 149 GOTO 170
 150 IFNUM<0THEN 160
 151 GOTO 170
 160          PRINT"THE NUMBER IS NEGATIVE"
 170 REM     END
 180     GOSUB  220
 190 IFNUM=0THEN 192
 191 GOTO 50
 192 REM
 200  STOP
 210  REM
 220 REM SUBROUTINE &INPUT
 230    PRINT"TYPE IN A NUMBER, TYPE ZERO TO STOP";
 240    INPUT NUM
 250  RETURN
```

No, that was not a typo error above. I do have 8K of RAM on my SYM. U1, the address decoder, fully decodes the first 8K of memory, with only 4K implementable using the sockets provided. I added a small "piggyback" or daughter board to the SYM that fits in the area of the logo and the "Synertek Systems Corp." label. DIP plugs from this board plug into the sockets on the SYM for U12 and U19. These two 2114s plus 8 more mount on the added board. Jumper wires connect from it to U1, pins 7, 9, 10, and 11. The design violates worst case design rules since, if all the chips are providing their worst case load to the data and address lines, the lines will be loaded to higher capacitance than the 6502 is guaranteed to drive. I have all the PROM and ROM sockets full, U28 (the extra 6522) installed, and have seen no degradation of the 6502 signals with several different supplier's 2114s installed. It just will not fail a memory test! None of other SYM owners to whom I have supplied boards have had any problems either. It sure is nice to have the full 8K available for BASIC!

I can't positively guarantee that it will work for everybody, but it sure is a simple and inexpensive way to get additional memory. The PC boards with plated thru holes, reflowed solder plating, and instructions are available from me at the address below for $5.00 each, plus SASE. If it doesn't work for someone, I'll refund their money provided the board is returned undamaged.

I highly recommend the assembler/text editor supplied by M. S. S., Inc., PO Box 2034, Marshall TX 75670 for $25.00. I have modified it to run on the SYM, and I am very pleased with it. I also have Tom Pittman's Tiny Basic modified for the SYM. One can write reasonable sized programs with either of these packages and still keep within the original 4K memory size since they both take up just over 2K each. However, 8K is sure a lot better!

I'll attempt to answer any letters regarding KIM/SYM if a SASE is enclosed. Thank you, and keep up the good work!

John Blalock
3054 West Evans Drive
Phoenix, Arizona 85023

Thanks to Jim Butterfield for *Inside Pet Basic* in MICRO 8:39. His FIND and RESEQUENCE programs were useful and informative, as were his remarks concerning how PET BASIC is built. I modified FIND to run on my Ohio Scientific "C2-8P" with the following changes.

OSI BASIC user programs start at location 0301 hex while PET's start at 0401. In line 9000, change A = 1025 to A = 769 and change X = PEEK(1029) to X = PEEK (773). In line 9005, change (1029 + L) to (773 + L).

While the program will list and run with these changes, it cannot be saved on cassette without modifying lines 9005 through 9007. This is necessary because OSI software limits the line length to 72 characters and line 9005, when listed, expands to 76 characters. To correct this, change lines 9005, 9006 and 9007 to:

```
9005 FOR L=1 TO 80:Y=PEEK(773-
   +L)
9006 IF Y=0 THEN ?*256PEEK(A+3)-
   +PEEK(A+2);:RETURN
9007 IF Y=PEEK(K+L) THEN NEXT L
9008 RETURN
```

To modify RESEQUENCE, we have to know what tokens OSI BASIC uses for keywords. In Jim's RESEQUENCE program, line 60220 searches for PET keywords GOTO (137), GOSUB (141) and THEN (167). For OSI BASIC change these to 136, 140 and 160 respectively. Change all occurences of V% to V and W% to W. Then change all undimensioned variables V to U and W to Z. Change the 1025 in line 60160 to 769.

Since OSI software looks at cassette input as if it were from the keyboard, these programs can be loaded before or after the program of interest as long as there is no line number conflict.

Alvin L. Hooper
207 Self St.
Warner Robins, GA 31093

There appears to be a growing problem with APPLE Software. Some companies selling software for the APPLE are so, concerned with theft of their product, that they are resorting to self-modifying code and programs that modify certain key registers used by the APPLE monitor. This is supposed to prevent people from listing or copying the program.

This is a very short sighted position to take. The bad part of all this is the fact that any computer is difficult at best, and sometimes impossible, for the average home computer owner to operate. This particularly true with a new and unfamiliar program.

One mistake on the part of the new user can turn a $20.00 to $500.00 disk-based program into useless junk. Furthermore, the new user cannot store the program on another disk for backup or more convenient use.

We suggest you don't buy software that does any of the following:

1. Executes automatically after loading.
2. Modifies the screen memory while loading.
3. That you cannot load from disk, using the basic DOS commands.
4. That you cannot unlock using the basic DOS commands.
5. That you cannot list.
6. That you cannot change.
7. That have basic line numbers greater than 32000.
8. That you did not try in the computer store, before you bought it.

Paul Lamar
Lamar Instruments
2107 Artesia Boulevard
Redondo Beach, CA 90278

If you have the occasion to publish readers opinions of hardware products, please add my recommendation of "The Net Works" brand serial interface adapter for the PET. It comes with excellent documentation both on the IEEE-488 interface of the PET and on the RS-232 as found on terminals and modems. It also includes sample programs to assist in learning to use the relevent portion of the PET operating system. Mine has worked flawlessly for some 6 months now; this letter was typed with it, using an AJ 841 for input/output.

Also, you might warn readers that Programma Consultants version of Forth for PETs requires 16K memory to operate, contrary to their advertisements last fall.

Richard L Morgan
PO Box 25305
Houston, TX 77005

MICRO

Intercepting DOS Errors from Integer BASIC

Andy Hertzfeld
2511 Hearst Street
Berkeley, CA 94709

Implement true turnkey applications on the APPLE with this DOS error handling interface. Now Integer BASIC programs can trap errors from DOS, diagnose problems, and take remedial action with no intervention from the operator.

When a DOS error such as FILE NOT FOUND occurs during execution of a BASIC program, execution is suspended and an error message is printed. Unfortunately, this is often not what we want to happen. We would prefer for the program to be notified of the error and allowed to continue execution, dealing with the error in any fashion it desires.

This is fairly easy to achieve under AppleSoft because it includes an ONERR error intercepting facility. It is much harder to intercept errors from Integer BASIC; this article describes one method for doing so.

Unlike Integer BASIC, the DOS resides in normal RAM. This means that it can be patched to make it do almost anything we wish. It turns out that location 9D5A (for 48K systems) holds the address of the BASIC error-handling routine that DOS vectors to whenever an error arises. It usually contains E3E3, for Integer BASIC, and D865 for ROM AppleSoft. However, we can store our own address into 9D5A (5D5A for 32K systems) and thereby gain control whenever a DOS error occurs.

The following 24-byte, relocatable routine will intercept errors from BASIC. When a DOS error arises, it will store the error number at location 2; the line number of the statement that caused the error in locations 3 and 4; and, finally, it will transfer control to the BASIC statement whose line number is found in locations 0 and 1. Since the routine is relocatable, you can position it anywhere you wish. Location 300 appears to be a pretty good place, unless you are keeping your printer driver there.

To activate the error intercept facility, perform the following two POKEs which store the address of the intercept routine in $9D5A:

POKE -25254,0: POKE -25253,3
(for 48K systems) or

POKE 23898,0: POKE 23899,3
(for 32K systems)

The error intercept routine itself can be POKEd into page 3 or BLOADed off disk, whichever you prefer. If you locate it somewhere other than $300, make sure to alter the above POKEs accordingly.

After the routine is loaded into memory, it is very easy to use. If LINE is the line number of the statement where the error handling portion of your program begins, you should "POKE 0, LINE mod 256" and "POKE 1, LINE/256" to inform the interceptor where you want it to branch to. Your BASIC error-handler can figure out which statement caused the error by PEEKing at locations 3 and 4.

PEEK(3) + 256 * PEEK(4) is the line number. It can determine which type of DOS error occured by PEEKing at location $2. Table 1 gives the numbers for the various different classes of error.

Unfortunately, there is still one minor problem. Even though you regain control when a DOS error occurs, DOS still rings the bell and prints out an error message. One simple POKE will inhibit DOS from doing this but, since the POKE will supress all DOS error messages, including immediate execution errors, it is a little bit dangerous. Also, the POKE is different for different memory size systems and for different versions of DOS.

48K with DOS V3.1:	POKE -22978,20
48K with DOS V3.2:	POKE -22820,18
32K with DOS V3.1:	POKE 26174,20
32K with DOS V3.2:	POKE 26332,18

On all systems, you can restore error messages by POKEing 4 into the system-dependent address cited above.

The ability to capture DOS errors is very important, especially for turn-key systems where it is a disaster if a program crashes for any reason at all. Perhaps this little routine will allow more people to program in faster, more elegant Integer BASIC rather than choosing the AppleSoft language.

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

3030                ORG   $300
3030 86 02          STX   $0002  SAVE ERROR NUMBER
3032 A0 01          LDYIM $0001
3034 B1 DC          LDAIY $00DC  GET LOW BYTE OF ERRING
3036 85 03          STA   $0003  LINE NUMBER AND SAVE AT $3
3038 C8             INY
3039 B1 DC          LDAIY $00DC  DITTO FOR HIGH BYTE
303B 85 04          STA   $0004
303D A5 00          LDA   $0000  GET LOW BYTE OF LINE NUMBER
303F 85 CE          STA   $00CE  OF ERROR HANDLING STATEMENT
3041 A5 01          LDA   $0001  DITTO FOR HIGH BYTE; SET
3043 85 CF          STA   $00CF  THINGS UP FOR BASIC AND
3045 4C 5E E8       JMP   $E85E  LET THE FIRMWARE TAKE OVER
```

Table I — Error Numbers and Messages

Number	Message
1	Language Not Available
2	Range Error
3	Range Error
4	Write Protection Error
5	End of Data Error
6	File Not Found Error
7	Volume Mismatch Error
8	Disk I/O Error
9	Disk Full Error
10	File Locked Error
11	Syntax Error
12	No Buffers Left Error
13	File Type Mismatch
14	Program Too Large Error
15	Not Direct Command

Note that these are error messages for DOS V3.2; the V3.1 messages are slightly different.

AIM Your Spouse toward Success at the Supermarket

Melville Evans and Vernon Larrowe
Environmental Research Institute of Michigan
3300 Plymouth Road
Ann Arbor, MI 48107

This grocery list generator requires no programming. It will prove that your computer really is a useful gadget just one hour after you unpack it from shipment. Today the supermarket. And Tomorrow?

If she's like my wife Marie, she looks at you, sweating over software, with a tolerant smile. Nothing useful will come of it, but it keeps you off the street, and it's probably cheaper than a sailboat. If that's your picture, take note: here's a "program" that needs no home-built software, that you can get running the first time you fire up your AIM, that demonstrates most of the neat AIM features, and that several local computer-owner's wives agree provides a really useful function.

Well, only two that have actually tried it so far, but that's two out of two, and the rest all say it sounds good. Marie says it saves her time making her list, saves time in the store, and prevents her arriving back home and realizing she forgot the beer. It takes an hour to gather the data, and a half-hour to type it in. Then your wife sits down at the "console", runs it, and it works the first time. Here's how.

Gather the data. The next time she goes to the supermarket, go with her, armed with notebook and pencil. Ask her to take her usual route through the store and to point out, as she goes, any item she sometimes buys. Not just those she's buying today, but anything she ever buys. Note them down in order, with current prices if you have time. You can come back for prices later, if they prove useful. Ask her to be specific. Not to say just "canned vegetables", but to specify which canned vegetables she sometimes buys. Peas? Carrots? If she walks right by the beer without seeing it, put it on the list anyhow.

Type it in. Fire up your AIM and call the editor, with all of RAM for the buffer, and input from the keyboard (i.e. hit "E, SP, SP, SP"). Now type in your list, in the same order you gathered it, abbreviated to one item per line. My list is shown in Figure 1. It's a long list, and takes a little over 2K of RAM. If you only have 1K to work with, you may have to delete some items later, but try putting them all in. It's surprising how many lines 1K will hold.

Dump it to cassette. So you can load it next week. It's supposed to save time, remember.

Try it yourself before you demonstrate. Escape to the monitor and turn the printer off (ESC, CTRL PRINT OFF). Now pretend you're going grocery shopping. Hit "T", and there's your first line on the display. If you have a title at the top, use "D" to step down to the first item. Need that this time? No? Hit "D", and there's the next item. Need that? Yes? Hit "'PRINT", and it goes on the list. Now "D" for the next item. Just step down the list with "D", and hit "PRINT" for any item you want on today's shopping list. If you change your mind after hitting "D", you can back up with "U".

When you finally get to "END", hit "LF" about six times, tear off the paper, and there's your list. All neatly typed, and in the order you'll find them in the store, and with the beer on there, by golly!

If you find some lines that need changes, feel free. You're in the editor, after all, and "C" is fun to use. But remember to dump the new version onto cassette before you sign off.

Call your wife. Before she sits down to it for the first time, be sure it's properly loaded, with printer off, and displaying Item One. You're trying to impress her, both with AIM and with your expertise, right? It detracts from the impression if you blow the first tape load and have to do it again, and then kick the plug out of the wall as you swing out of the chair.

After she sees the payoff, she may even agree that it's worth putting up with hassles like that!

```
GROCERY LIST--
  (PRICES AS OF
    24 MARCH 1979)

APPLES .32/LB
ORANGES .30/LB
BANANAS .15/LB
CANTALOUPE .97
GRAPEFRUIT .33
LETTUCE .48
GR. ONIONS .25
GR. PEPPER .33
TOMATOES .
CUCUMBERS .33
CELERY .69
EGGPLANT .59
MUSHROOMS
ASPARAGRAS 1.39/LB
COB CORN .20
CARROTS .38/LB
YEL. ONIONS .20/LB
ZUCCINI .69/LB
CABBAGE .48/LB
IDAHO POTAT. .99/8LB
SWEET POTAT.
BOLOGNA 1.98/LB
FRANKS .98/LB
FRENCH BREAD .25
BACON
SAUSAGE
DISH.LIQUID .39/QT
AJAX .42
SOS .77
HAND SOAP .20/4.75OZ
SPIC&SPAN .41/LB
TRASH BAGS 9.40/BAG
FABRIC SOFT.
WINDEX
FURN.POLISH
CLOROX .38/LB
LIGHT BULBS
MATCHES
P. TOWELS .61/ROLL
SANDW.BAGS 1.30/BAG
TOILET PAPER .77/4
PLAS.WRAP 1.20/FT
ALUM.FOIL 1.20/FT
WAX PAPER .57/BOX
LUNCH BAGS .61/100
P. NAPKINS .59/140
KLEENEX .47/200
L'EGGS 1.74
CAT FOOD 1.88/4LB
K.LITTER 1.69/25LB
ASPIRIN 1.07/100
BAND-AIDS
MARSHMALLOWS .58/LB
POPCORN .49/LB
PEANUTS 1.18/12OZ
OATMEAL .58
RAISIN BRAN 1.17/LB
SPECIAL K 1.18/15OZ
RICE CRISP. .96/13OZ
SHAMPOO
APPLESAUCE .47/LB
CRANB.JELLY .47/CAN
FRUIT COCKTAIL
CAN.PEACHES
CAN.PEARS
MARASCH.CHERRIES
SL.PINEAPPLE
GLASSWARE
SANDW.BRD. .93/1.5LB
HAMB.BUNS
HOTDOG BUNS
CRESCENT ROLLS .66
REFRIG.BISC. .43
V8 JUICE .41/24OZ
LASAN.NOODLES .48/LB
SALTINES .50/LB
POTAT.CHIPS
FRITOS
SNACK CRACKERS
COOKIES
POP
SPRY 1.44/2LB10OZ
MAZOLA 1.37/QT
FLOUR .77/5LB
BISQUICK .99/40OZ.
CAKE MIX
PIE FILLING
CHOC.CHIPS
BAKING CHOC.
JELLO
JEL.PUDDING
SALT
CINNAMON
PEPPER
CAN.MILK .38/CAN
PANCAKE MIX
PANCAKE SYRUP
BAKING SODA
POWD.SUGAR
BROWN SUGAR
KARO
BAK.POWDER
PICKLES
RELISH
OLIVES
CANNED TUNA
MUSTARD
C.BEEF HASH
SPAM
CATSUP
BBQ SAUCE
STEAK SAUCE
TACOS
VINEGAR
SALAD DRESS.
MAYONNAISE
BAKED BEANS
PEANUT BUTTER
MAC.&CHEESE
HAMB.HELPER
RAVIOLI
MILK 1.71/GAL
COTT.CHEESE .83/LB
SOUR CREAM
CAN.SOUP
INST.SOUP
CAN.CORN .26
CAN MUSHROOMS
CAN.VEGETABLES
SAUERKRAUT
NOODLES
C.ARTICHOKES
C.TOMATOES .44/28OZ
CAN.POT.SALAD .57
TOM.SAUCE .18/8OZ
INST.MASH.POTATOES
SOUP STARTER .97
RICE 1.77/3LB
CAN GRAVY .33
STUFF.MIX 1.79/LB
COCOA 1.21/LB
TV DINNERS
FROZ.ORANGE JUICE
TEA BAGS
COFFEE FILTERS
FROZ.DONUTS
CAN.COFFEE 4.66/2LB
INST.COFFEE
COFFEE CAKE
TARTAR SAUCE
CHEDDAR CHEESE
SWISS CHEESE
MARGERINE .54/LB
EGGS .73/DOZ
FROZ.BROCCOLI
FROZ.CAULIFLOWER
FROZ.PEAS
FROZ.BRUSS.SPROUTS
FROZ.SPINACH
COOLWHIP
ICE CREAM
POTATO CHIPS
BEER
WINE
MAGAZINES
CIGARETTES

END
```

***** AIM-65 *****

EXCERT INCORPORATED

P/N		Qty 1-9
A65-1	AIM-65 w/1K RAM	$375
A65-4	AIM-65 w/4K RAM	$450
A65-A	Assembler ROM	$85
A65-B	BASIC ROMS	$100

EXCERT has concentrated on the AIM-65 to guarantee YOU that the products we offer are fully compatible. We know how these products work since they are used in our systems. EXCERT can help you get the system YOU want!

ACCESSORIES

	P/N		
	PRS1	+5V at 5A, +24V at 2.5A, ±12V at 1A (does not fit inside ENC1)	$95
	PRS2	+5V at 5A, +24V at 1A (mounts inside ENC1)	$50
	ENC1	AIM-65 case w/space for PRS2 and MEB1 or MEB2 or VIB1	$45
NEW!	ENC2	ENC1 w/PRS2 inside	$100
	TPT1	Approved Thermal Paper Tape, 6/165' rolls	$10
NEW!	MCP1	Dual 44 pin Mother Card takes MEB1, VIB1, PTC1	$80
	MEB1	8K RAM, 8K Prom sockets, 6522 and programmer for 5V Eproms (2716)	$245
NEW!	PTC1	Prototype card same size as KIM-1, MEB1, VIB1	$40
	VIB1	Video bd w/128 char, 128 user char, up to 4K RAM, light pen and ASCII keybd interfaces	$245
NEW!	MCP2	Single 44 pin (KIM-4 style) Mother Card takes MEB2, PGR2 and offers 4K RAM sockets	$119
		w/4K RAM	$169
	MEB2	16K RAM bd takes 2114's	$125
		w/8K RAM	$225
		w/16K RAM	$325
NEW!	PGR2	Programmer for 5V Eproms w/ROM firmware, up to 8 Eproms simultaneously	$195
		w/4 textool skts	$245

SYSTEMS

"ASSEMBLED & TESTED"

All AIM-65 systems are self contained and have the power supply (PRS2) mounted inside ENC1.

"STARTER" SYSTEMS

P/N		
SB65-1	A65-1 in ENC2	$475
SB65-4	A65-4 in ENC2	$540
SB65-4B	Same Plus BASIC	$640

"EXPANDED" SYSTEMS

P/N		"B" MEB1	"C" MEB2	"D" VIB1
E_65-4	A65-4, ENC2, w/one MEB1, MEB2, or VIB1	$775	$855	$775
E_65-4B	Same Plus BASIC	$875	$955	$875

Higher quantities and systems with other options quoted upon request!

Mail Check or Money Order To:

EXCERT, INCORPORATED
Attn: Laurie
4434 Thomas Avenue South
Minneapolis, Minnesota 55410
(612) 920-7792

Add $5.00 for shipping, insurance, and handling.

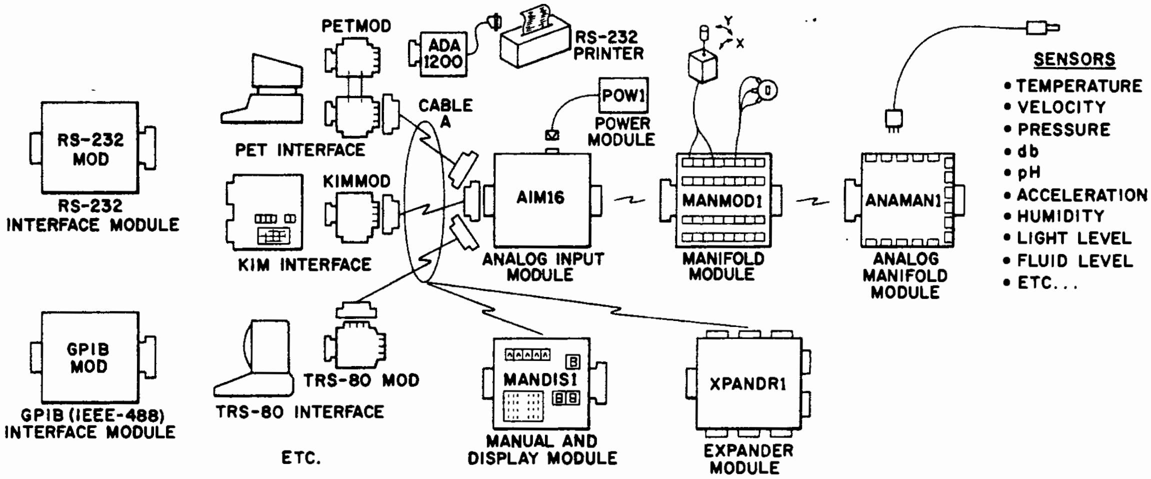
Minnesota residents add 4% sales tax.

CONNECTICUT microCOMPUTER, Inc.

150 POCONO ROAD - BROOKFIELD, CONNECTICUT 06804

(203) 775-9659

DAM SYSTEMS



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.

Boolean Equations Reduced on the PET

A deceptively small BASIC program trains the PET to perform computer aided logic design. It will reduce any single output process to a minimal, two level network.

Alan K. Christensen
1303 Suffolk Street
Austin, TX 78723

When a home experimenter tries to design a device, there are often one or two chips he doesn't have on hand. The builder might stop and order parts, then wait for delivery; but often this problem can be solved by falling back on basic gates and keeping some of these on hand for emergencies.

Reducing a truth table to an acceptable number of equations is often a tedious task. As an aid in this endeavor, I wrote a program to solve the Boolean equations using my PET computer. The program is based on the Quine-McCluskey method. It will reduce any sum of products to a minimum, two level network.

The general approach used in the program is to reduce the number of inputs using the equation

$$X'Y + XY = Y$$

And then reduce the number of terms using the equation

$$XY + V'Z + YZ = XY + XZ$$

This program works only for multiple inputs producing a single output, but it can be a powerfull aid in multiple output networks too.

The output of a network can be defined as all of the inputs for which a "1" is wanted. In addition, there may be some conditions where you don't care what the output is because that input condition will never be present. For this program, the "don't cares" are assigned in such a way as to reduce the number of inputs to required terms, but they are not considered when choosing the terms necessary for the output.

This routine is written in modules. An explanation of the function of each module will aid in translating the program into other languages. Important facts about PET BASIC are: if there are multiple statements on the same line after an IF THEN combination, none will execute when the condition is false. All variables are zero unless otherwise set, and a zero subscript is permitted in arrays.

The code with line numbers 0-99 performs general set up. Important global variables are: A$ — an array of required and don't care terms, B$ — an array of only required terms, A — an array of flags for A$, Q — an array of flags for B$, B — the number of required terms (-1), N and N2 — the number of terms in A$, and L — the number of input variables for each term.

The module 100-399 is for the data input. For this input scheme the user types in the input combinations for which a 1 output is desired. These can be either strings of zeroes and ones or upper and lower case letters. If there are don't cares present, the user enters "X" and follows with the don't care terms. The last input is followed by "END".

If the user wants to create a different input, such as from a tape or a truth table, the important results are: B$ should contain terms which have a "1" output, where the first entry is B$(0). B should equal the highest index of B$, A$(0-N) contains all the terms of B$ plus any don't care terms. N and N2 both equal the highest index of A$. Arrays A and Q should both equal zero for all entries, and L should equal the number of input variables.

Module 400-449 is where the literals are reduced from the terms. Each term is compared to every other term and, if they differ by only one variable, the

```
500 REM -COMPARE DIFFERENCES IN TERMS-
505 N$=''
510 D=0
515 FORM=1TOL
520 C$=CHR$(FNA(I))
525 IF FNA(I)=FNA(J) THEN 535
530 D=D+1:C$='-'
535 N$=N$+C$
540 NEXT M
545 RETURN
550 REM -ADD TERM TO LIST-
553 IFN2=N THEN 595
555 FOR X=0 TO N2
560 IF N$=A$(X) THEN RETURN
565 NEXT X
570 IF I=0 THEN 595
575 FOR X=0 TO I-1
580 IF A(X)=0 THEN 590
585 A(X)=0:A$(X)=N$:RETURN
590 NEXT X
595 N2=N2+1:A(N2)=0:A$(N2)=N$:RETURN
600 REM -REMOVE REDUCED TERMS FROM LIST-
605 I=0:J=N2
610 IF A(I)=0 AND I=<J THEN I=I+1:GOTO 610
615 IF A(J)=1 AND I=<J THEN J=J-1:GOTO 615
620 IF I>J THEN 635
625 A$(I)=A$(J):A(I)=0:I=I+1:J=J-1
630 GOTO610
635 N=J:N2=J
645 RETURN
650 REM -COUNT DIFFERENCE IN TERMS (DISREGAURD DON'T CARES)-
655 D=0
660 FORM=1TOL
665 IF FNB(I)=FNA(J) THEN 680
670 IF FNA(J)=45     THEN 680
675 D=D+1
680 NEXT M
685 RETURN

READY.
```

variable is replaced by a don't care (–). The new term is added to the list, and the two combined terms are marked for later removal. The process continues until the program loops through the entire list without further reductions.

In module 450-499, the reduced terms in A$ are matched against the original terms in B$. Each required term is matched with the most-reduced term that covers it.

Module 500-549 is used to compare different terms in A$. I and J are the index values of the terms. The routine returns the number of variable differences in D. N$ is the reduced expression and is only valid if D = 1.

In lines 550-599, a term N$ is added to A$ outside the range of the present loop. It is designed to conserve memory. No term will be added which is already in the list. The process usually generates duplicate terms, and it will place the new terms at the front of the list if those terms are marked for removal by A(I) = 1.

Module 600-649 removes all terms which were reduced but did not get removed in lines 550-599. It resets N and N2 to point to the end of the new list. The module from 650-699 compares terms in B$ to A$. I is the index of the B$ term and J indexes A$. In this routine, a comparison of any single variable in B$ is considered a match with A$ if the variables are equal or if the corresponding varialbe in A$(J) is a don't care, ASCII 45. The difference is returned in D.

Module 700-799 finds the most restricted term in B$. The key to arriving at the minimum solution, as opposed to just a valid solution, is to find each required term with only one reduced term to satisfy it, an essential term. If all of them have more than one possible term, we select the term in B$ which could be satisfied by the least usefull term from A$.

This is so that bad matches can be avoided early and, in the case of cyclic expressions which have several equivalent but different solutions, so that evaluation will not introduce redundant terms.

In lines 800-899, the reduced terms are sorted to bring the terms that satisfy the most conditions to the beginning of the list. This insures that the best choice will be found first.

The last module, at lines 900-999, locates the minimum number of reduced terms which satisfy the problem. The most restricted B$ term is paired with its best match in A$, and all other terms in B$ which are also satisfied are removed from further consideration.

If the flag W is set to one, it means more than one solution exits for this problem.

A	B	C	D	V	W	X	Y	Z
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

Table 1: Four-bit Binary to 5-bit BDC Conversion Map

Usually the other solutions can be found by entering the terms in a different order. Sometimes, when there is more than one solution, the most economical solution will not be the first one found. This problem could be cured by generating all of the multiple solutions, but that would require more than the 8K of memory I had available.

The result might be further reduced by going to a three level solution. This again requires more than 8K, but it would be reasonable to feed intermediate results into a second program to obtain a completely reduced result.

The idea is to look for pairs of terms, each with a variable that matches with a don't care variable in the other term, and matching in all other variables. The matching terms can be combined by ANDing with the non-matching terms, making an OR at the next level. Terms that match in some variables but not in others can be combined in a next level of the matching gates with the differing variables in the lower level.

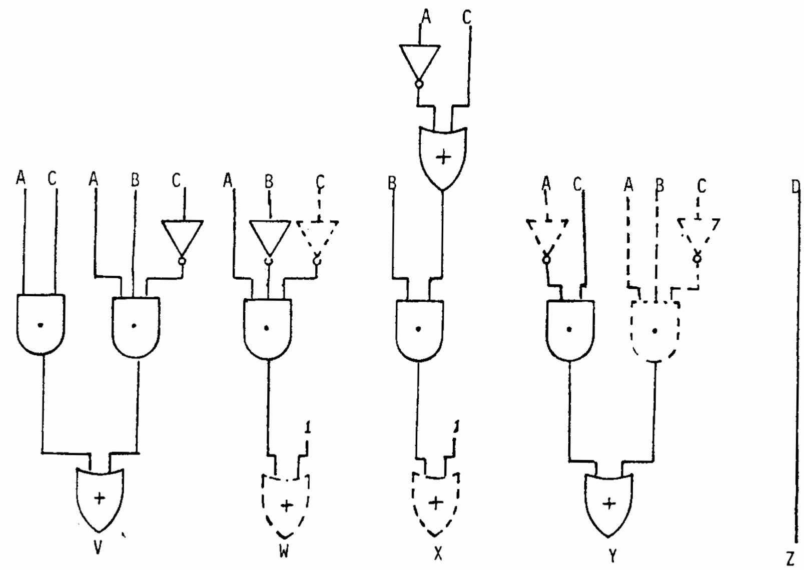


Figure 1

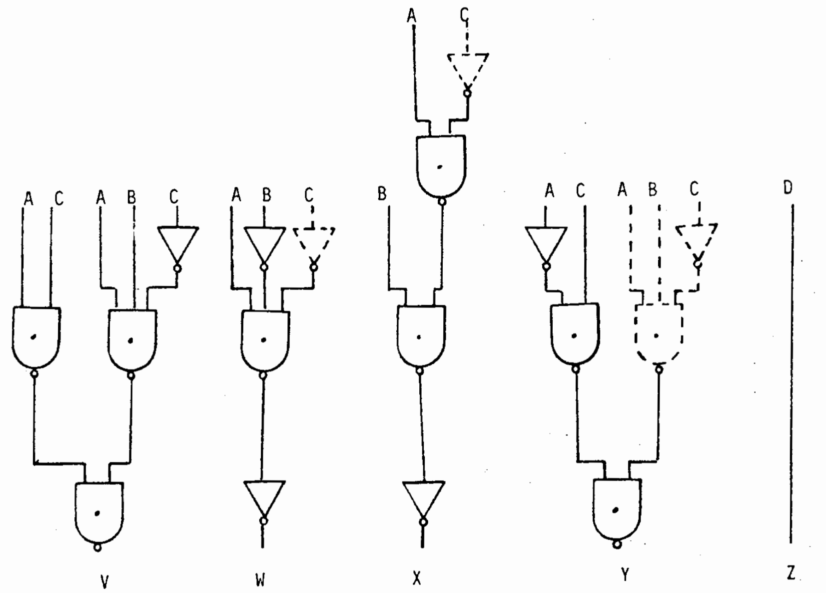


Figure 2

making an OR at the next level. Terms that match in some variables but not in others can be combined in a next level of the matching gates with the differing variables in the lower level.

I have not yet been able to determine whether my method will result in the minimal equation. As of now, no technique for this problem is known. The following example will illustrate the entire process.

The problem is to convert a 4 bit number into BCD (5 bits). The truth table for this conversion is shown in Table 1. We begin by entering the inputs for which we want output V to be true (1). The sequence is:

? 1010
? 1011
? 1100
? 1101
? 1110
? 1111
? END

and the computer replies, after a short delay, with:

1 – 1 –
11 – –

This signifies that the minimum two level solution for V is AC + AB. The process is repeated for the rest of the outputs giving results of:

```
700 REM -PUT MOST RESTICTED TERM AT BEGINNING OF LIST
705 FORI=0TOB
710 Q(I)=0:T=B
715 FORJ=0TON2
720 GOSUB 650
725 IF D=0 THEN Q(I)=Q(I)+1:IFA(J)<T THEN T=A(J)
730 NEXT J :Q(I)=Q(I)+T/10000: NEXT I
735 IF B=0 THEN 755
740 FOR I=1TOB
745 IFQ(I)<Q(0)THENN$=B$(I):B$(I)=B$(0):B$(0)=N$:X=Q(I):Q(I)=Q(0):Q(0)=X
750 NEXT I
755 RETURN
800 REM -PUT REDUCED TERMS WHICH COVER THE MOST AT THE FRONT OF THE LIST-
805 FORJ=0TON2
810 A(J)=0
815 FORI=0TOB
820 GOSUB 650
825 IF D=0 THEN A(J)=A(J)+1
830 NEXT I : NEXT J
835 FOR I=0TON2-1
840 FOR J=I+1 TO N2
845 IF A(I)>A(J) THEN 860
850 N$=A$(I):A$(I)=A$(J):A$(J)=N$
855 X=A(I):A(I)=A(J):A(J)=X
860 NEXT J : NEXT I
865 RETURN
900 REM-FIND ESSENTIAL TERM AND ELIMINATE ALL ORIGINAL TERMS THAT IT COVERS
905 GOSUB 800:GOSUB 700:I=0:J=0
910 GOSUB 650
915 IF D>0 THEN J=J+1:GOTO 910
920 IF Q(0)>=2THEN W=1
925 GOSUB 975
930 GOTO 950
935 GOSUB 650
940 IF D>0 THEN I=I+1
945 IF D=0 THEN GOSUB 975
950 IF I<=B THEN 935
955 N$=A$(J):A$(J)=A$(N2):A$(N2)=N$:N2=N2-1
960 RETURN
975 N$=B$(I):B$(I)=B$(B):B$(B)=N$:B=B-1:RETURN
```

W = 100 –   A B′ C′
X = 01 – – , – 11 –   A′B + B C
Y = 110 – , 0 – 1 –   A B C′ + A′ C
Z = – – – 1   D

The next step is to input the values for output which have a reasonable number of identical terms. For example, V and X have inputs of 1110 and 1111 in common. To see if sharing a gate will reduce the equations, we enter V again with those terms as don't cares. The input sequence is:

```
?1010
?1011
?1100
?1101
?X
?1110
?1111
?END
```

The output is the same as before; therefore, no gates are saved by combining these terms. When the same thing is tried with V and Y we get a shared equation of 110 – (which is already a term of Y) and re-entering V with 1100 and 1101 as don't cares gives an output of 1 – 1 – which indicates that we can save a gate by using V = AC + ABC′.

Further testing shows no more gates can be saved by this method, so the next step is to try to increase the levels. X is the only output which has terms that differ only at don't cares. 01 – and – 11 – can combine to (0)1(1) –, or B (A + C).

This leads directly to the circuit of Figure 1. Duplicates or unnecessary gates are shown by dashed lines. A network of alternating OR - AND gates can be converted directly to a NAND - NAND network by inverting the literals on odd levels, with the level nearest the output as one. This brings us directly to Figure 2.

There is still one problem. There are two gates which have three inputs and I only keep two-input NAND gates and inverters as spares. A three-input NAND can be replaced by 2 two-input NANDS and an inverter (A NAND B NAND C) = ((A NAND B) NAND C). Looking at the two offending gates, we see that they share A NANDC′ in their equations, so we can share a gate.

The final circuit is shown in Figure 3. It can be realized with two quad NANDs and one hex inverter. This process could have been performed by entering the terms for which a zero value was desired (and don't cares) resulting in a network of NOR gates. Basic gates nearly always take more wiring in a circuit, but when purchased in quantity they are cheap, and they can make the difference between finishing a project today or just waiting for parts.

```
5 REM BOOLEAN EQUATION REDUCER
10 REM ALAN K. CHRISTENSEN
15 REM AUSTIN,TEXAS 4-14-79
20 DIM A$(250),A(250)
25 DEFFNA(I)=ASC(MID$(A$(I),M,1))
30 DEFFNB(I)=ASC(MID$(B$(I),M,1))
35 POKE 59468,14
100 REM -DATA INPUT-
105 B=-1:N=-1:N2=-1:I=0:J=0
110 INPUT N$
115 IF N$="X" THEN B=N2:GOTO 110
120 IF N$="END" THEN 130
125 GOSUB 550:GOTO 110
130 IF B<0 THEN B=N2
135 DIM B$(B),Q(B)
140 FOR I=0TOB:B$(I)=A$(I):NEXT I
145 L=LEN(A$(0)):N=N2
400 REM -REDUCE TO MINIMUM LITERALS-
405 L2=0:N2=N
410 FOR I=0TON-1
415 FOR J=I+1 TO N
420 GOSUB 500
425 IF D=1 THEN A(I)=1:A(J)=1:L2=1:GOSUB 550
430 NEXT J
435 NEXT I
440 GOSUB 600
445 IF L2<>0 THEN 400
450 REM -ELIMINATE REDUNDANT TERMS-
455 N3=N2
460 GOSUB 900
465 PRINTN$
470 IF B>=0 THEN 450
475 IF W=1THEN PRINT"MULTIPLE SOLUTIONS"
480 STOP

READY.
```

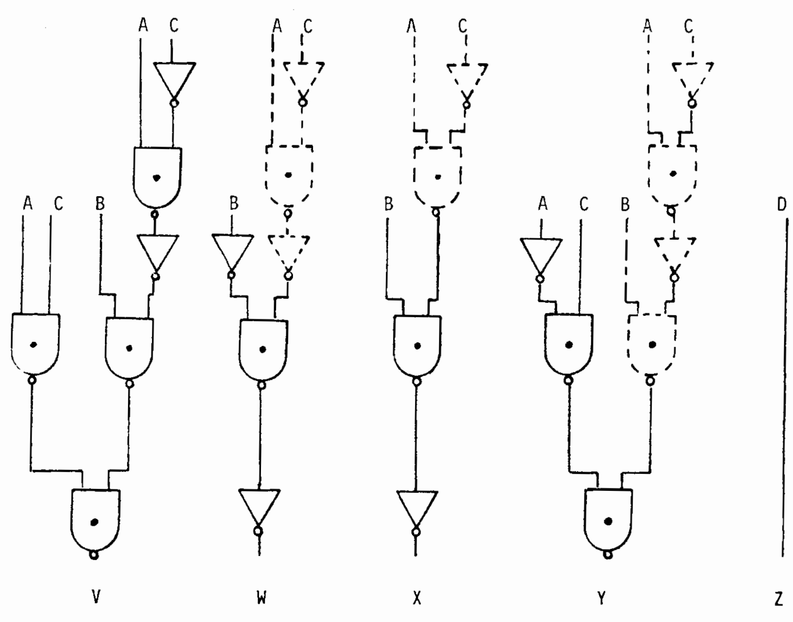


Figure 3

Screen Dump to Printer for the APPLE II

No need to print yards of listing when you want only one or two screenfulls of data. Print only the display segments you select with this versatile BASIC language output routine.

R. M. Mottola
Cyborg Corporation
342 Western Avenue
Boston, MA 02135

In certain programs it is often desirable to be able to print a screenfull of information on your printer after you have reviewed it on the screen. Long lists of data could be reviewed, one screenfull at a time, and only those pages that were needed would be printed.

The following short routine is a BASIC version of a machine language printer driver. Its advantages are that it will work with the Apple Parallel Printer Interface Card and any printer, without the need to re-write the printer driver. Also, since it is written in BASIC, it is easy to understand and to modify.

The first step required is to put a short machine language routine into memory. Lines 90 to 130 of the sample program POKE a routine into the free memory area starting at location $300. For systems using Apple DOS, it is important that you perform this step after DOS is booted, because this area of memory is clobbered during boot. This routine will make a character available to the character output routine in the monitor, $FDED, which will in turn pass it to the appropriate printer driver.

The second step is to add the screen printer subroutine to your BASIC program. This subroutine is shown in lines 500 to 610 of the sample program. Starting at the "home" position on the screen, this subroutine passes each character in screen memory (page 1) to the printer card, via the COUT routine in the monitor.

The POKE in line 560 passes the character to the machine language routine at $300. Although it may seem like a lot of "passing", this method allowes the use of a conventional PR#X command from BASIC to specify which slot is to receive the output. Other commands of note are those in lines 520 and 590. The first tells the parallel printer interface to print only on the printer, and not on the screen. The second returns output to both the printer and the screen.

The third step in implementing the screen printer is to add an INPUT statement to your program which asks the user if the screen is to be printed. This is found in line 250. Also note the POKE 34, 23 in line 240. This command sets the top of the scrolling window to line number 23, the bottom line of the screen, thus insuring that the prompt itself does not get printed.

The sample program listed is a demonstration program designed to show the screen printer in use. The routines in it can be adapted to any BASIC program with little dificulty. One thing to keep in mind, though, is that flashing or inverse characters may print out in various different ways, depending on the printer.

If you want to include flashing or inverse characters on the screen, the addition, noted in lines 552 to 560, listed after the demonstration program, should be included. These lines test for and "normalize" blinking or inverse characters so they will appear normally on the printer. However, using this modification will slow down the screen printer routine considerably. Its BASIC implementation is pretty slow to begin with. Replacing all constants with variables will make either version much faster.

See AppleSoft II BASIC Programming Reference Manual, Appendix E, for more on this. If you are using Apple DOS, remember to replace all PR#X commands with print control D; "PR#X" to keep DOS from being turned off. Finally, if you are using Integer BASIC, please note that you will have to modify the logic structure found in line 554. For a complete map of how the various characters are stored in screen memory, see "An Apple II Page 1 Map" by M.R. Connolly Jr., MICRO 8:41. Happy screen printing!

```
]LIST
0  REM  DEMONSTRATION PROGRAM
10  REM  SCREEN PRINTER ROUTINE
20  REM  FOR APPLE II APPLESOFT B
     ASIC
30 :
40  REM  DEFINE VARIABLES
50 SLOT = 1
60 OFFSCREEN$ = "": REM  "<CTRL>I4
     0N"
70 RETSCREEN$ = "                    ":
      REM  "<CTRL>II"
80 :
90  REM  PUT MACHINE LANGUAGE ROUT
     INE INTO MEMORY
100  FOR N = 768 TO 774
110  READ X: POKE N,X
120  NEXT
130  DATA  173,11,3,32,237,253,96
140 :
150  REM  FILL SCREEN FOR DEMONSTR
     ATION
160  FOR X = 1 TO 3
170  HOME : READ TXT$
180  FOR Y = 1 TO 22
190  FOR Z = 1 TO 6
200  PRINT TXT$;
210  NEXT Z
220  PRINT "": REM  NULL STRING
230  NEXT Y
240  POKE 34,23
250  PRINT : INPUT "PRINT SCREEN?
     (Y/N) ";ANS$
260 : IF ANS$ = "Y" THEN  GOSUB 50
     0
270  POKE 34,0
280  NEXT X
290  DATA  " MICRO"," APPLE"," 650
     2 "
300  END
400 :
450 :
500  REM  SCREEN PRINTER SUBROUTIN
     E
510  PR# SLOT
520  PRINT OFFSCREEN$
530  FOR A = 0 TO 80 STEP 40
540  FOR B = 0 TO 7
550  FOR C = 1024 + A TO 1063 + A
560  POKE 779, PEEK (C + B * 128)
570  CALL 768
580  NEXT : NEXT : NEXT
590  PRINT RETSCREEN$
600  PR# 0
610  RETURN
PR#0
]LIST 552,560
552 CHAR =  PEEK (C + B * 128)
554  IF CHAR < 192 THEN CHAR = CH
     AR + 64: GOTO 554
556  IF CHAR = 224 THEN CHAR = 16
     0
560  POKE 779, PEEK (C + B * 128)
```

Classified Ads

FREE SAMPLE COPY - New BUY-SELL-TRADE Publication, indexed by product type, mailed monthly First Class. 3500 circ., low, low ad rates. Send for your free sample copy now! Subscription $3.50 per year, FREE THREE LINE AD with subscription. Order from:

LECTRONICS EMPORIUM
Dick Costello, Pub.
P.O. Box 828
Derry, NH 03038

The TARGET for users of Rockwell's AIM 65. Find out how to use the printer, keyboard and display. Reviews of upcoming Assembler and BASIC in ROMs. Six bimonthly issues for $5.00 US and Canada ($12.00 elsewhere). Order from:

Don Clem
RR#2
Spencerville, OH 45887

ZIPTAPE loads 8K BASIC in 15 seconds! Slower than a speeding disc? Sure, but it only costs $22.50 plus $1.00 S&H. $3.00 extra for software on KIM cassette. Described in MICRO #6. SASE for info. Order from:

Lew Edwards
1451 Hamilton Ave.
Trenton, NJ 08629

SYM-1 complete with 2 K RAM, Synertek tech notes, other frills, $ 200.00 (have two, first order gets First Book of KIM and KIM cassette thrown in). KIM power supply, $ 25.00.
Tel. (415) 933-1123.

MICRO

OSI Memory Test in BASIC

William L. Taylor
246 Flora Road
Leavittsburg, OH 44430

All memory tests are not alike. This one features an extensible, BASIC language implementation.

Have you experienced the complete failure of your favorite program lately? Have you reloaded it into the machine only to have it bomb over and over again? Well, I have, and many times! This could be caused by a bug in the program, but if the program has run before and now bombs there must be something wrong in the hardware. This usually means that there is a reclusive bug hidden somewhere in those many K's of RAM.

How do you find this reclusive bug? If you have a machine code monitor and loader, you could load the memory and step through the program checking for errors. You might also load a diagnostic program to test the memory. "OK" you say, "but I don't have a machine code monitor. My machine has only BASIC in ROM. What do I do to check for these bugs in my machine? I have no means to get at these bugs in my machine with this BASIC only!"

Well take heart, all is not lost. I have had this same experience. Felt the same wrath, of the same bug in those many K's of RAM, that you are feeling now! From this experience I made a decision. I decided to prevent this from doing me in over and over again. My solution to the bug-in-memory caper was to write a diagnostic program, in BASIC, to check the memory of the BASIC-in-ROM only machine.

The program that I have written will load memory with an inital value stored in the D variable, between the address limits P1 and P2. The program increments the D variable from its initial value to 255 decimal. This represents all combinations of bits that can be stored in a memory location. After the bits are stored, the program compares the data bits in memory to the initial value that was stored there and, if they are not the same, a report will be printed out to the terminal.

I have written the program to request page numbers for the starting and ending addresses. This could be changed to use decimal equivalents if the reader wishes. The starting address is contained in variable P1 at line 700. The ending address is contained in P2 at line 710. The contents of both variables are multiplied by 256 to obtain the decimal equivalent of the page numbers. Line 720 is the inital value of the data and is usually set to 0.

At line 750 the program is told to load the limits of memory between P1 and P2 via a FOR-NEXT loop. At line 760 the data bits are POKEd into memory. Line 785 looks at the data in the memory location that was previously stored. At line 790 I compare the data stored in memory against the data in variable D to see if the two are equal. The next byte is loaded and compared at line 800.

Line 825 increments the data value in the D variable. Line 830 checks the D variable to see if 255 decimal has been reached and, if not, executes a return loop through the program. Line 840 reports the results of the memory test.

This program was written in MicroSoft BASIC for the OSI Challenger. It should run under other BASICs with minor modifications. The program will be of interest to users of machines with BASIC in ROM and others who want a simple way to test memory. The program is some what slow, but this a very small price to pay for the ease of operation. Good luck and good memory testing.

```
650 REM MEMORY TEST  BY W.L. TAYLOR 1/2/79
660 PRINT " ******MEMORY TEST****** ":PRINT
665 PRINT " ENTER STARTING PAGE AND ENDING PAGE":PRINT
700 INPUT " STARTING PAGE ";P1
710 INPUT " ENDING PAGE   ";P2
720 D=0
730 LET A=P1*256
740 LET B=P2*256
750 FOR C= A TO B
760 POKE C,D
770 E=PEEK (C)
780 IF E<>D THEN PRINT " BAD DATA BYTE AT";C
790 IF E<>D THEN END
800 NEXT C
810 D=D+1
820 IF D<256 THEN 750
830 IF D=256 THEN PRINT " TEST COMPLETE WITH NO BAD DATA BITS
    DETECTED":PRINT
840 END
```

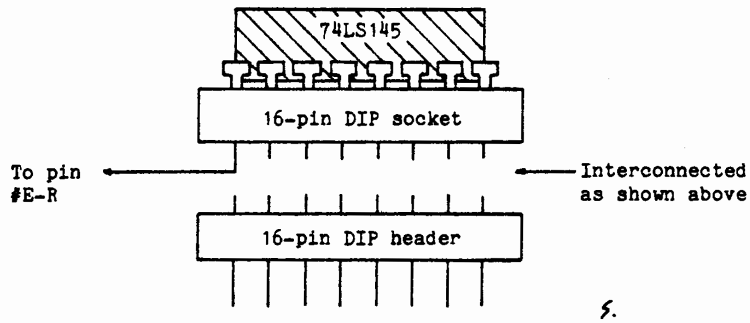
SYM and AIM Memory Expansion

An easy hardware modification addresses extended memory in contiguous 8K blocks with no gaps. This neat enhancement makes Memory Plus a natural for RAM-ming more data into the SYM and AIM.

Paul Smola
Acushnet Corporation
P. O. Box E916
New Bedford, MA 02742

In an attempt to implement BASIC on the SYM it became apparent that the 4K of onboard RAM was insufficient for our needs. Although we have several Memory Plus boards around, the RAM on these boards is addressable in 8K byte blocks decoded at 8K boundaries, beginning at location 2000. Unfortunately, this decoding scheme leaves a 4K block of memory unimplemented. That block of memory is from address 1000 through 1FFF.

In order to overcome this shortcoming, it is desirable to decode the Memory Plus board in 8K blocks that are addressable at 4K boundaries; that is, at locations 1000, 3000, 5000, etc. With this scheme several MP boards could be added on to expand the SYM memory in a continuous fashion. There are methods available for making this change, but most of these require changes on the MP board itself. This is undesirable, especially if servicing becomes a problem.
The solution lies in replacing the three high order address line decoding schemes with one that will address memory at 4K boundaries. This can be accomplished by bringing addresses A12, A13, and A14 into the inputs of the 74LS138, as opposed to the present A13, A14, and A15. With this change any position of the rotary switch which selects the RAM decoding address enables the RAM at 4K boundaries, and also only in 4K blocks.



Remove the 74LS138 from socket U4 on the MP board and replace it with the above assembly

Figure 2

If we were to OR two adjacent outputs together, we would have 4K boundaries with 8K blocks. However, because the outputs of a 74LS138 are totem-pole, ORing them must be done with additional gating and not simply by tying the outputs together, as is done with open-collector outputs.

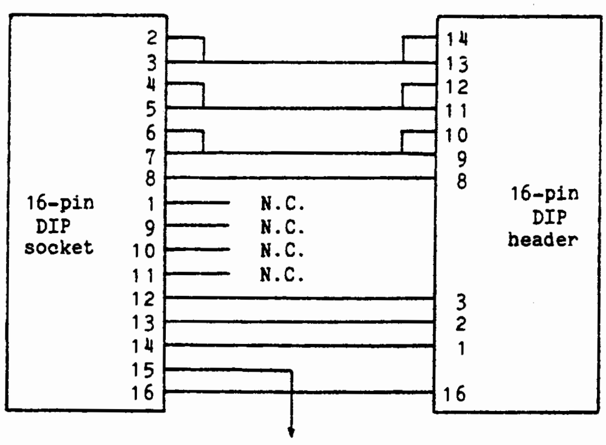
One method of doing this is by replacing the '138 with a 74LS145 BCD-to-decimal decoder driver. This device has open collector outputs enabling them to be wire OR'ed together. However, the pin out on the '145 is radically different from that on the '138.

The way to get around this is to mount the '145 in a 16 pin dip socket which is in turn connected to a 16 pin dip header. However, rather than matching the pins number for number, the connection diagram in Figure 1 is followed. This is most easily accomplished by using a three level wire-wrap socket and cutting short all the pins except 8 and 16. These shortened pins are then wired to the correct position on the header by soldering jumpers on. This causes the pin out connections to be changed and thus allows the '145 to operate in the socket which was previously loaded with the '138.

The 16 pin dip header is then loaded into the MP board into socket U4 as shown in Figure 2. The '145 has the advantage of having four address input lines. Thus address lines A12, A13, A14, and A15 are brought into it and fully decoded. Since address line A12 is not brought to socket U4, it must be separately wired. A convenient place to make this connection is on the MP expansion connector pin #E-R.

With these changes, the RAM select rotary switch now selects hex locations 1000-2FFF at the first two positions. At the second two positions RAM is selected at 3000-4FFF. In the third two positions RAM is selected at locations 5000-6FFF. RAM will not be selected with the selector switch in the seventh position.

With the switch in the first or second position, BASIC on the SYM can be implemented with 12K memory; the 4K onboard, plus the 8K from the MP. The addition of another MP board set up the same way with the RAM selection switch in either position 3 or 4 would yield a system with 20K of continuous memory.



Solder to pin #E-R on the Memory Plus Expansion Connector

Figure 1

6502 Based SYSTEMS

The COMPUTERIST offers the best in the single-board, 6502-based microcomputers. These include the Rockwell **AIM-65**, Synertek Systems **SYM-1**, Commodore **KIM-1**, and, late this fall, The COMPUTERIST **MICRO PLUS**. As you will see from this catalog,, The COMPUTERIST is devoted to supporting this class of 6502 systems. Think of us first - for all of your 6502 needs: Systems, Expansion, Power, Software, and other items.

The **AIM 65** is a complete microcomputer system, not just a single board computer. It has many of the features of the KIM-1 and SYM-1, but also has three alphanumeric type devices which make it significantly different:

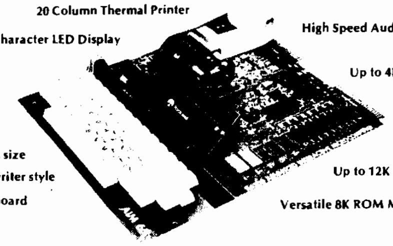
- **Full size typewriter style keyboard** - makes it easy to enter data.
- **Twenty character LED display** with sixteen segment displays for good looking, easy-to-read alphabetic and numeric characters.
- **Twenty column thermal printer** for alphanumeric hardcopy.

Other features include:

- An **8K ROM Monitor** with a **mini-assembler/disassembler, editor,** numerous operator functions and many important subroutines for program development.
- Comes with 1K RAM expandable on-board to 4K.
- Has provision for an additional **12K of ROM** including a **4K Assembler** and an **8K BASIC.**
- The expansion and application pin-outs are compatible with the KIM and SYM, making it simple to interface to existing devices.
- Supports KIM format cassette tapes at 1 and 3 times normal speed, plus its own high speed cassette I/O. Includes two complete cassette ports with remote control facilities.

AIM 65 by Rockwell International

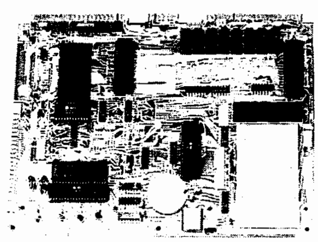
The Complete 6502 System



AIM 65: \$375[00] 1K RAM - \$420[00] 4K RAM

SYM-1 by Synertek Systems

An Expandable 6502 System



SYM-1: \$270[00] 1K RAM - 315[00] 4K RAM

The **SYM-1** is a relatively new entry into the 6502 market by Synertek Systems. The board is the same size and shape as the KIM-1 and uses the same connector placement and pin-outs, thereby maintaining a fair degree of compatibility with the KIM-1. Its main advantages are:

- It comes with **1K of user RAM,** and is expandable on-board to **4K RAM.**
- **A larger Monitor - 4K** vs the KIM 2K - with a number of useful functions.
- It has room on-board for **an additional 12K ROM.** This ROM may be programs and data defined by the user or Synertek supplied programs such as an Assembler or BASIC.
- It has much more **I/O capability** than the KIM-1 and improved timers.
- It has KIM compatible tape format as well as a **higher speed tape format.**
- Like the KIM, it supports a teletype terminal, but it also supports more sophisticated terminal interfaces.
- The touch-pad type of entry keypad is more reliable than the type used on the KIM.

If you need the added features of the SYM-1, especially the extra RAM and ROM provision, then this is a best buy. It currently has limited supporting software, being new to the market, but this should not be a long term problem.

The **KIM-1** is the grand-daddy of all 6502 based microcomputer systems. It was orignally created by MOS Technology, the inventors of the 6502, as a way to demonstrate the power of the 6502 to the industrial community. To their surprise, the KIM-1 became a highly successful single board computer - used in industrial control, education, hobby, and many other applications. It is still very popular today. Features of the KIM-1 are:

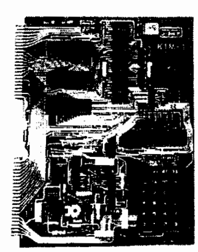
- Based on the 6502 microprocessor with its powerful instruction set.
- **Two 6530 multi-purpose** chips each containing 1K ROM, 64 bytes RAM, a programmable timer and 15 I/O lines.
- **1K bytes of RAM,** a **Hex Keypad** for entering programs and data, and a six character **LED display.**
- It supports a **20mA Current Loop TTY** and **Audio Cassettes** for program/data storage.

The very low price makes this an excellent buy - and the expansion bus structure is compatible with the AIM 65 and SYM-1 so that conversion to one of these other systems can be made with minimal hardware difficulty. There exists a large body of literature and many "ready-to-run" programs for the KIM-1.

KIM-1 by Commodore

The Original 6502 System

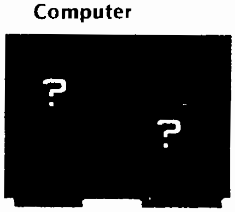
20 mA Current Loop TTY Interface
Audio Cassette Interface
15 User I/O lines
2 Interval Timers
1K+ RAM
2K KIM Monitor ROM
Hex Keypad/LED Display



KIM-1: \$180[00]

MICRO PLUStm

The Super 6502 Single Board Computer



Features:
- FLOPPY DISK CONTROLLER
- Up to 16K PROM/EPROM on-board
- KIM Compatible CASSETTE I/O
- 20MA Current Loop TTY Interface
- Lots of 6522 type I/O
- Up to 16K RAM on-board
- Planned for Late 1979

MICRO PLUStm is currently in the advanced design stages.
It will be a single board microcomputer featuring:

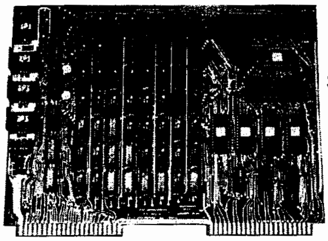
- **6502 Microprocessor**
- **Floppy Disk Controller** for Mini and Regular Floppy Disks
- **Cassette I/O** including KIM compatability
- **20 MA Current Loop TTY Interface**
- Up to **16K RAM** on-board
- Up to **16K ROM/EPROM** on-board
- Several **6522 VIAs**
- Same **SIZE** and **SHAPE** and **PIN-OUTS** as **KIM-1/SYM-1**

Plus a couple of proprietary features to be announced later. Scheduled for initial delivery late 1979. Please **do not** call or write for additional info until September 1979.

SYSTEM EXPANSION

The COMPUTERIST makes it easy for you to expand your **KIM-1, SYM-1** or **AIM 65** based system. Four boards are offered to: increase the memory of your system, add full feature video to your system, provide a means to add your own circuits, and a means to get all of these added features working together. The design of these boards makes it possible for you to choose one vendor for all your normal system expansion requirements. The four boards are designed to work together and fit together in a system configuration which makes sense. The **PLUS** on each board represents added features that are not found on similar boards offered by other manufacturers - **PLUSES** that often dramatically enhance the capabilities of your basic system.

MEMORY PLUS™ FOR AIM/SYM/KIM



8K STATIC RAM LOW POWER

Sockets for 8K Eprom

6522 I/O Port

ON BOARD REGULATORS

EPROM PROGRAMMER

MEMORY PLUS: $200.00 FULLY ASSEMBLED AND TESTED

EXPAND YOUR SYSTEM WITH MEMORY PLUS™

MEMORY PLUS combines four of the most important system expansion capabilities on one PC board. This board uses the **standard KIM-4 Expansion Bus** and is the same size/shape as the KIM-1/SYM-1 so it can be conveniently placed under any AIM/SYM/KIM system. The four functions are:

- **8K RAM** - with low power 2102 static RAM - the most important addition for most systems.
- **8K EPROM** - sockets and address decoding for up to 8K of Intel 2716 type EPROM.
- **EPROM Programmer** - program your EPROMS on the board! **I/O - 6522 Versatile Interface** provides two 8 bit I/O ports, two multi-mode timers, and a serial/parallel shift register.

Other features of Memory Plus include:

- On-board voltage regulators for +5V for general power and +25V for the EPROM Programmer.
- Independent switch selection of the RAM and ROM starting addresses.
- All IC's socketed for easy field replacement.
- Fully assembled and burned in - ready to plug in and go.
- Documentation includes a 60+ page manual with schematics, program listings, 2716 and 6522 data sheets, and a cassette tape with an EPROM Programming Program and a Memory Test.
- Over **800 MEMORY PLUS** units are already in use with **AIMs, SYMs** and **KIMs**.
- May be directly connected to your system with our cable or through our MOTHER PLUS™ board.

IT'S EASY TO ADD VIDEO PLUS™ TO YOUR SYSTEM.

VIDEO PLUS is the most powerful expansion board ever offered for 6502 based systems. It has many important video features including:

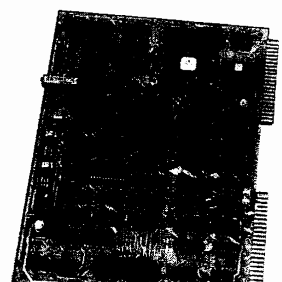
- **Programmable Display Format** - up to 100 characters by 30 lines on a good monitor.
- A ROM Character Generator with **UPPER and lower case ASCII** characters.
- A **Programmable Character Generator** for up to 128 user defined characters which may be changed under program control. You can define **graphics, music symbols, chess pieces, foreign characters, gray scale** - and change them at will!
- May be used with an inexpensive TV set or an expensive monitor.
- Up to **4K of Display RAM,** with **Hardware scrolling, programmable cursor,** and more.

In addition to the video features, VIDEO PLUS also has:

- A **Keyboard Interface** which will work with any "reasonable" keyboard.
- A built-in **Light Pen Interface.**
- Provision for a **2K EPROM** or ROM for video control or other software.
- All of the memory - 6K RAM and 2K EPROM can be used as system memory whenever it is not in use as display or programmable character generator.
- VIDEO PLUS may be used directly as an expansion of an AIM/SYM/KIM system, or has provision for the addition of a 6502 for use as a **Stand-Alone** system or Terminal!
- Only requires +5V and has on board voltage regulators. Since it's the same size/shape as the KIM or SYM, it may easily be placed under an AIM/SYM/KIM system. It uses the KIM-4 expansion format.

Fully assembled, tested and burned in. Connect directly to your system or via the MOTHER PLUS board.

VIDEO PLUS™ FOR AIM/SYM/KIM



UPPER/lower case ASCII

128 Additional User Programmable Characters: GRAPHICS-SYMBOLS-FOREIGN CHARACTERS

Programmable Screen Format up to 80 CHARACTERS - 24 LINES

KEYBOARD and LIGHT PEN Interfaces

Up to 4K DISPLAY RAM

Provision for 2K EPROM

Provision to add 6502 for STAND-ALONE SYSTEM

ASSEMBLED AND TESTED WITH 2K DISPLAY RAM

VIDEO PLUS: $245.00

PROTO PLUS™ FOR AIM/SYM/KIM

Same SIZE and SHAPE as KIM/SYM

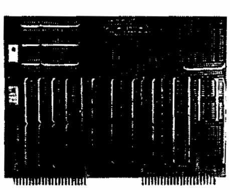
Professional Quality

Double Sided, Plated through Holes

Two Sets of GOLD Plated Dual 22. Fingers

Designed for WIRE WRAP or SOLDER Connections

Provisions for 40 14/16 pin sockets
4 24/40 pin sockets
3 voltage regulators



PROTO PLUS: $40.00

ADD YOUR OWN CIRCUITS WITH PROTO PLUS™

PROTO PLUS is the simple way to add special circuits to your system. It is the same size and shape as the KIM and SYM, making it extremely easy to use with these systems, and can be neatly added to the AIM as well. It provides about 80 square inches of work area. This area has provision for about **40** 14/16 pin sockets, about **4** 24/40 pin sockets, 3 regulators, etc. The connections to the board are made through two sets of gold plated fingers - exactly like the AIM/SYM/KIM. This means that there are a total of 88 edge connections - more than enough for most applications. This is a professional quality, double sided board with plated through holes. The layout was designed so that you can use wire wrap sockets or solder sockets - each IC pad comes out to multiple pads. There is room for voltage regulators and a number of other "non-standard" devices. The PROTO PLUS will plug directly into the MOTHER PLUS making for a handy package.

PUT IT ALL TOGETHER WITH MOTHER PLUS™.

MOTHER PLUS provides the simpliest way to control and package your expanded system. MOTHER PLUS does three major things: 1 - provides a method of interconnecting the individual boards (**MEMORY PLUS, VIDEO PLUS, PROTO PLUS**); 2 - provides buffering for the address, data and control signals; and, 3 - acts as a traffic cop for determining which addresses are reserved for the processor and which for the expansion boards. It supports the **standard KIM-4 Expansion Bus,** so it is electrically compatible with a large number of expansion boards. It is structured so that the processor board fits into the top slots with the expansion boards mounting below. This permits a system to be neatly packaged - it doesn't have its guts hanging out all over a table top. Provision is also made for application connections through solder eyelet connectors. Specifically designed to work with **AIM/SYM/KIM** systems. Other features are: a **terminal** for bringing power into your system; phono jacks for the **Audio In/Audio Out;** phono jacks for connecting a TTY device; provision for a **TTY/HEX switch** for the KIM; a **16 pin I/O socket** for accessing the host Port A/Port B; plus two undedicated 16 pin sockets which may be used to add inverters, buffers, or whatever to your system.

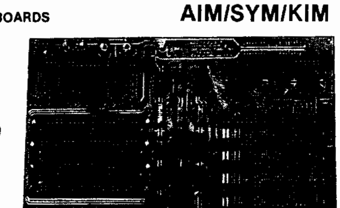
MOTHER PLUS™ FOR AIM/SYM/KIM

ADD UP TO FIVE ADDITIONAL BOARDS

AUDIO/TTY CONNECTIONS

POWER TERMINALS

APPLICATION CONNECTORS



FULLY BUFFERED

FULLY DECODED

KIM-4 Bus Structure

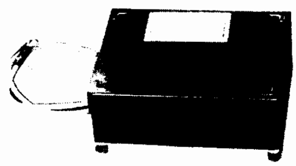
MOTHER PLUS: $80.00 FULLY ASSEMBLED AND TESTED

POWER SUPPLIES

The COMPUTERIST offers a variety of power supplies to meet the varied requirements of 6502 based systems.

POWER PLUS™

ALL THE POWER A KIM-1/SYM-1 NEEDS



Neat, Compact, Economical

Thousands in Use

INPUT: 115V/60Hz

OUTPUTS: Regulated +5V at 1.4A
+12V at 1.0A
Unregulated +8V up to 4.3A
+16V up to 1.0A

Will Power a KIM-1/SYM-1 and one Additional Board Such as MEMORY PLUS or VIDEO PLUS

POWER PLUS: $40.00

We offered the first power supply built specifically for the KIM-1 and since May 1977 have delivered over a thousand units. This unit - POWER PLUS - is a simple model. It does not even have an On/Off switch or Pilot Light, but does provide the power for a KIM-1 or SYM-1 with enough to spare for an additional MEMORY PLUS or VIDEO PLUS board. For the small home system, the electronics lab, the class room, etc., where the system is not going to be greatly expanded, this is an ideal unit, and is priced very low.

For more advanced systems or more demanding environments we offer three heavy duty supplies. Each of these comes in an all metal case; includes an On/Off Switch and Pilot Light; may be run on 115V/60Hz or 230V/50Hz AC power; has a grounded three-wire power cord; and has a screw-type terminal strip for each connection.

A special supply is available for the basic AIM 65 system. This is a small, open-frame unit which may be placed inside the standard AIM Enclosure. It provides enough power for the AIM 65 including printer and one additional board.

POWER PLUS™

5 SUPER 5 5/24

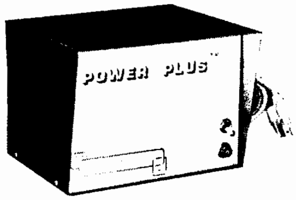
All Include the Following Features:

ALL METAL HEAVY DUTY CASE

ON/OFF SWITCH and PILOT LIGHT

115/60Hz or 230/50Hz INPUT

GROUNDED THREE-WIRE POWER CORD



POWER PLUS 5: +5V at 5A, ±12V at 1A $75.00
POWER PLUS SUPER 5: +5V at 10A, ±12V at 1A $95.00
POWER PLUS 5/24: +5V at 5A, +24 at 2.5A, ±12V at 1A $95.00

POWER A PLUS™

SPECIFICALLY DESIGNED FOR THE AIM 65

Small Enough to Fit Inside the AIM Enclosure

Enough Power for the AIM 65 Fully Loaded

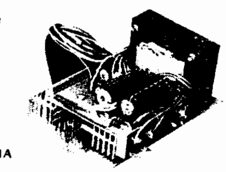
Plus an Additional Board

Works on 115V/60Hz or 230V/50Hz

Provides Regulated +5V at 5A and +24V at 1A

Grounded Three-Wire Power Cord

ON/OFF Switch and Pilot Light

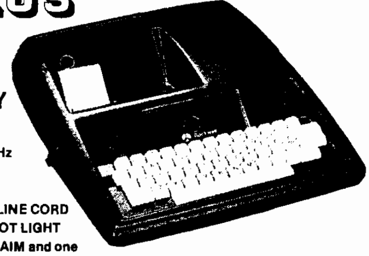


POWER A PLUS: $50.00

ENCLOSURES AND CASSETTE RECORDERS

AIM PLUS™

ENCLOSURE
WITH BUILT IN
POWER SUPPLY

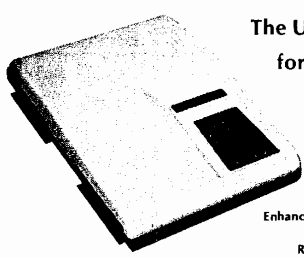


SPECIFICATIONS:
INPUT: 110/220 VAC 50/60 Hz
OUTPUT: +5V @ 5A
+24V @ 1A
GROUNDED THREE-WIRE LINE CORD
ON/OFF SWITCH WITH PILOT LIGHT
Enclosure has room for the AIM and one additional board: MEMORY PLUS or VIDEO PLUS

AIM PLUS: $100.00 AIM and AIM PLUS: $475.00

ENCLOSURE PLUS™

The Ultimate Enclosure for the KIM-1



Protects Your KIM-1

Neat, Attractive, Professional

Full Access to the Expansion and Application Connectors

Enhances the LED Display with a Red Lense

Room for the KIM-1 and One Additional Board such as MEMORY PLUS or VIDEO PLUS.

ENCLOSURE PLUS for KIM: $30.00

The SUPERSCOPE(R) C-190 Cassette Tape Recorder by Marantz is a very high quality audio tape recorder which has a number of features which make it particularly well suited to use with microcomputers.

Runs on 110V AC or 6V DC from a power pack or batteries. Has Tone Control and separate Volume Controls for Recording and Playback.

Has VU Meter for recording level, and has three recording modes: Automatic Record Level, Limiter or Manual. Has Tape Speed Control - Adjusts ±20%. This is especially useful when using tapes recorded on other recorders.

Tape Counter - 000 to 999.

Electronics remain ON when recording is being held OFF in Route.

An excellent unit which has been recommended by several of the microcomputer manufacturers.

CASSETTE C-190

SUPERSCOPE C-190 by Marantz

A High Quality Cassette Recorder with all of the Features Required for Microcomputer Systems:

VU Meter Displays Recording Level

110V AC or 6VDC or Battery Operation

Tape Location Counter

Three Recording Methods

Variable Speed Control: ±20%

Remote Control Leaves Electronics ON

SUPERSCOPE C-190: $90.00

SOFTWARE and Other Good Stuff

To make any microcomputer system useful, you need software. The COMPUTERIST has software packages available for three systems. Each of these packages come with full User/Operator Instructions, a Cassette Tape, and, with the exception of MICRO-ADE, a complete set of Source Listings so that you can more fully understand, utilize, and modify the software.

PLEASEtm is a collection of games and demonstrations. It contains a dozen programs such as a 24 Hour Clock, a High/Low number guessing game, "Shooting Stars", a Drunk Test, an Adding Machine, and so forth. PLEASE is written in a "high level language" which permits the user to make simple modifications and create his own demonstrations. It will run on an unexpanded KIM-1, or on a SYM or AIM with 2K RAM. **$10.00**

MICROCHESStm is the original chess player for small systems. While it does have some limitations, it does play a reasonably good game of chess. It includes a number of "canned" openings and makes a good tutor for a beginner or a brush-up challenger for the more advanced player. Includes three levels of difficulty. It will run on an unexpanded KIM-1, or on a SYM or AIM with 2K RAM. **$15.00**

HELPtm **Mailing List** is a complete package for the maintenance and printing of mailing lists. It includes an Editor for entering and updating the mailing lists; a List Printer which outputs a single tabular format line per entry for analysis and updating; and a Label Printer which outputs to mailing labels. The List and Label functions include the capability of abstracting subsets of the total mailing list and of adding an extra line of information - such as "Subscription Expired" - to a subset of the mailing list. It requires program control of two cassettes and some form of printing terminal. It will run on an unexpanded KIM-1, or on a SYM or AIM with 2K RAM. **$10.00**

Bits and Bytes

While The COMPUTERIST does not, in general, sell IC's and other small pieces of hardware, there are a few useful devices which we use in large quantity in our own products and which we can offer at good prices to our customers.

2114 Low Power Static RAM (1K by 4 bits) Used to expand SYM, AIM, or VIDEO PLUS	2 for $15.00 (1K bytes)
2102 Low Power Static RAM (1K by 1 bit) Type used in KIM-1 and MEMORY PLUS	8 for $10.00 (1K bytes)
6522 VIA Versatile Interface Adapter Used in SYM, AIM, MEMORY PLUS and VIDEO PLUS	$7.50
Dual 22/44 Pin Connectors - Solder Tail or Solder Eyelet Three required for MEMORY PLUS or VIDEO PLUS	$2.00each
AIM/SYM/KIM to MEMORY PLUS/VIDEO PLUS CABLE	$15.00
HELP Relay Package - everything required to control two audio cassette recorders (except the PC board)	$10.00

HELPtm **Information Retrieval** is a package for creating and retrieving from a cassette based data base. The Editor portion permits the user to create files with up to six independent Data Fields plus a Flags Field which contains abstract data about the file. The Retrieval portion permits entries to be selected by the contents of any combination of Data Fields and/or by up to six independent tests on the Flags Field. The Flags Field tests include three "equal" tests, one each "not equals", "greater than" and "less than" test. The program is a good demonstration of the power of a small system. It will run on an unexpanded KIM-1, or on a SYM or AIM with 2K RAM. It also requires program control of two cassettes and some form of ASCII terminal. **$10.00**

MICRO-ADEtm is a complete **Assembler, Disassembler,** and **Editor** package. The Assember is a full scale version with six character labels, two-pass capabilities, and makes good use of the cassettes for assembling large programs. The Disassembler converts object code into user readable source code. If a symbol table is available for the code being disassembled, then a complete listing with labels may be obtained. The Editor can be used separately or in conjunction with the Assembler. It features Line Insert/Delete, can Move sections of lines, and uses the Cassettes for automatic control of large files. MICRO-ADE will run on a KIM, SYM or AIM with at least 8K RAM starting at address 2000. A version to run in 4K ROM plus 4K or more of RAM is included on the cassette tape. While MICRO-ADE can work entirely with RAM, it is most powerful when used in conjunction with two cassette recorders under computer control. Some type of ASCII terminal is required. MICRO-ADE comes with complete Operator Instructions and the Source Listing for the I/O portion of the code so that a user can adapt it to his own specific devices. Complete Source Listings may be purchased separately. **$25.00 each**

Shipping and Handling
United States

Total Order	Regular Items	Power Supply or Cassette Recorder
Up to $15.00	$1.00	
Up to $50.00	$2.00	$3.00
Over $50.00	$3.00	$5.00

Please provide Street Address for UPS.
Prepaid or COD unless credit has been established.
Mass. Residents add 5% sales tax or provide Tax Exempt Certificate.

Foreign

Add 10% of total, minimum $3.00
Overpayment in excess of $5.00 will be refunded.
All items AIR Parcel Post except Power Supplies which due to their weight must go surface.
All payments must be via International Money Order.

THE COMPUTERIST INC

PO Box 3 · So. Chelmsford, Mass 01824 · 617/256-3649

The AIM/SYM/KIM Leader

The First Book of KIM — on a SYM

Programs presented in The First Book of KIM can be modified to run on a SYM. What's more, the techniques presented here will aid in the conversion of other KIM software.

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

Anyone who purchased "The First Book of KIM" with the expectation of easily modifying the programs to run on their SYM quickly found that the KIM and SYM might be hardware compatable, but the monitors are a lot different. The SYM manual has a list of SYM counterparts to the KIM routines. It also makes the disclaimer that "the routines do not perform identically." This is an over simplification! Some of the SYM routines are really only distant cousins to their KIM counterparts. The routines listed in the SYM manual are not close enough to the KIM routines to be easily substituted for the KIM entry points used in the book.

The first couple of programs I converted the hard way, with lots of relocating and some logic changes. I finally got smart and took the time to write these routines using simple address substitutions. These routines are obviously not identical to the KIM versions they replace, and definitely do not take the same number of execution cycles.

You may have to "tweek" some of the delay loop counters in the programs. Otherwise, replace the KIM addresses with these, fix up the I/O addresses (which I will also discuss later) and about 90% of your conversion is done, at least for the games.

I have not bothered to try any of the cassette programs yet. I have enough problems with the SYM standard routines. There will be some places where you may need to get a little fancy to do the conversion without relocating things. Just remember that if you can perform an equivalent function in fewer bytes you can use NOP's to avoid relocation.

Before I get down to discussing the routines and some notes about writing directly to the displays, I would like to mention that these routines require one hardware modification to the SYM board in order to work properly. The modification is to remove the jumper that enables system RAM write protect, jumper MM-45, just to the left of the crystal.

This is the first modification I made to my SYM, and I have not regretted it at all. If you are leary about permanently disabling something, as I was, you will find that a four position DIP switch does nicely. You will get the added advantage of being able to write protect user RAM. The alternative is to insert a JSR ACCESS at the start of each routine.

The first routine is the one to light the on-board displays, and actually has two entry points. If you enter at SCAND, the byte indirectly pointed to by POINTL is moved to INH, and then the program falls through to SCANDS. This routine lights the display with the six hex values corresponding to the three bytes POINTH, POINTL, and INH, and then returns.

The SYM "equivalent" standard routines OUTBYT and SCAND are not suitable replacements. OUTBYT takes the bytes in the A register, converts them to two hex digits, and rolls them into the display from the right. Repeated calls to OUTBYT cause the characters to march from right to left across the display.

SCAND, on the other hand, lights the display with six hex digits as we want, but it assumes that the segment codes are already in the display buffer. This is further complicated by the fact that the display buffer is at $A640, which is a two byte address instead of the single byte used by the KIM.

What I did was to pick up the data from the KIM addresses, convert it into segment codes by using each nibble as an index into the SYM segment code table, and store all six bytes of segment code in the display buffer before calling the SYM SCAND routine to light the display. Fortunately, the KIM addresses do not conflict with important SYM addresses. Specifically, $FA and $FB are used by SYM as the pointer to RAM for the EXECUTE command, and $F9 is used as a work area for the terminal I/O routines.

The SYM subroutine GETKEY superficially resembles the KIM routine of the same name. The SYM does a lot more for you, since it lights the display and waits for the key to be pressed. It also debounces the keyboard, and converts the key code to ASCII. The KIM routine, on the other hand, reads the keyboard and returns with a binary number corresponding to the key pressed. It does not wait to debounce the keyboard, nor does it light the display. This makes it easier to program the keyboard independently of the display. It is also more work, by the way.

The SYM routine LRNKEY is a closer approximation to the routine we want. It scans the keyboard once, converts the key code to ASCII, and returns. Conveniently, the value in the X register is the index that was used to get the ASCII equivalent of the key pressed. This table starts with the code for ZERO, so the value in X is neatly set 0 through F for those keys, and all we need to do is transfer it to the A register.

The SYM has more keys than the KIM, so these are set to the KIM value for "no key" on the assumption that the KIM routines wouldn't know what to do with them anyway. For the remaining keys we just use a translate table that is somewhat arbitrary since the keys are not labeled identically. See the program listing for which keys are translated to what, and note that the SYM shift key is made equivalent to the KIM "no key" value.

The KIM routine KEYIN has a very close equivalent in the SYM entry KEYQ. The main difference between them is which way the zero flag gets set if a key is down. The KIM returns a zero condition if a key is down, and the SYM returns as not zero. All this routine does is load a $FF or $00 into the X register to reverse the SYM zero flag setting.

The reason the X register is loaded with $FF for a "no key" is that LRNKEY in the SYM monitor does an INX immediately before returning if entered without a key down. With X set to $FF upon entry, this will result in a zero condition from the LRNKEY routine. Since none of the ASCII codes are zero, we can set the appropiate key value in the GETKEY routine. This way a JSR KEYIN followed by a JSR GETKEY will be consistant with the KIM routines.

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

                      SYM-1 VERSIONS OF VARIOUS KIM ROUTINES
                      BY: NICK VRTIS - LSI/CCSD     04/12/79
                      MODIFIED BY MICRO STAFF       06/06/79

                      THE PURPOSE OF THESE ROUTINES IS TO PROVIDE A CERTAIN
                      AMOUNT OF SOFTWARE COMPATIBILITY BETWEEN THE SYM AND
                      KIM MONITORS.  THIS WILL MAKE IT EASIER TO CONVERT
                      PROGRAMS WRITTEN FOR THE KIM TO RUN ON THE SYM.

                      TIME DEPENDENT CODE IS NOT SIMULATED

                      NO ATTEMPT IS MADE TO DUPLICATE THE KIM MONITOR,
                      ENTRY POINT FOR ENTRY POINT.  RATHER, THESE ARE
                      THE MAIN ROUTINES AS USED IN 'THE FIRST BOOK OF
                      KIM'.

0170                  TRANSO *      $0137  TRANSLATE TABLE LESS OFFSET $11
0170                  PZSCR  *      $00FC  PAGE ZERO SCRATCH LOCATION
0170                  POINTH *      $00FB  EXECUTE RAM POINTER HIGH
0170                  POINTL *      $00FA  EXECUTE RAM POINTER LOW
0170                  INH    *      $00F9  TERMINAL CHARACTER INPUT
0170                  SYMPAD *      $A400  OUTPUT PORT A ON 6532
0170                  SYMPBD *      $A402  OUTPUT PORT B ON 6532
0170                  SYMDIS *      $A640  DISPLAY BUFFER
0170                  SYMSCA *      $8906  LED OUTPUT DISPLAY BUFFER
0170                  SYMKEY *      $8923  CHECK FOR ANY KEY DOWN
0170                  SYMLRN *      $892C  DETERMINE KEY PRESSED
0170                  SYMSEG *      $8C29  LED SEGMENT CODES

0100                         ORG    $0100  OUT OF THE WAY ON STACK PAGE


                      ***********************************************
                      * SYM-1 VERSION OF KIM SCAND & SCANDS ROUTINES
                      ***********************************************

0100 A0 00            SCAND  LDYIM $0000   ENTER HERE TO GET BYTE
0102 B1 FA                   LDAIY POINTL  ADDRESSED BY POINTL
0104 85 F9                   STA   INH     AND MOVE IT TO INH AREA

0106 A0 00            SCANDS LDYIM $0000   ENTER HERE IF INH ALREADY STORED
0108 A5 FB                   LDA   POINTH  POINTH FIRST TO DISPLAY BUFFER
010A 20 1A 01                JSR   SPLITP
010D A5 FA                   LDA   POINTL  THEN DO POINTL
010F 20 1A 01                JSR   SPLITP
0112 A5 F9                   LDA   INH     LAST BUT NOT LEAST DO INH
0114 20 1A 01                JSR   SPLITP
0117 4C 06 89                JMP   SYMSCA  SET SYM MONITOR LIGHT & RETURN

011A 48               SPLITP PHA           SAVE ORIGINAL
011B 4A                      LSRA          ON STACK FOR LATER
011C 4A                      LSRA          SHIFT HI HALF TO LO HALF
011D 4A                      LSRA
011E 4A                      LSRA          WHICH IS 4 BITS DOWN
011F AA                      TAX           PUT INTO X AS AN INDEX
0120 BD 29 8C                LDAX  SYMSEG  GET APPROPRIATE SEGMENT CODE
0123 99 40 A6                STAY  SYMDIS  AND PUT INTO DISPLAY BUFFER
0126 C8                      INY           BUMP 'Y' FOR NEXT BYTE
0127 68                      PLA           NOW GET ORIGINAL VALUE BACK
0128 29 0F                   ANDIM $000F   KEEP ONLY LOW ORDER 4 BITS
012A AA                      TAX           AND REPEAT SEGMENT PROCESS
012B BD 29 8C                LDAX  SYMSEG
012E 99 40 A6                STAY  SYMDIS
0131 C8                      INY           INCLUDING BUMP FOR NEXT BYTE
0132 60                      RTS           AND RETURN
```

Writing to the displays is, again, a little more difficult than changing a set of addresses. It is also something that gets spread through the program, so I can't write a nice software solution as I did for the other routines. Fortunately, you can usually perform the same functions on the SYM as on the KIM in either the same or a smaller number of bytes. Less is as good as the same, since one can always add NOP's to pad it out.

The first problem is to set the data direction registers on the I/O ports to output to the displays. The normal code to look for in the KIM programs would be the following:

```
LDAIM  $7F
STA    $1741
```

On the SYM we need to set the two direction registers at $A401 and $A403. In order to do this in the same number of bytes we can make use of the SYM monitor CONFIG routine as follows:

```
LDAIM  $09
JSR    $89A5
```

This routine sets both I/O ports to output, and additionally stores zero in both I/O registers.

Individual digit selection is also different between the two systems, but both use a multiplex concept. This means that one I/O register determines which segments get lighted, and one register determines which digit is selected. The KIM hardware selects the leftmost digit with a 9 stored into location $1742. This is incremented by two for each digit to the right.

The SYM starts with a value of zero to location $A402. This needs to be increased by one for each digit to the right. You may be in for a little extra for those routines that increment and then check to see if they are done. Storing a 6 to location $A402 enables the onboard beeper, so if your routine suddenly starts beeping at you, don't be surprised. Tell everybody how great your sound effects are.

The actual segment codes are written to location $1740 on the KIM and $A400 on the SYM. These two addresses are one-for-one replacements. In order to convert routines that use these ports, change the address of the store instructions to the display, and find the place where the digit selector is bumped twice to get to the next digit, then simply NOP the second bump.

One final note about the timers. The KIM timer returns zero to a read before the clock has timed out, whereas the SYM returns the current clock count. This means that, in addition to changing the addresses, you will also have to change the branch after the check for clock expiration.

```
                    ****************************************
                    * SYM-1 VERSION OF KIM GETKEY SUBROUTINE
                    ****************************************

0133 20 2C 89  GETKEY JSR   SYMLRN GET SYM VERSION OF THE KEY

0136 D0 03            BNE   KEYDWN BRANCH IF ANY KEY IS DOWN
0138 A9 15     GKNONE LDAIM $0015  ELSE SET TO KIM NO KEY DOWN
013A 60               RTS          AND RETURN
013B 8A        KEYDWN TXA          X HOLDS INDEX INTO ASCII TABLE
013C C9 11            CMPIM $0011  NEED TO FUDGE KEY VALUE?
013E 90 07            BCC   GKRTS  00-0F IS OK 10=AD(KIM)=CR(SYM)
0140 C9 16            CMPIM $0016  CHECK FOR OUT OF KIM RANGE
0142 B0 F4            BCS   GKNONE AND TREAT AS A NO KEY
0144 BD 37 01         LDAX  TRANSO ELSE TRANSLATE THROUGH TABLE
0147 60        GKRTS  RTS          AND RETURN

0148 12        TRANST =     $12    '+'(KIM)='-/+'(SYM)
0149 11               =     $11    'DA'(KIM)='>/<'(SYM)
014A 15               =     $15    SHIFT (SYM)=NO KEY (KIM)
014B 13               =     $13    'G'(KIM)='GO/LP'(SYM)
014C 14               =     $14    'PC'(KIM)='REG/SP'(SYM)


                    ****************************************
                    * SYM-1 VERSION OF KIM KEYIN SUBROUTINE
                    ****************************************

014D 20 23 89  KEYIN  JSR   SYMKEY GET KEYBOARD STATUS
0150 D0 03            BNE   KEYIN2 REVERSE ZERO FLAG
0152 A2 FF            LDXIM $00FF  KIM NOT ZERO - NO KEY - FF FOR LRNKEY
0154 60               RTS
0155 A2 00     KEYIN2 LDXIM $0000  AND IS ZERO IF KEY IS DOWN
0157 60               RTS


                    ***************************************************
                    * SYM-1 VERSION OF KIM CONVD ROUTINES $1F48 & $1F4E
                    ***************************************************

0158 84 FC     CONVD  STY   PZSCR  SAVE Y IN SCRATCH AREA
015A A8               TAY          MOVE NIBBLE OF A TO INDEX REGISTER
015B B9 29 8C         LDAY  SYMSEG GET HEX SEGMENT CODES FROM TABLE
015E 8E 02 A4  DISPCH STX   SYMPBD SELECT THE DIGIT
0161 8D 00 A4         STA   SYMPAD OUTPUT THE SEGMENT CODES
0164 A0 10            LDYIM $0010  KEEP IT LIT FOR A WHILE
0166 88        LIGHT  DEY
0167 D0 FD            BNE   LIGHT


0169 8C 00 A4         STY   SYMPAD TURN ALL SEGMENTS OFF FOR NEXT ONE
016C E8               INX          BUMP X TO NEXT DIGIT
016D A4 FC            LDY   PZSCR  RESTORE THE Y REGISTER
016F 60               RTS          AND RETURN
ID=

-T

SYMBOL TABLE 2000 2096
CONVD  0158  DISPCH 015E  GETKEY 0133  GKNONE 0138
GKRTS  0147  INH    00F9  KEYDWN 013B  KEYIN  014D
KEYINR 0155  LIGHT  0166  POINTH 00FB  POINTL 00FA
PZSCR  00FC  SCAND  0100  SCANDS 0106  SPLITP 011A
SYMDIS A640  SYMKEY 8923  SYMLRN 892C  SYMPAD A400
SYMPBD A402  SYMSCA 8906  SYMSEG 8C29  TRANSO 0137
TRANST 0148
```

HDE inc.

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

HUDSON DIGITAL ELECTRONICS INC.

JUST THINK OF IT!

YOUR KIM-1 — No longer limited to those long cassette saves and loads.

YOUR KIM-1 — Backed up by 8K static ram so conservatively designed, well manufactured and thoroughly tested, HDE includes a no-nonsense, unconditional, 6 months parts and labor warranty. (Excluding misuse).

YOUR KIM-1 — Transformed into one of the most powerful 6502 development systems available today.

HDE, INC. supports the KIM-1 with 8" and 5" single and dual drive disk systems, prototyping cards, card racks, desk top cabinets, motherboards, and the finest memory board available, anywhere, at any price.

AND THIS IS JUST FOR STARTERS . . .

Consider: A fast, 2 pass assembler; a complete line oriented editor; a comprehensive text output processing system; an efficient dynamic debugging tool; and, a memory diagnostic package so thorough it's the basis of our memory warranty.

Plus, after the sale support that you've known you deserve, but just couldn't seem to get — until now.

And, HDE products are KIM-1, KIM-4 compatible. All boards include an oversized 5 volt regulator and address selection switches, in a state-of-the-art 4.5" X 6.5" format, designed, manufactured, and tested for commercial/industrial application.

HDE products — built to be used with confidence.

AVAILABLE 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

AMPERSORT

A fast, machine language sort utility for the APPLE II that handles integer, floating point and character records. Because it is callable from BASIC, this sort routine is a worthwhile addition to any software library.

Alan G. Hill
12092 Deerhorn Drive
Cincinnati, OH 45240

A sort utility is usually one of the first programs needed for records management application programs. If the utility is written in BASIC and runs under an interpreter, one quickly discovers that the sort is painfully slow on a micro. The sort program presented here, written in machine language for the APPLE II with AppleSoft ROM, will certainly remedy that problem. While no speed records will be set, it will run circles around BASIC, sorting 900 integer, 700 floating point, or 300 30-character records in about 60 seconds.

Speed is not the only beauty of AMPER-SORT. As its name implies, the BASIC-to-machine language interface utilizes the powerful, but not-widely-known, feature of AppleSoft — the Ampersand. What is the Ampersand and why is it so useful? Consider the following example of how a BASIC program passes sort parameters to AMPER-SORT:

```
100  &SRT#(AB$,0,10,7,10,A,1,5,D)
```

This statement, when embedded in a BASIC program or entered as an immediate command, will command AMPER-SORT to sort AB$(0) through AB$(10) in ascending order based on the 7th to 10th characters and in descending order for the 1st through 5th characters. Of course, POKEs could be used to pass parameters from other 6502 BASICs, but there's something more professionally pleasing about the Ampersand interface.

There is no user documentation from APPLE on the Ampersand feature. I first read of the feature in the October 1978 issue of *CALL APPLE*. When the AppleSoft interpreter encounters an ampersand (&) character at the beginning of a BASIC statement, it does a JSR $3F5. If the user has placed a JMP instruction there, a link is made to the user's machine language routine. APPLE has thoughtfully provided some ampersand handling routines described in the November and December issues of *CALL APPLE*. The routines enable your machine language routine to examine and convert the characters or expressions following the ampersand. The routines used in AMPER-SORT are:

CHRGET ($00B1)

This routine will return, in the accumulator, the next character in the statement.

The first character is in the accumulator when the JSR $3F5 occurs. The zero flag is set if the character is an end-of-line token (00) or statement terminator ($3A). The carry flag is set if the character is non-numeric, and cleared if it is numeric. The character pointer at $B8 and $B9 is advanced automatically so that the next JSR $B1 will return the next character. A JSR $B7 will return a character without advancing the pointer.

FRMNUM ($DD67)

This routine evaluates an expression of variables and constants in the ampersand statement from the current pointer to the next comma. The result is placed in the floating point accumulator.

GETADR ($E752)

This routine will convert the floating point accumulator to a two-byte integer and place it in $50 and $51. FRMNUM and GETADR are used by AMPER-SORT to retrieve the sort parameters and convert each to an unteger.

GETBYT ($E6F8)

This routine will retrieve the next expression and return it as a one-byte interger in the X-register.

It is the user's responsibility to leave the $B8 and $B9 pointer at the terminator.

Parameters are passed to AMPER-SORT in the following form:

```
100  &SRT#(AB$,B,E,7,10,A,1,5,D)
```

where:

- AB$ — is the variable name of the string array to be sorted. The general form is XX$ for string arrays, XX% for integer arrays, and XX for floating point arrays.
- B — is a variable, constant or expression containing the value of the subscript element where the sort is to begin, e.g. AB$(B).
- E — is a variable or constant or expression containing the value of the subscript element where the sort is to end, e.g. AB$(E). B and E are useful when the AB$ array is partially filled or has been sectioned into logically separate blocks that need to be sorted independently.
- 7 — is a variable, constant or expression specifying the beginning position of the major sort field.
- 10 — is a variable, constant or expression specifying the ending position of the major sort field.
- A — is a character specifying that the major sort field is to be sorted in ascending order.
- 1 — is a variable, constant or expression specifying the beginning position of the first minor sort field.
- 5 — is a variable, constant or expression specifying the ending position of the first minor sort field.
- D — is a character specifying that the first minor sort field is to be sorted in descending order.

The &SRT command will sort character, integer or floating point arrays and can be used in either the immediate or deferred execution mode similar to other AppleSoft BASIC commands. Of course, the named array must have been previously dimensioned and initialized in either case.

A. Character Arrays
   1. Equal or unequal element lengths
   2. Some or all elements
   3. Ascending or descending order
   4. A major sort field and up to 4 minor sort fields

Examples:

```
 10  DIM NA$(500)

100  &SRT#(NA$,0,500,1,5,A)
200  &SRT#(NA$,0,500,1,5,A,6,10,
     D,11,11,A)
299  F% = 0: L = 10
300  &SRT = (NA$,F%,L,10,15,D)
```

Line 100 sorts on positions 1 through 5 in ascending order for all 501 elements of NA$(500).

Line 200 is the same as Line 100 except that minor sort fields are specified. The sort sequence on positions 1-5 is in ascending order, positions 6-10 are in descending order, and position 11 is ascending order.

Line 299 and 300 sort on positions 10-15 in descending order for NA$(0) through NA$(10).

B. Integer and Floating Point Arrays

1. Some or all elements
2. Ascending order only. (Step through the array backwards if needed in descending order.)

Examples:

```
10   DIM AB%(100),FP(100)

100  &SRT#(AB%,0,100)
299  S=50:E=100
300  &SRT#(AB%,S,E)
399  X=49
400  &SRT#(FP,0,X)
```

Line 100 sorts all 101 elements of AB%(100) in ascending order. Lines 299 and 300 sort from AB%(50) through AB%(100), while lines 399 and 400 sort from FP(0) through FP(49).

Limited editing has been included in the parameter processing code. Therefore, one must be careful to observe such rules as:

1. $0 \leq B < E \leq$ maximum number of AB$ elements.
2. AB$ must be a scalar array. e.g. AB$(10), not AB$(20,40).
3. The sort array name must be less than 16 characters only the first two count, and they must be unique.
4. The maximum number of sort fields is 5.
5. The beginning sort field position must not be greater than the ending sort field position.

Options:

1. Constants, variables, or expressions may be used for subscript bounds and sort positions.
2. The &SRT command may be used in immediate or deferred execution mode.

Some editing checks are made. You will notice this when you get a "?SYNTAX ERROR IN LINE XXX" error message. You will also get a "VARIABLE XXX NOT FOUND" message if the routine cannot find the AB$ variable name in variable space.

The AMPER-SORT program is listed in its entirety. A BASIC demo program is also shown. Anyone desiring a cassette tape containing the latest version of the object code assembled at $5200, a copy assembled at $9200, and the source program text in the Microproducts APPLE II Assembler format may receive these by sending the author $5.00 at the above address.

AMPER-SORT Demo

```
1000  GOTO 10000
1050  REM  CHARACTER SORT
1060 CH$ = "ABCDWXYZ":L =  LEN (CH$) - 1
1070 N% = 8
1080  DIM AB$(N%)
1090  FOR I = 0 TO N%
1100 C$ =  MID$ (CH$, INT ( RND (1) * L) + 1,1)
1110 B$ =  MID$ (CH$, INT ( RND (1) * L) + 1,1)
1120  FOR J = 1 TO 3
1130 C$ = C$ + C$:B$ = B$ + B$
1140  NEXT J
1150 AB$(I) = B$ + C$
1160  NEXT I
1170  GOSUB 1240
1180  REM  SORT HALF ASCENDING
1190  REM  SORT HALF DESCENDING
1200  & SRT#(AB$,0,N%,1,8,A,9,16,D)
1210  GOSUB 1260
1220  GOTO 11000
1230  REM  PRINT ROUTINE
1240  PRINT "    BEFORE"
1250  GOTO 1270
1260  PRINT "    AFTER": PRINT "ASCEND  DESCEND"
1270  FOR I = 0 TO N%
1280  PRINT AB$(I): NEXT I: RETURN
2000  REM  INTEGER SORT
2010 N% = 8
2020  DIM IN%(N%)
2030  FOR I = 0 TO N%
2040 IN%(I) = 7500 -  INT ( RND (1) * 15000 )
2050  NEXT I
2060  GOSUB 2120
2070  REM  SORT
2080  & SRT#(IN%,0,N%)
2090  GOSUB 2130
2100  GOTO 11000
2110  REM  PRINT ROUTINE
2120  HTAB 10: PRINT "BEFORE": GOTO 2140
2130  HTAB 10: PRINT "AFTER"
2140  FOR I = 0 TO N%
2150  PRINT IN%(I): NEXT I: RETURN
3000  REM  FLOATING POINT
3010 T% = 8
3020  DIM FP(T%)
3030  FOR I = 0 TO 8
3040 FP(I) = 1000 *  RND (1) *  SIN (I * 7.16)
3050  NEXT I
3060  GOSUB 3120
3070  REM  SORT
3080  & SRT#(FP,0,T%)
3090  GOSUB 3130
3100  GOTO 11000
3110  REM  PRINT ROUTINE
3120  HTAB 10: PRINT "BEFORE": GOTO 3140
3130  HTAB 10: PRINT "AFTER"
3140  FOR I = 0 TO T%
3150  PRINT FP(I): NEXT I: RETURN
```

```
                      : ***********************
                      : *      AMPER-SORT      *
                      : *          BY          *
                      : *     ALAN G. HILL     *
                      : *      APRIL, 1979     *
                      : *  COMMERCIAL RIGHTS   *
                      : *       RESERVED       *
                      : ***********************
                      NAPT .DL 00D0
                      NMS1 .DL 00D4
                      ASI1 .DL 00D6
                      CSI1 .DL 00D8
                      ASI2 .DL 00DA
                      CSI2 .DL 00DC
                      IIII .DL 00DE
                      NNNN .DL 00E0
                      FSTR .DL 00E2
                      FLEN .DL 00E7
                      DISP .DL 00EC
                      JJJJ .DL 00ED
                      LENI .DL 00EF
                      LENJ .DL 00F0
                      TYPE .DL 00F1
                      ZZ50 .DL 0050
                      ZZ6B .DL 006B
                      CHRG .DL 00B1
                      GETB .DL E6F8
                      SNER .DL DEC9
                      FRNM .DL DD67
                      GETA .DL E752
                      MPLY .DL FB63
                      COUT .DL FDED
                           .OR 5200
                      :
                      : PROCESS '&'
                                                ENTER WITH FIRST CHAR
5200-   48            SORT PHA                  
5201-   20 DE 54           JSR SVZP             SAVE A WORK AREA IN ZERO PAGE
5204-   68                 PLA
5205-   A2 00              LDX 00
5207-   DD 24 55      SR01 CMP SRTS,X           EDIT FOR 'SRT#('
520A-   D0 46              BNE ERRX             SIGNAL 'SYNTAX ERROR'
520C-   20 B1 00           JSR CHRG             GET NEXT CHARACTER
520F-   E8                 INX
5210-   E0 05              CPX 05
5212-   D0 F3              BNE SR01
5214-   A2 00              LDX 00               OK SO FAR
5216-   F0 03              BEQ VNAM
5218-   20 B1 00      SR04 JSR CHRG             GET ANOTHER CHARACTER
521B-   C9 2C         VNAM CMP ',               LOOP TO GET ARRAY NAME
521D-   F0 0A              BEQ SR05
521F-   9D 6A 55           STA NAME,X           SAVE NAME
5222-   E8                 INX
5223-   E0 10              CPX 10               16 CHARACTERS IS LONG
5225-   D0 F1              BNE SR04             ENOUGH FOR A NAME
5227-   F0 29              BEQ ERRX             SIGNAL ERROR
5229-   CA            SR05 DEX
522A-   BD 6A 55           LDA NAME,X           WHAT TYPE
522D-   C9 24              CMP '$
522F-   F0 24              BEQ CHAR             CHARACTER
5231-   C9 25              CMP '%
5233-   D0 15              BNE FP00             FLOATING POINT
                      :
                      : INTEGER SORT
5235-   A2 01         INTE LDX 01               INTEGER
5237-   A9 80         INT1 LDA 80
5239-   1D 6A 55           ORA NAME,X           NEG. ASCII
523C-   9D 6A 55           STA NAME,X
523F-   CA                 DEX
5240-   10 F5              BPL INT1
5242-   A9 02              LDA 02               INITIALIZE DISPLACEMENT
5244-   85 EC              STA *DISP
5246-   A9 01              LDA 01
5248-   D0 19              BNE SR06
                      :
```

```
                    : F.P. SORT
524A-   A9 05       FP00 LDA 05
524C-   85 EC            STA *DISP
524E-   A9 02            LDA 02
5250-   D0 11            BNE SR06
                    :
5252-   4C A5 52    ERRX JMP ERRO
                    :
                    : CHARACTER SORT
5255-   A9 80       CHAR LDA 80
5257-   0D 6B 55         ORA NAME+01       NEG. ASCII
525A-   8D 6B 55         STA NAME+01
525D-   A9 03            LDA 03
525F-   85 EC            STA *DISP
5261-   A9 00            LDA 00
                    :
                    : **  SET UP SORT LIMITS  **
5263-   85 F1       SR06 STA *TYPE         0=CH 1=INT 2=FP
5265-   20 B1 00         JSR CHRG          NOW GET SUBSCRIPTS
5268-   20 67 DD         JSR FRNM          AND PUT IN F.P. ACC.
526B-   20 52 E7         JSR GETA          CONVERT TO INTEGER
526E-   A5 50            LDA *ZZ50
5270-   85 DE            STA *IIII         FIRST SUBSCRIPT
5272-   A5 51            LDA *ZZ50+01
5274-   85 DF            STA *IIII+01
5276-   20 B1 00         JSR CHRG
5279-   20 67 DD         JSR FRNM
527C-   20 52 E7         JSR GETA
527F-   A5 50            LDA *ZZ50
5281-   85 D4            STA *NMS1         LAST SUBSCRIPT INTO N-1
5283-   18               CLC
5284-   69 01            ADC 01
5286-   85 E0            STA *NNNN         N
5288-   A5 51            LDA *ZZ50+01
528A-   85 D5            STA *NMS1+01
528C-   69 00            ADC 00
528E-   85 E1            STA *NNNN+01
5290-   A5 F1            LDA *TYPE
5292-   D0 59            BNE TERM          BRANCH NOT CHARACTER SORT
5294-   F0 15            BEQ SR16
                    :
                    : ***  ERROR  ***
5296-   A2 00       ERR3 LDX 00
5298-   BD 29 55    SR11 LDA MSG1,X        ARRAY VARIABLE NAME
529B-   09 80            ORA 80            NOT FOUND
529D-   20 ED FD         JSR COUT          NOTIFY USER
52A0-   E8               INX
52A1-   E0 17            CPX 17
52A3-   D0 F3            BNE SR11
52A5-   20 01 55    ERRO JSR RSZP          RESTORE ZERO PAGE AND
52A8-   4C C9 DE         JMP SNER          SIGNAL SYNTAX ERROR
                    :
                    : **  GET SORT FIELDS  **
52AB-   A0 00       SR16 LDY 00
52AD-   8C 81 55         STY SAVY
52B0-   20 B1 00    SR17 JSR CHRG          GET NEXT CHARACTER
52B3-   20 F8 E6         JSR GETB
52B6-   CA               DEX
52B7-   AC 81 55         LDY SAVY
52BA-   96 E2            STX *FSTR,Y       START COLUMN -1
52BC-   20 B1 00         JSR CHRG
52BF-   20 F8 E6         JSR GETB
52C2-   AC 81 55         LDY SAVY
52C5-   96 E7            STX *FLEN,Y       END COLUMN
52C7-   20 B1 00         JSR CHRG
52CA-   90 D9            BCC ERRO          SHOULD BE 'A' OR 'D'
52CC-   C9 44            CMP 'D
52CE-   F0 04            BEQ SR07          DESCENDING
52D0-   A9 FF            LDA 0FF           ASCENDING
52D2-   30 02            BMI SR09
52D4-   A9 00       SR07 LDA 00
52D6-   99 7A 55    SR09 STA UPDN,Y        SAVE SEQUENCE
52D9-   C8               INY
```

```
52DA-    8C 81 55   1480        STY SAVY
52DD-    20 B1 00   1490        JSR CHRG
52E0-    C9 29      1500        CMP ')
52E2-    F0 06      1510        BEQ LAST
52E4-    C9 2C      1520        CMP ',
52E6-    F0 C8      1530        BEQ SR17      LOOP FOR NEXT SORT FIELD PARMS
52E8-    D0 BB      1540        BNE ERRO
52EA-    8C 80 55   1550   LAST STY PRSN      NO. OF SORT FIELDS
52ED-    20 B1 00   1560   TERM JSR CHRG      MUST BE TERMINATOR
52F0-    D0 B3      1570        BNE ERRO      IT WASN'T
                    1580   :
                    1590   : SEARCH SORT ARRAY NAME
52F2-    A0 00      1600   MC20 LDY 00
52F4-    B1 6B      1610        LDA (ZZ6B),Y
52F6-    CD 6A 55   1620        CMP NAME
52F9-    D0 08      1630        BNE MC22
52FB-    C8         1640        INY           FOUND FIRST CHARACTER
52FC-    B1 6B      1650        LDA (ZZ6B),Y
52FE-    CD 6B 55   1660        CMP NAME+01
5301-    F0 2B      1670        BEQ SETN      FOUND BOTH
5303-    18         1680   MC22 CLC           KEEP LOOKING
5304-    A0 02      1690        LDY 02
5306-    B1 6B      1700        LDA (ZZ6B),Y
5308-    65 6B      1710        ADC *ZZ6B
530A-    48         1720        PHA
530B-    C8         1730        INY
530C-    B1 6B      1740        LDA (ZZ6B),Y
530E-    65 6C      1750        ADC *ZZ6B+01
5310-    85 6C      1760        STA *ZZ6B+01
5312-    68         1770        PLA
5313-    85 6B      1780        STA *ZZ6B
5315-    C5 6D      1790        CMP $6D
5317-    A5 6C      1800        LDA *ZZ6B+01
5319-    E5 6E      1810        SBC $6E
531B-    B0 03      1820        BCS SR27        NO LUCK. OUT OF BOUNDS
531D-    4C F2 52   1830        JMP MC20


                    1840   :
                    1850   :  ** NAME NOT FOUND **
5320-    A2 02      1860   SR27 LDX 02
5322-    BD 6A 55   1870   SR28 LDA NAME,X
5325-    9D 33 55   1880        STA VARI,X      PUT NAME IN BUFFER
5328-    CA         1890        DEX
5329-    10 F7      1900        BPL SR28
532B-    4C 96 52   1910        JMP ERR3        SEND A MESSAGE


                    1920   :
                    1930   : * INITIALIZE ARRAY POINTER *
532E-    18         1940   SETN CLC             FOUND VARIABLE NAME OF
532F-    A5 6B      1950        LDA *ZZ6B        ARRAY TO BE SORTED.
5331-    69 07      1960        ADC 07           COMPUTE ADDRESS OF
5333-    85 52      1970        STA $52          STRING LENGTH BYTE.
5335-    A5 6C      1980        LDA *ZZ6B+01
5337-    69 00      1990        ADC 00
5339-    85 53      2000        STA $53
533B-    A5 DE      2010        LDA *IIII        (6B.6C)+7+DISP*IIII
533D-    85 50      2020        STA *ZZ50
533F-    A5 DF      2030        LDA *IIII+01
5341-    85 51      2040        STA *ZZ50+01
5343-    A5 EC      2050        LDA *DISP
5345-    85 54      2060        STA $54
5347-    A9 00      2070        LDA 00
5349-    85 55      2080        STA $55
534B-    20 63 FB   2090        JSR MPLY         ROM MULTIPLY ROUTINE
534E-    A5 50      2100        LDA *ZZ50
5350-    85 D6      2110        STA *ASII        SAVE ADDRESS FOR MUCH USE
5352-    A5 51      2120        LDA *ZZ50+01
5354-    85 D7      2130        STA *ASII+01
5356-    4C 66 53   2140        JMP SR22
```

```
                        2150  :
                        2160  : **** BEGIN SORT ****
                        2170  :
                        2180  :   ** FOR I=II TO N-1 LOOP **
5359-    18             2190  CONI CLC
535A-    A5 D6          2200       LDA *ASII
535C-    65 EC          2210       ADC *DISP       NEXT I ADDRESS
535E-    85 D6          2220       STA *ASII
5360-    A5 D7          2230       LDA *ASII+01
5362-    69 00          2240       ADC 00
5364-    85 D7          2250       STA *ASII+01
5366-    A0 01          2260  SR22 LDY 01
5368-    B1 D6          2270       LDA (ASII),Y    GET ADDRESS OF THE
536A-    85 D8          2280       STA *CSII       CHARACTER STRING
536C-    C8             2290       INY
536D-    B1 D6          2300       LDA (ASII),Y
536F-    85 D9          2310       STA *CSII+01
5371-    18             2320       CLC
5372-    A5 D6          2330       LDA *ASII       ALSO NEED ADDRESS OF
5374-    65 EC          2340       ADC *DISP       ADJACENT ELEMENT FOR
5376-    85 DA          2350       STA *ASI2       BUBBLE SORT COMPARISON
5378-    A5 D7          2360       LDA *ASII+01
537A-    69 00          2370       ADC 00
537C-    85 DB          2380       STA *ASI2+01
537E-    18             2390       CLC
537F-    A5 DE          2400       LDA *IIII
5381-    69 01          2410       ADC 01
5383-    85 ED          2420       STA *JJJJ      J=I+1
5385-    A5 DF          2430       LDA *IIII+01
5387-    69 00          2440       ADC 00
5389-    85 EE          2450       STA *JJJJ+01
538B-    4C 9B 53       2460       JMP SR24
                        2470  :
                        2480  :   ** FOR J=I+1 TO N LOOP **
538E-    18             2490  CONJ CLC
538F-    A5 DA          2500       LDA *ASI2
5391-    65 EC          2510       ADC *DISP      INCREMENT AB$(J) ADDRESS
5393-    85 DA          2520       STA *ASI2
5395-    A5 DB          2530       LDA *ASI2+01
5397-    69 00          2540       ADC 00
5399-    85 DB          2550       STA *ASI2+01
539B-    A0 01          2560  SR24 LDY 01
539D-    B1 DA          2570       LDA (ASI2),Y
539F-    85 DC          2580       STA *CSI2      GET NEW STRING ADDRESS
53A1-    C8             2590       INY
53A2-    B1 DA          2600       LDA (ASI2),Y
53A4-    85 DD          2610       STA *CSI2+01
53A6-    A5 F1          2620       LDA *TYPE
53A8-    F0 03          2630       BEQ CHST       CHARACTER SORT
53AA-    4C 2F 54       2640       JMP NCHH
                        2650  :
                        2660  :   ** CHARACTER SORT **
53AD-    A0 00          2670  CHST LDY 00
53AF-    B1 D6          2680       LDA (ASII),Y   STRING LENGTH
53B1-    F0 52          2690       BEQ MC40       NULL STRING: SKIP
53B3-    85 EF          2700       STA *LENI      SAVE LEN(AB$(I))
53B5-    B1 DA          2710       LDA (ASI2),Y
53B7-    F0 4C          2720       BEQ MC40
53B9-    85 F0          2730       STA *LENJ      SAVE LEN(AB$(J))
53BB-    A2 00          2740       LDX 00
53BD-    B4 E2          2750  SR29 LDY *FSTR,X   STARTING SORT COLUMN
53BF-    BD 7A 55       2760  MC33 LDA UPDN,X    SEQUENCE
53C2-    30 0C          2770       BMI ASND       BRANCH ASCENDING
53C4-    B1 D8          2780       LDA (CSII),Y   CHARACTER BY CHARACTER
53C6-    D1 DC          2790       CMP (CSI2),Y   COMPARISON FOR DESCENDING
53C8-    B0 14          2800       BGE MC26       POSSIBLE SWAP
53CA-    20 B9 54       2810       JSR SWAP       DEFINITE SWAP
53CD-    4C 05 54       2820       JMP MC40                 NEXT RECORD
53D0-    B1 D8          2830  ASND LDA (CSII),Y  ASCENDING
53D2-    D1 DC          2840       CMP (CSI2),Y
53D4-    90 2F          2850       BLT MC40       NO SWAP: NEXT RECORD
53D6-    F0 19          2860       BEQ MC27       POSSIBLE SWAP
53D8-    20 B9 54       2870  MC25 JSR SWAP      SWAP
53DB-    4C 05 54       2880       JMP MC40       NEXT RECORD
```

```
53DE-   D0 25        MC26 BNE MC40          NO SWAP
53E0-   C8                INY               LOOK AT REMAINING CHARACTERS
53E1-   C4 EF             CPY *LENI
53E3-   F0 06             BEQ MC39          UP TO THE LIMITS OR UNTIL
53E5-   C4 F0             CPY *LENJ
53E7-   F0 16             BEQ MC29          WE FIND A REASON TO SWAP
53E9-   90 0F             BLT MC28
53EB-   C4 F0        MC39 CPY *LENJ
53ED-   90 E9             BLT MC25          SWAP
53EF-   F0 0E             BEQ MC29          NO SWAP
53F1-   C8           MC27 INY
53F2-   C4 EF             CPY *LENI
53F4-   F0 09             BEQ MC29
53F6-   C4 F0             CPY *LENJ
53F8-   F0 DE             BEQ MC25
53FA-   98           MC28 TYA
53FB-   D5 E7             CMP *FLEN,X     END OF SORT FIELD?
53FD-   D0 C0             BNE MC33        BRANCH NO
53FF-   E8           MC29 INX
5400-   EC 80 55          CPX PRSN        YES. ANY MORE FIELDS?
5403-   D0 B8             BNE SR29
                     :
                     : ** NEXT J **
5405-   E6 ED        MC40 INC *JJJJ
5407-   D0 02             BNE MC38
5409-   E6 EE             INC *JJJJ+01    J=J+1
540B-   A5 ED        MC38 LDA *JJJJ
540D-   C5 E0             CMP *NNNN       J=N?
540F-   A5 EE             LDA *JJJJ+01
5411-   E5 E1             SBC *NNNN+01
5413-   90 14             BCC JMPJ        BRANCH NO
                     :
                     : ** NEXT I **
5415-   E6 DE             INC *IIII
5417-   D0 02             BNE MC41
5419-   E6 DF             INC *IIII+01    I=I+1
541B-   A5 DE        MC41 LDA *IIII
541D-   C5 D4             CMP *NMS1       I=N-1?
541F-   A5 DF             LDA *IIII+01
5421-   E5 D5             SBC *NMS1+01
5423-   90 07             BCC JMPI        BRANCH NO
                     :
                     : ****  SORT DONE    ****
5425-   20 01 55     SDON JSR RSZP        RESTORE ZERO PAGE
5428-   60                RTS
5429-   4C 8E 53     JMPJ JMP CONJ
542C-   4C 59 53     JMPI JMP CONI
542F-   18           NCHH CLC             NOT A CHARACTER SORT SO
5430-   6A                ROR             IT MUST BE INTEGER OR F.P.
5431-   B0 03             BCS INTC        IT'S INTEGER
5433-   4C 6D 54          JMP FPCC        IT'S FLOATING POINT
                     :
                     : ** INTEGER SORT **
5436-   A0 01        INTC LDY 01
5438-   B1 D6             LDA (ASII),Y    ASCENDING ORDER ONLY
543A-   D1 DA             CMP (ASI2),Y
543C-   88                DEY             COMPARE IN%(I) WITH IN%(J)
543D-   B1 D6             LDA (ASII),Y
543F-   F1 DA             SBC (ASI2),Y
5441-   90 22             BCC NOSP        POSSIBLE SWAP
5443-   B1 D6             LDA (ASII),Y
5445-   51 DA             EOR (ASI2),Y
5447-   30 BC             BMI MC40
                     :
                     :  ** SWAP I WITH J **
5449-   C8           SWIN INY
544A-   B1 DA             LDA (ASI2),Y
544C-   48                PHA
544D-   88                DEY
544E-   B1 DA             LDA (ASI2),Y    SWAP IN%(I) WITH IN%(J)
5450-   48                PHA
5451-   B1 D6             LDA (ASII),Y
```

```
5453-   91 DA             STA (ASI2),Y
5455-   C8                INY
5456-   B1 D6             LDA (ASII),Y
5458-   91 DA             STA (ASI2),Y
545A-   88                DEY
545B-   68                PLA
545C-   91 D6             STA (ASII),Y
545E-   C8                INY
545F-   68                PLA
5460-   91 D6             STA (ASII),Y
5462-   4C 05 54          JMP MC40          NEXT RECORD
5465-   B1 D6       NOSP  LDA (ASII),Y
5467-   51 DA             EOR (ASI2),Y
5469-   30 DE             BMI SWIN          SWAP
546B-   10 98             BPL MC40
                    :
                    :  ** FLOATING POINT SORT **
546D-   A0 00       FPCC  LDY 00
546F-   38          FP01  SEC
5470-   B1 D6             LDA (ASII),Y
5472-   F1 DA             SBC (ASI2),Y
5474-   F0 04             BEQ FP02
5476-   10 1F             BPL FPSP
5478-   30 07             BMI MBSP          THIS BIT OF CONVOLUTED
547A-   C8          FP02  INY               LOGIC TELLS ME IF
547B-   C0 05             CPY 05            FP(I) IS GREATER THAN,
547D-   D0 F0             BNE FP01          EQUAL TO, OR LESS THAN
547F-   F0 3E             BEQ JM40          FP(J).
5481-   A0 01       MBSP  LDY 01
5483-   B1 D6             LDA (ASII),Y      A TRUTH TABLE HELPS
5485-   31 DA             AND (ASI2),Y
5487-   11 DA             ORA (ASI2),Y
5489-   30 20             BMI FP03
548B-   88                DEY
548C-   B1 DA             LDA (ASI2),Y
548E-   D0 2F             BNE JM40
5490-   C8                INY
5491-   B1 D6             LDA (ASII),Y
5493-   10 16             BPL FP03
5495-   30 28             BMI JM40
5497-   A0 01       FPSP  LDY 01
5499-   B1 D6             LDA (ASII),Y
549B-   31 DA             AND (ASI2),Y
549D-   11 D6             ORA (ASII),Y
549F-   30 1E             BMI JM40
54A1-   88                DEY
54A2-   B1 D6             LDA (ASII),Y
54A4-   D0 05             BNE FP03
54A6-   C8                INY
54A7-   B1 DA             LDA (ASI2),Y
54A9-   10 14             BPL JM40
54AB-   A0 04       FP03  LDY 04
54AD-   B1 D6       FP04  LDA (ASII),Y      SAVE FP(I) IN STACK
54AF-   48                PHA
54B0-   88                DEY
54B1-   10 FA             BPL FP04
54B3-   C8          FP08  INY
54B4-   B1 DA             LDA (ASI2),Y
54B6-   91 D6             STA (ASII),Y      SWAP
54B8-   68                PLA
54B9-   91 DA             STA (ASI2),Y
54BB-   C0 04             CPY 04
54BD-   D0 F4             BNE FP08
54BF-   4C 05 54    JM40  JMP MC40          NEXT RECORD
54C2-   A0 00       SWAP  LDY 00
54C4-   B1 D6             LDA (ASII),Y
54C6-   48                PHA               ROUTINE TO SWAP THE
54C7-   C8                INY
54C8-   A5 D8             LDA *CSII         CHARACTER POINTERS FOR
54CA-   91 DA             STA (ASI2),Y
54CC-   C8                INY               CHARACTER SORT.
54CD-   A5 D9             LDA *CSII+01
54CF-   91 DA             STA (ASI2),Y
```

```
54D1-    A5 DD          4280         LDA *CSI2+01
54D3-    91 D6          4290         STA (ASII),Y
54D5-    85 D9          4300         STA *CSII+01
54D7-    88             4310         DEY
54D8-    A5 DC          4320         LDA *CSI2
54DA-    91 D6          4330         STA (ASII),Y
54DC-    85 D8          4340         STA *CSII
54DE-    88             4350         DEY
54DF-    B1 DA          4360         LDA (ASI2),Y
54E1-    91 D6          4370         STA (ASII),Y
54E3-    68             4380         PLA
54E4-    91 DA          4390         STA (ASI2),Y
54E6-    60             4400         RTS
54E7-    A2 00          4410    SVZP LDX 00            SAVE SOME OF APPLESOFT'S
54E9-    B5 D0          4420    MC51 LDA *NAPT,X       ZERO PAGE.  SORT ROUTINE
54EB-    9D 49 55       4430         STA ZPSV,X        NEEDS SOME ROOM TO WORK.
54EE-    E8             4440         INX
54EF-    E0 22          4450         CPX 22
54F1-    D0 F6          4460         BNE MC51
54F3-    A5 6B          4470         LDA *ZZ6B         ALSO $6B.6C
54F5-    8D 71 55       4480         STA SV6B
54F8-    A5 6C          4490         LDA *ZZ6B+01
54FA-    8D 72 55       4500         STA SV6B+01
54FD-    A2 00          4510         LDX 00
54FF-    B5 50          4520    MC55 LDA *ZZ50,X       ALSO $50.55
5501-    9D 6B 55       4530         STA SV50,X
5504-    E8             4540         INX
5505-    E0 06          4550         CPX 06
5507-    D0 F6          4560         BNE MC55
5509-    60             4570         RTS
550A-    A2 00          4580    RSZP LDX 00            RESTORE ZERO PAGE DATA
550C-    BD 49 55       4590    MC61 LDA ZPSV,X
550F-    95 D0          4600         STA *NAPT,X
5511-    E8             4610         INX
5512-    E0 22          4620         CPX 22
5514-    D0 F6          4630         BNE MC61
5516-    AD 71 55       4640         LDA SV6B
5519-    85 6B          4650         STA *ZZ6B
551B-    AD 72 55       4660         LDA SV6B+01
551E-    85 6C          4670         STA *ZZ6B+01
5520-    A2 00          4680         LDX 00
5522-    BD 6B 55       4690    MC65 LDA SV50,X
5525-    95 50          4700         STA *ZZ50,X
5527-    E8             4710         INX
5528-    E0 06          4720         CPX 06
552A-    D0 F6          4730         BNE MC65
552C-    60             4740         RTS
                        4750    :
552D- 53 52 54          4760    SRTS .AS 'SRT#('
5530- 23 28             4770    MSG1 .HS 8D
5532- 8D                4780         .AS 'VARIABLE '
5533- 56 41 52 49       4790    VARI .HS 202020
5537- 41                4800         .AS ' NOT FOUND'
5538- 42 4C 45          4810    ZPSV .HS 0000000000000000
553B- 20 20 20          4820         .HS 0000000000000000
553E- 20 20             4830         .HS 0000000000000000
5540- 4E 4F             4840         .HS 0000000000000000
5542- 54 20 46 4F       4850         .HS 0000
5546- 55 4E             4860    SV50 .HS 000000000000
5548- 44                4870    SV6B .HS 0000
                        4880    NAME .HS 0000000000000000
                        4890         .HS 0000000000000000
                        4900    UPDN .HS 0000000000
                        4910    INDS .HS 00
                        4920    PRSN .HS 00
                        4930    SAVY .HS 00
                        4940         .EN
```

```
10000  REM   ** &SORT DEMO **
10010  REM  SAVE ROOM FOR
10020  REM  SORT ROUTINE
10030  HIMEM: 20992: REM  $5200
10040 D$ =  CHR$ (4)
10050  PRINT D$;"BLOAD B.AMPER-SORT"
10060  REM  SET UP '&' HOOK
10070  REM   AT $3F5:JMP $5200
10080  POKE 1013,76: POKE 1014,0: POKE 1015,82
10090  HOME : CLEAR
10100  VTAB 8: HTAB 15: PRINT "SORT DEMO"
10110  PRINT : HTAB 15: PRINT "SELECTIONS"
10120  PRINT : HTAB 10: PRINT "1  INTEGER SORT"
10130  HTAB 10: PRINT "2  FLOATING POINT SORT"
10140  HTAB 10: PRINT "3  CHARACTER SORT"
10150  HTAB 10: PRINT "4  EXIT"
10160  VTAB 17: INPUT "SELECTION ";SE%
10170  IF SE% < 0 OR SE% > 4 THEN 10090
10180  ON SE% GOTO 2000,3000,1050,10190
10190  END
11000  PRINT "HIT ANY KEY TO RETURN TO MENU"
11010  WAIT  - 16384,128
11020  POKE  - 16368,0
11030  GOTO 10090
```

```
]RUN
              SORT DEMO

              SELECTIONS

         1  INTEGER SORT
         2  FLOATING POINT SORT
         3  CHARACTER SORT
         4  EXIT



SELECTION 1
          BEFORE
7153
335
-1300
-4376
-6944
4948
-2914
3416
-2955
          AFTER
-6944
-4376
-2955
-2914
-1300
335
3416
4948
7153
HIT ANY KEY TO RETURN TO MENU
```

```
              SORT DEMO

              SELECTIONS

         1  INTEGER SORT
         2  FLOATING POINT SORT
         3  CHARACTER SORT
         4  EXIT



SELECTION 2
          BEFORE
0
65.0306039
831.056575
483.823094
-296.508742
-370.915344
-226.85172
-61.023044
353.768754
          AFTER
-370.915344
-296.508742
-226.85172
-61.023044
0
65.0306039
353.768754
483.823094
831.056575
HIT ANY KEY TO RETURN TO MENU
```

```
            SORT DEMO

            SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT
SELECTION 3
    BEFORE
XXXXXXXXCCCCCCCC
AAAAAAAADDDDDDDD
DDDDDDDDBBBBBBBB
AAAAAAAAXXXXXXXX
CCCCCCCCAAAAAAAA
YYYYYYYYCCCCCCCC
YYYYYYYYWWWWWWWW
BBBBBBBBWWWWWWWW
XXXXXXXXBBBBBBBB
    AFTER
ASCEND  DESCEND
AAAAAAAAXXXXXXXX
AAAAAAAADDDDDDDD
BBBBBBBBWWWWWWWW
CCCCCCCCAAAAAAAA
DDDDDDDDBBBBBBBB
XXXXXXXXCCCCCCCC
XXXXXXXXBBBBBBBB
YYYYYYYYWWWWWWWW
YYYYYYYYCCCCCCCC
HIT ANY KEY TO RETURN TO MENU


            SORT DEMO

            SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT



SELECTION 11
            SORT DEMO

            SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT
```

```
SELECTION 1
         BEFORE
-103
-3561
-5898
3111
2627
-1089
7465
2340
-5242
         AFTER
-5898
-5242
-3561
-1089
-103
2340
2627
3111
7465
HIT ANY KEY TO RETURN TO MENU


              SORT DEMO

              SELECTIONS

         1   INTEGER SORT
         2   FLOATING POINT SORT
         3   CHARACTER SORT
         4   EXIT
SELECTION
?REENTER


SELECTION 2
         BEFORE
0
281.379543
659.537768
185.655704
-186.595071
-736.508304
-10.1274439
-77.9707171
352.15675
```

```
        AFTER
-736.508304
-186.595071
-77.9707171
-10.1274439
0
185.655704
281.379543
352.15675
659.537768
HIT ANY KEY TO RETURN TO MENU


              SORT DEMO

              SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT


SELECTION 3
    BEFORE
AAAAAAAADDDDDDDD
CCCCCCCCAAAAAAAA
CCCCCCCCDDDDDDDD
WWWWWWWWYYYYYYYY
YYYYYYYYXXXXXXXX
BBBBBBBBCCCCCCCC
CCCCCCCCCCCCCCCC
CCCCCCCCXXXXXXXX
AAAAAAAAWWWWWWWW
    AFTER
ASCEND  DESCEND
AAAAAAAAWWWWWWWW
AAAAAAAADDDDDDDD
BBBBBBBBCCCCCCCC
CCCCCCCCXXXXXXXX
CCCCCCCCDDDDDDDD
CCCCCCCCCCCCCCCC
CCCCCCCCAAAAAAAA
WWWWWWWWYYYYYYYY
YYYYYYYYXXXXXXXX
HIT ANY KEY TO RETURN TO MENU
              SORT DEMO

              SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT
```

```
SELECTION 1
         BEFORE
2888
6273
-900
-4864
-7349
6889
4183
1853
-4013
         AFTER
-7349
-4864
-4013
-900
1853
2888
4183
6273
6889
HIT ANY KEY TO RETURN TO MENU


              SORT DEMO

              SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT


SELECTION 2
         BEFORE
0
370.781155
264.527624
345.96456
-119.00236
-881.17073
-302.459631
-77.2997615
444.30628
         AFTER
-881.17073
-302.459631
-119.00236
-77.2997615
0
264.527624
345.96456
370.781155
444.30628
```

```
HIT ANY KEY TO RETURN TO MENU


           SORT DEMO

           SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT


SELECTION 3
    BEFORE
WWWWWWWWWWWWWWWW
DDDDDDDDBBBBBBBB
WWWWWWWWYYYYYYYY
DDDDDDDDDDDDDDDD
DDDDDDDDXXXXXXXX
YYYYYYYYBBBBBBBB
WWWWWWWWBBBBBBBB
BBBBBBBBYYYYYYYY
BBBBBBBBBBBBBBBB
```

```
     AFTER
ASCEND  DESCEND
BBBBBBBBYYYYYYYY
BBBBBBBBBBBBBBBB
DDDDDDDDXXXXXXXX
DDDDDDDDDDDDDDDD
DDDDDDDDBBBBBBBB
WWWWWWWWYYYYYYYY
WWWWWWWWWWWWWWWW
WWWWWWWWBBBBBBBB
YYYYYYYYBBBBBBBB
HIT ANY KEY TO RETURN TO MENU


           SORT DEMO

           SELECTIONS

        1   INTEGER SORT
        2   FLOATING POINT SORT
        3   CHARACTER SORT
        4   EXIT
```

OSI Fast Screen Erase under BASIC

William L. Taylor
246 Flora Road
Leavittsburg, OH 4430

When a BASIC program erases the screen by writing blanks, it can take more time to clear the display than to fill it. Speed up that slow poke with this fast machine language approach.

While working on a number of game programs written in BASIC, the need for a faster method of screen clearing for animated characters was a desirable feature that I did not have with the POKE function of BASIC. The usual method is to set the desired number of lines to be cleared and POKE the ASCII equivalent for a blank out to the screen. This gives a slow, line-by-line screen clearing effect that is not acceptable with fast games using animated characters. The screen clear routine must be ultra-fast for this type of game program.

The following subroutine will work with most BASIC programs that require a fast screen clear. The routine is written in BASIC and assembly language. The ultra-fast screen erase portion is in assembly object code and is placed in user memory. It can be used with programs written in OSI MicroSoft BASIC for the OSI computer systems.

My system is composed of the system boards sold by Ohio Scientific Instruments. The CPU board is a Model 500 with the 8K OSI BASIC by MicroSoft. The display board is a Model 440 with 4 pages of screen memory and alphanumerics only. My system has 8K of read-write memory on two 420C memory boards, along with a 430A Super I/O board for the audio cassette interface.

The program is a subroutine that uses BASIC as a housekeeper to count the number of pages to be cleared. The actual work is done in the machine code routine that is called by the mainline BASIC program. This program can be set up as a subroutine and called from your mainline when a screen erase is required.

At line 10, the variable D contains the initial location for the machine code routine that performs the store-to-screen function. This is the location at the beginning of the screen memory. The screen memory begins at hex D000, or 53213 decimal, on the 440 and the 540 OSI display boards.

Line 20 defines the USR vector and sets the vector point to hex 0F000, or 3840 decimal, where the machine code routine is located. Line 30 causes a jump to the user vector located at hex 0A, 0B, and 0C in page zero of the user memory.

The machine code routine will execute and one page of screen memory will be cleared. Line 40 updates the page count by changing the machine code routine at location 0F08, or 3848 decimal. At line 50, the page pointer is incremented by increasing variable D by 1.

Lines 60 and 70 check to see whether all pages, or all screen locations have been cleared. If they have not (variable D not equal to 213 or 217) then another loop will be forced until all pages of screen memory have been cleared. Line 70 should be a return, if called as a subroutine: 70 IF D = 213 THEN RETURN for a 440 display board, and 70 IF D = 217 THEN RETURN for a 540 display board.

The loading of machine code into user memory can be performed by storing the machine code in DATA statments. Then the user location is defined and the data is read and POKEd into user memory. An example of this method is found in the subroutine at lines 100 through 150.

A word of caution may be in order at this point. The memory size must be set when bringing up BASIC. That is, before loading your program you must set the size of memory to protect the machine code routine. Set the memory size to 3839 decimal, for this routine, to prevent BASIC from destroying your machine code.

```
 10 D=208
 20 POKE 11,00: POKE 12,15
 30 X=USR(X)
 40 POKE 3848,D
 50 D=D+1
 60 IF D<213 THEN 30
 70 IF D=213 THEN RETURN



100 FOR R=3840 TO 3853
110 READ M: POKE R,M
120 NEXT R
130 DATA 162,0,232,169,32,234
140 DATA 157,0,208,224,255,208,245,96
150 RETURN
```

THE MICRO SOFTWARE CATALOG: X

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

Name: **DISK TEXT EDITOR**
System: **Apple II**
Memory: **Minimum of 24K with DOS & Applesoft ROM**
Language: **Applesoft II BASIC**
Hardware: **Apple II, Disk II, optional Applesoft ROM & printer.**
Description: EDIT is a DOS Text Editor designed to facilitate changes to disk files, but also supporting input and output via cassette. The text editor will operate on fixed or variable length disk records and has 27 commands. System commands allow the user to DELETE, INSERT, CHANGE, DISPLAY, ADD, and PRINT records. String commands, such as STRING CHANGE and SEARCH, find and change a single character string or the entire file. User defined TABS, file APPEND, and CONCATENTATION, file creation, and other manipulations are also provided to modify text from the keyboard or existing files.
Copies: **Just released**
Price: Cassette **$16.95**
Diskette **$21.95** (specify Applesoft ROM)
Shipping **$1.25**
Includes: User manual and documentation
Author: **Robert A. Stein, Jr.**
Available from:
Services Unique, Inc.
2441 Rolling View Dr.
Dayton, Ohio 45431

Name: **AMATEUR RADIO LOG PROGRAM**
System: **APPLE II**
Memory: **8K**
Language: **Applesoft II**
Hardware: **Apple II, cassette tape recorder**
Description: This program provides a computerized record of an amateur radio operator's log book.
There are seven functions:

1. Add log entries
2. Print log entries by date.
3. Print log entries by call letters.
4. Print log entries by entering only first 3 digits of call letters and/or entering only call area or district or call sign.
5. Print all log entries.
6. Print names of places (cities, states, counties, countries, etc.) or other info that you enter.
7. Print log entries by entering only the QTH.

Data is printed in for form of:
Date: Time: Call: Freq: MODE: QSL: QTH: Name:
The program is very useful for QSO's, contests, DX, awards, QSLing, QTHs, names.
All of the above questions will be answered after you enter your data and other information.
Copies: **Just released** (at least 10 copies have been sold)
Price: **$12.00**
Includes: Cassette, sample run and instructions to revise.
Author: **Alex Massimo**
Available from:
Alex Massimo — A F 6 W
4041 41st Street
San Diego, CA 92105

Name: **Programmer's Utility Pack**
System: **Apple II**
Memory: **4K to 6K depending on the program used**
Language: **Integer BASIC and Applesoft**
Hardware: **Apple II with cassettee or disk drive**
Description: Set of 11 programs. Appends, STR$ () and VAL () are on printed documentation with the tape version. Programs include: Renumber-Integer & Applesoft, Append-Integer & Applesoft, Line Find-Integer & Applesoft, Address/Hex Converter, Screen find, Memory Move, and the STR$() and VAL() function simulations for Integer. By using the various programs one can renumber Integer and Applesoft programs with all GOTO's, etc, being renumbered and the user alerted to unusual situations in the program. These include referenced line #'s not in the program, lines referenced by a variable or expression, and a number of others. Line Find allows the user to locate the actual address range of a line in memory so as to be able to insert CLR, HIMEM:, etc. Can also be used on occasion to recover programs garbaged by dropped bits. Address/Hex Converter converts between the Hex, Integer, and Applesoft address formats. It also provides the two byte breakdown of numbers greater than 256 for use in pointers, etc. Screen Find is used for printing directly on the screen by POKEing appropriate values into the proper locations in memory. Screen Find gives these values and locations when the characters desired and the horizontal, vertical screen positions are input. Memory Move allows one to move blocks of memory up or down any number of bytes from Integer or Applesoft. The Monitor has a routine similar to this but it cannot be used to move blocks up a small distance and it is not possible to use it directly from Applesoft. STR$( ) simulates the function of this name in Applesoft for use in Interger programs. STR$() in Applesoft converts a number to a string. VAL() is similar but converts strings to numbers.
Copies sold: **Just released**
Price: **$16.95** Calif. residents add 6% sales tax
Includes: Two cassettes or 1 diskette plus documentation

Author: **Rober Wagner**
Available from:
Local Apple dealers or:
Southerwestern Data Systems
P.O. Box 582-MC
Santee, CA 92071
(714) 562-3670
SASE for info.

Name: **MACRO Assebler/Text Editor**
Systems: **PET, Apple II, SYM**
Memory: **16K system recommended. Program occupies 8K.**
Language: **Assembly**
Hardware: **Terminal and one or two cassette decks.**
Disk may be used in lieu of cassette decks.
Description: Combined assembler and text editor software (2000-3FFF) which has the following features: Marco and Conditional Assembly support; binary, hex and decimal constants; labels up to 10 characters; loads/records and appends from tape; string search and/or replace commands; auto line numbering; copy and more commands; linkage vectors to disks; syntax — similar to MOS Technology specs. Over 25 commands, 22 pseudo ops, and 5 conditional assembly operators.

Copies: **Just released. 25 as of April 1979**
Price: **$35.00** plus $2.00 shipping and handling.
Includes: Manual and either PET, Apple II, or SYM (H.S.) cassette tape. No source.
Order Info: Check or money order.
Author: **Carl Moser**
Available from:
C. W. Moser
3239 Linda Drive
Winston-Salem, N.C. 27106

Name: **Commodity File**
System: **Apple II**
Memory: **32K or more**
Language: **Applesoft II**
Hardware: **Disk II, optional printer**
Description: The program stores and retrieves virtually every commodity traded on all exchanges. A self- prompting (burned-in) program allowing the user to enter open/closed contracts. Figures profits/losses, and maintains a running cash balance. Takes into account any amending of cash balance such as new deposits or withdrawals from account. Instantaneous readouts (CRT or printer) of contracts on file, cash balances, P/L statements. Includes color bar graphs depicting cumulative and individual transactions. Also includes routine to proof-read contracts before filing.
Copies: **Just released**
Price: **$14.95** on diskette, **$9.95** on cassette
Includes: Program cassette or diskette, Complete documentation.
Author: **S. Goldstein**
Available from:
MIND MACHINE, Inc.
31 Woodhollow Lane
Huntington, N.Y.
11743

Name: **METRIC-CALC™**
System: **Commodore PET**
Memory: **8K**
Language: **BASIC**
Hardware: **Pet 2001-8 (or 2001-4 with 4K external memory).** Available as special order for 2001-16 or 2001-32.
Description: METRIC-CALC turns your PET into a powerful stack-operated (RPN) scientific calculator that includes metric conversions. Unlike other metric converters, this one lets you *use* the converted figures in your calculations. Unlike other stack-operated calculators, this one lets you *see* the contents of the stack . . . the top five levels are displayed during calculations, and all twenty can be reviewed at any time (as can the twenty addressable storage locations). Numbers "buried" in the stack can be copied to stack-top with a keypress. Functions include instructions, arithmetic, inversion, logarithms, trigonometry, powers . . . too many to include here. Write for flyer. Reviewed in Spring 79 issues of PET Gasette, and Best of PET Gazette.
Copies: **More than 60 sold**
Price: **$7.95** (quantity discount available)
Includes: Cassette in Norelco style box, description and operating instructions, zip-lock protective package.
Designer: **Roy Busdiecker**
Available from: Better computer stores or directly from
Micro Software Systems
P.O. Box 1442
Woodbridge, VA 22193

Name: **MAZE GAME**
System: **PET 2001**
Memory: **8K**
Language: **PET BASIC**
Hardware: **Standard**
Description: This is a real-time game of skill which tests your co-ordination as you attempt to guide a ball through a maze that is displayed on the screen using the PET graphics. There are four levels of play which grade the speed of the ball and the number of mistakes you can make, from the slow learner speed to the ultra-fast masochist level. The maze is 19 by 11 squares and you have to go from left to right (i.e. the long way).
Copies: **Many**
Price: **$19.95**
Author: **Jeff Law**
Available from:
Southern Software Limited
P.O. Box 8683
Auckland, New Zealand

Name: **Sales Forecasting**
System: **Apple**
Memory: **16K**
Language: **Apple II Soft**
Description: Program displays business forecast from the best fit of four curve fits. Manual operation is optional.
Copies: **30**
Price: **$9.95** + $1.00 postage & handling (PA residents add 6% sales tax)
Includes: Cassette with instructions
Author: **Neil D. Lipson**
Available from:
Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

Name: **Table Generator**
System: **Apple**
Memory: **16K**
Language: **Applesoft II**
Description: A program that forms shape tables with ease. Program adds in other information such as starting address, length and position. Saves all of this information into a useable location in memory.
Copies: **10**
Price: **$9.95** & $1.00 postage & handling (PA residents add 6% sales tax)
Includes: Cassette with instructions
Author: **Murray Summers**
Available from:
Progressive Software
P.O. Box 273
Ply. Mtg., PA 19462

Name: **Restaurant Evaluation**
System: **Apple II**
Memory: **16K**
Language: **Applesoft II**
Hardware: **Disk II (optional)**
Description: Evaluates potential restaurant/nite club sites and thereby reduces the margin of risk involved in purchasing a new or existing business. The program design is of a computer question, user answer nature. The auther has borrowed against his many years of experience in the restaurant business and has built into the program all the necessary percentages to evaluate whether a potential site will be profitable or not. The program calculates monthly gross, computes monthly loan notes (or mortgage) and arrives at a monthly net proft/loss reported in dollar amounts and percentages.
Copies: **Just released**
Price: **$14.95** Diskette, $9.95 cassette + $1.00 Shipping
Author: **M. Goldstein**
Available from:
MIND MACHINE, Inc.
31 Woodhollow Lane
Huntington, NY 11743

Name: **Personal Accounting System—PAS**
System: **PET**
Language: **BASIC**
Hardware: **Single cassette drive or COMPUTHINK disk**
Description: PAS relies heavily on the PET's file capabilities to generate and validate files containing a detailed description of your financial transactions. PAS consists of six programs including those to generate and edit data files, balance your checkbook, reconcile your bank statement, report your outstanding checks and summarize your transactions over a period of time. PAS creates files for monthly transactions, outstanding checks, and summaries.
Includes: Excellent user manual, cassette or disk
Author: **Ronald C. Smith,** SMITHWARE
Copies: **Just released**
Price: Cassette version (8K), **$19.95;** disk version, **$24.95**
Author: **Ronald C. Smith,** SMITHWARE
Available from:
PROGRAMMA INTERNATIONAL
3400 Wilshire Blvd.
Los Angeles, CA 90010

Name: **SIGNS**
System: **PET 2001**
Memory: **8K**
Language: **PET BASIC** (IEEE port 5)
Hardware: **Printer** (PET or RS-232)
Description: The signs package is intended for producing posters, headings and other signs, in several formats, to be printed on a printer. The package consists of two programs written for 8K PET systems. One program initializes data for the signs program and then the second program requests text for the sign and prints the sign out with three sizes of letter (micro, small and big); left, centre or right justified on tha page, with options to specify foreground and background characters. Other options include NEWPAGE, SPACE n, and END.
Copies: **Many**
Price: **$19.95**
Author: **Terry Teague**
Available from:
Southern Software Limited
P.O. Box 8683
Auckland, New Zealand

Name: **Othello**
System: **6502 SYM-1** bare system
Memory Required: **1K**
Language Used: **6502 Machine Language**
Hardware Required: **None**
Description: The look ahead ply depth is entered through the key board. Player or computer may move first. All sequences of moves are evaluated, with the 2,3,4,5, etc. ply game requiring 1 sec, 8 sec, 1 min, 8 min, etc. respectively per move. Every move, is checked for legality, (beeper sounds if move is invalid) and all moves and number flipped are displayed automatically. Player enters his moves through the keyboard. Ply depth is automatically incremented near the end of the game. For example, in 1 min, the computer plays the last 7 moves perfectly!
Price: **$6.95**
Includes: Cassette (KIM format) and instructions
Author: **David B. Schaechter**
Available from:
David B. Schaechter
4343 Ocean View Blvd. Apt. 261
Montrose, CA 91020

Name: **ALGEBRA**
System: **APPLE II**
Memory: **16K**
Language: **Integer BASIC and Machine Language**
Description: School tested enjoyable algebra programs, using missing words, this interactive program starts the student learning algebra on the high school level.
Copies: **Just released**
Price: **$9.95** for cassette with 2 lessons
Includes: Cassette and loading instructions
Author: **George Earl**
Available from:
George Earl
1302 S. Gen. McMullen
San Antonio, TX 78237

To Tape or Not to Tape: What is the Question?

Noel G. Biles
P.O. Box 1111
San Andreas, CA 95249

Dust off that oscilloscope and clear up some of the mystery behind digital data recording on audio cassette.

These lines are penned in an attempt to clear up some of the mysteries of doing the impossible, and to explain some of the apparent idiosyncrasies of electronics. Some microcomputer operators are neophytes in basic electronics, and so, this little lesson will endeavor to explain what each part is, how it works, and why it is used in a given circuit. I would suggest you try the experiments shown in Figure 3 for a better understanding of the circuit theory.

Those who don't own an oscilloscope, could make one of your club meetings into an evening away from talking about the merits of software or peripherals, and try to understand what you are paying for when you lay out that long green. Of course, remember to invite someone who owns an oscilloscope.

As the title of this episode suggests, we will investigate why such a simple thing as making a tape recording can cause so much discussion. Most computerists have seen a drawing of the electrical signal put out from a Teletype keyboard and have noted the similarity to drawings of an ASCII signal; let's face it, we've got to learn how to handle these fast changes of DC voltage called square waves, obviously a misnomer because we all know that waves are rythmic undulations of matter and therefore can never really be square.

We are told that a square wave is an "instantaneous" change of voltage from one level to another, with both levels maintained without variation until the next change of state. For TTL circuits these levels are approximately plus 4.8V for level 2 and plus 0.2V for level 1, usually just called 5V for a "1" and zero V for a "0".

I hinted that I was going to talk about the tape recording of digital signals, and I will. First of all, as Dr. DeJong might say, Earthpeople have not yet invented an audio tape recorder that will record or playback digital signals composed of the classical description of the same, namely, "A series of square waves varying only in frequency or timing but unvarying in amplitude." A Teletype punched paper tape comes very close to the ideal way of making a permanent recording of digital signals and, when played back, will produce digital signals very close to the original; however, the expense of one of these machines puts it beyond the budget of most of us. And besides, where do you store all that paper tape?

Them fellers in Kansas City are pretty smart for flatlanders 'cause they figured out a way to fool a computer into thinking it is receiving square waves when it really ain't, and that's the gist of my story. All your computer wants to receive on the "from tape recorder" line is data to say that this frequency of tone means a "one" and this frequency of tone means a "zero". "Sounds so darn simple" you say, "How come one of us mountain folk never thought of that?" Now if we can just make our computer generate those two tones and put them on the "to tape recorder" line in the correct sequence and time, we will have a system like the boys from Kansas City envisioned.

As we said before, even the best tape recorder cannot record square waves, but that is all our computer can generate, so we must modify these square waves to fool the tape recorder into thinking they are distorted sine waves. Then, when they are played back to the computer, it will modify these distorted sine waves back to square waves which our computer can digest.

Figure 1 shows the "tape out" circuitry of the Synertek VIM-1 microcomputer. Because the tape recorder requires only a few millivolts on its input line, the 5 volt square wave from pin 9 must be reduced to usable proportions by the voltage divider formed by R90, R89, and R88. R90 does double duty in conjunction with C14; it forms a low pass filter which has the effect of slowing down the rise time of the square wave signal from pin 9 to a modified square wave with rounded corners as shown on the schematic, and if the "LO" terminal on this machine is used, some additional "rounding off" of the signal will be accomplished by the added cable capacitance in conjunction with R89.

Now, one important thing is that the recorder input level control must be set so that no overloading of the amplifier stages in the recorder occur (because that drives the transistors in there crazy) but so that a sufficient level is maintained for operating the tape head. Recorders with automatic level control (ALC) are great for this type of service because they don't have any recording level control to adjust.

"Aha!" you say, "My tape recorder is a hi fi unit and will reproduce these distorted sine waves just as recorded, and that is not what my computer wants to see." This is true, but the computer is expecting this type of a signal and is prepared for it, as in Figure 2. The output signals from most cassette tape recorders would be a little further distorted from the passage of semi-square waves through the output transformer, which no longer sees the correct load because we have disconnected the 8 ohm loudspeaker. It reflects this change of load impedance back to the primary, in turn destroying the fidelity of the output stage.

Looking at Figure 2, the schematic of the tape recorder input of the Synertek VIM 1, the recorder will see a load of approximately 270 ohms formed by the series impedance of R128 (100 ohms), C15 (170 ohms @ 2,000 Hz), and CR36,37 (approximately 100 ohms) to ground, less the parallel resistance of C16, R92, and diodes CR28, CR29 through R94 to ground, for a total of 264 ohms. The 0.5 watt or more available from the output of the recorder is capable of driving this load to better than 11 volts, which is now divided down to the correct voltage to drive the op amp "sine to square converter" U26.

This division is accomplished via the impedance of C16 (8,000 ohms @ 2,000 Hz) plus R92 (1,000 ohms) through CR28, CR29 (100 ohms) and R94 (3.3K ohms) to ground. So if we adjust the recorder gain control for approximately 8 volts at the input terminal we should have about 2V of signal at op amp pin 3.

This voltage is more than enough to cause diodes CR28 and 29 to clip the voltage peaks at 1.5V and limit the input to the op amp. With the amplified inverse voltage from pin 7 fed to pin 2 through R96, the signal at pin X on the expansion connector will be a nice clean replica of the near perfect, zero to 5 volt square wave we first generated from U37 in Figure 1. R128, C15 and diode CR37 form an audio voltmeter, while diode CR36 is a recording level indicator illuminated by the rectified voltage from CR37.

Now that we thoroughly understand all of the above, let's prove that this really works. Refer to Figure 3 and construct a simple square wave generator on a Proto board with an oscillator operating at approximately 2,000 Hz and an inverting buffer to simulate the internal generator in the computer. We will need a 4011 Quad Dual Gate Integrated Circuit, 5 resistors, and 2 capacitors to build the generator and divider chain. In addition, we will also require a 5V power supply to operate the unit.

Hook up the power supply and, if there is no smoke, start by connecting the oscilloscope to point X in Figure 3. It should reveal a fairly good square wave approximately 5V in amplitude. With C1 temporarily disconnected, point Y will show the same square wave at approximately 1.5V of amplitude, while point Z shows .036V of square wave.

Reconnect C1 to point Y and note the distortion at this point on the rise and

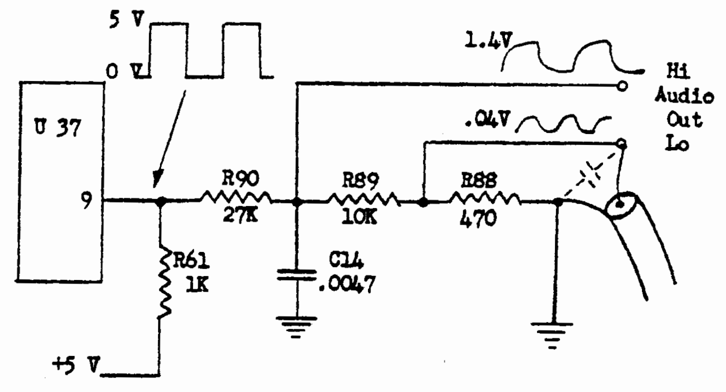


Figure 1

fall times, but not on the amplitude of the square waves. Point Z will be a reduced voltage version of this distorted square wave. Or is it a distorted sine wave?

The frequency chosen for this experiment (2,000 Hz is the center of the two frequencies used on the VIM or SYM microcomputers) will have a direct bearing on the values chosen for R1 and C1. Too large a value for either would reduce the amplitude and shape of the wave we are looking for. Too little value would reduce the rounding off of the rise time.

Try it: add 0.022 mf in parallel with C1 and note the added distortion and reduction in signal strength to near triangular wave at one-half the voltage.

Remove this added capacitor and construct Figure 4 on the Proto board, keeping Figure 3 intact. Now jumper point Y on Figure 3 to "IN" on Figure 4, as per the dotted line. Because the signal at point Y is only 1.2V, diodes CR36 and CR37 cannot conduct, effectively disconnecting R6 and C4 and lightening the load so that point Y does not distort much beyond the original shape prior to addition of the jumper. Checking

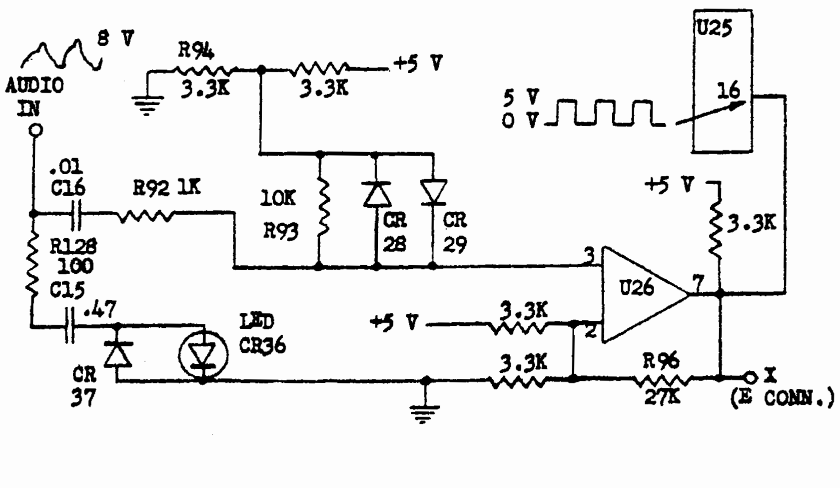


Figure 2

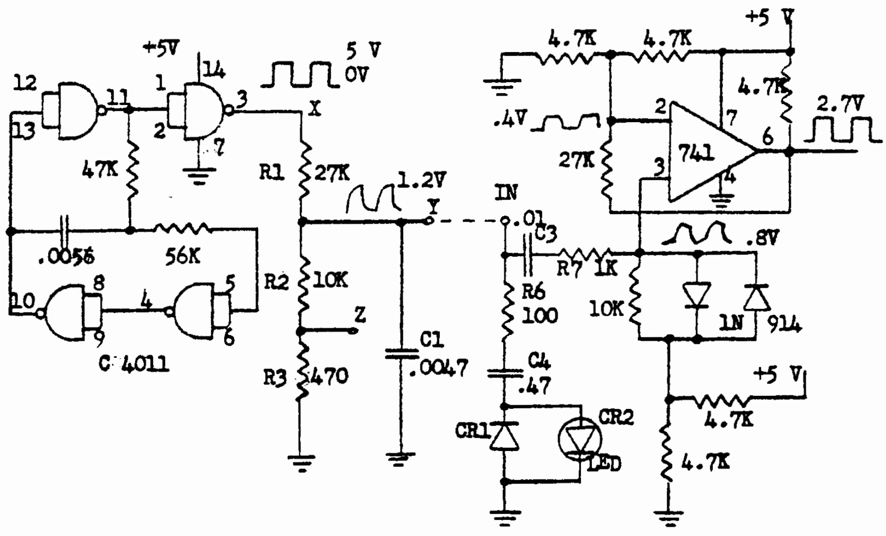


Figure 3

now at pins 2, 3, and 6 should yield signals approximating those shown on the schematic.

Disconnect the jumper from point Y to "IN" and prepare for the big test. Referring to your tape recorder instruction manual, connect a shielded lead from point Z or Y to the mike or auxiliary input and make a five minute recording of the 2,000 Hz signal. Rewind the tape and connect the IN terminal of Figure 4, again with a shielded line, to the monitor or earphone jack on the recorder. Press the PLAY button and adjust the volume control to obtain 6 to 8 volts of signal at the IN terminal. With the oscilloscope connected to pin 6 of the op amp, you should see a fair replica of the square wave you first saw at pin 3 of the 4011 oscillator buffer.

Your scope should have a 10 MHz bandwidth, to observe fast square waves, but any scope will do for these experiments, and that's why I said a "fair replica" of the signal.

All things considered, the design of the VIM 1 cassette interface is more than adequate. When I first fired up my VIM, the only tape I could lay may hands on immediately was a 39 cent, 200 times erasure/rewind tape that my daughter had used to bring home her French language home work. I used this tape to make a Sync tape and record the first few short programs. It still loads every digit without dropouts.

6502 Bibliography: Part XI

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

438. Kilobaud No. 27 (Mar., 1979)

Lindsay, Len "PETpourri", pg. 9-14
PET accessories, Software worth mentioning (renumber, Extended graphics, basic utilities, cassette magazines, etc) cassette maintenance, programming hints.

McFarland, Dr. Ward J. Jr. "The 'El Cheapo' EPROM Programmer", pg. 46-50.
An inexpensive EPROM programmer with software for the 6502.

Ruckdeschel, F.R. "The OSI Model 500", pg. 130-132.
The author concludes that OSI's Model 500 comprises a compromise between completeness and cost.

Carptenter, C.R. (Chuck), "Telpar Thermal Printer", pg. 138-139.
The Apple II is interfaced with a TELPAR PS-40 printer; with software.

439. PET User Notes 1 Iss 6 (Sept./Oct., 1978)

Butterfield, Jim "FOR-NEXT and GOSUB-RETURN Structures", pg. 2.
Clarification of these important commands for PET.

Paul, Grant "Head Alignment for the PET", pg. 2.
Instructions for a simple method of aligning the PET cassette recorder head.

Butterfield, Jim "Disabling the PET Stop Key", pg. 6.
Provides PET with a non-stop feature.

Wilcox, David H. "Index", pg. 7.
A program for PET to find a given program on tape.

Butterfield, Jim "View"
A program for placing an image of one given page of memory onto the screen of PET

Louder, Mike "Dynamic Keyboard Rvisited", pg. 11.
A technique for adding GOTO and GOSUB expressions while a program is running on the PET.

Buttefield, Jim "Cassette File Usage Summary", pg. 14.
Opening files, writing tapes with increased spacing, Closing files, etc.

Group, PET User "Machine Language from Basic", pg. 14.

Anon. "Non-Zero PIVOT ELEMENTS STRATEGY", back cover.
The program finds the inverse of the left hand coeff. matrix and solves for the roots of the linear equation system.

440. Rainbow 1 Iss 1 (Jan., 1979)

Anon. "Basic Music and Sound Effects", pg. 16-17.
Music for the Apple II incorporating Gary Shannon's routines.

441. Southeastern Software Issue 6 (Feb., 1979)

Staff, "Apple Diskettes", pg. 2.
Note on the use of the reverse side of diskettes to provide twice the storage space.

Staff, "Tape Save", pg. 2.
How to use a program TAPE SAVE with the Guil Banks EXEC GEN program from Issue 5. Provides Tape backup for your DISKS.

Staff, "Abbreviated Commands for the Apple DOS", pg. 35.
Change "Catalog" to "C", etc.

Staff, "How to Edit Print Commands Without Introducing Spaces", pg. 5-6.
A great editing aid.

Staff, "All about Call-868 and Call-958", pg. 6.
Explanation and examples.

442. Call · Apple 2 No. 1 (Jan., 1979)

Aldrich, Ron "Disk to Disk Transfer", pg. 3.
Integer Basic program for Apple to transfer programs disk to disk.

Wigginton, R. "Applesoft Chain", pg. 3-6.
A method whereby user programs in Applesoft can chain between programs and retain all variable values.

Finn, Jeffrey K. "Apple Sharing", pg. 8-10.
Standard format options for electronic data transfer, how to modify default settings on the Apple Communications Interface Card, etc.

Golding, Val "High Crimes and How to Commit Them", pg. 12.
How to set HIMEM: within a program; How to create illegal line numbers such as 65535 in Integer Basic. How to execute other illegal commands from within a program such as LOAD, Save, Run, DEL, NEW, etc.

Thyng, Michael "Apple Wash", pg. 12.
How to use the Apple II disk... variables, records and files

Schwartz, Marc "Avoiding End of Disk Error", pg. 18.
Involves use of ONERRGOTO command.

Aldrich, Ron "Disk to Tape transfer Program", pg. 19-20.
An integer basic program.

Aldrich, Ron "Split Catalog", pg. 20-21.
Use this program for your init program and your catalog will list out in two columns on booting disk.

Staff, "Tone Routine", pg. 22.
Routine demonstrates tones by setting variables P and D to A for next loop. Also demonstrates use of &.

443. Applecore Newsletter 1 No 5, (Aug., 1978)

Hertzfeld, Andy "Disk II review", pg. 1.
Transfers data at a rate of 156K bits per second, about 100 times as fast as the cassette interface.

Avelar, Ed "Apple II Multi-Cassette Dumper", pg. 3.
An easy project to save programs from Apple to six or more cassette recorders simultaneously.

Staff, "Apple Beeps Translated", pg. 4.
How to use the Tape beeps to tell how long a program is.

Wyman, Paul "Integer Basic Subroutine for Multiplying Whole Numbers Time a Fraction", pg. 5.
How to use a fraction with Integer basic, on the Apple.

Doty, Jim "String Arrays in Integer Basic", pg. 6.
A simple way to get around the lack of String array capability in Integer Basic in the Apple. Pack two characters into one integer value.

Wyman, Paul "Tale of a Klutzy Tape-Recorder Nurd", pg. 6.
How to recover parts of a program on a damaged tape.

Rainbow 1 No 2 (Feb., 1979)

Simpson, Rick "Introduction to Using HIRES Graphics in Integer Basic", pg. 5-11.
Welcome assistance in understanding HIRES Graphics.

Ellmers, Judd B. "Aligning the READ/WRITE Heads on the Panasonic RQ-309 DS Cassette Recorder", pg. 12-13.
How-to instructions using simple tools.

Staff, "Using the Apple II Mini-Assembler", pg. 19-21.
The Miniassembler is essentially a programming aid in converting a handwritten program to object code.

445. Applecore Newsletter 1 No 8 (Nov., 1978)

Hertzfeld, Andy "DOS—The Name Game", pg. 4.
How to use your own names for DOS commands; output and input "hooks" for the DOS; the advantages of typing 9DB9G from monitor to re-initialize the DOS—said to be safer than the 3DOG technique.

Kamins, Scot "MENU", pg. 5.
An effectual program to allow program choice by number from a disk catalog on the Apple II.

Wells, Arthur "No More 'Catalog' ", pg. 5.
How to make the catalog come up automatically on booting DOS.

Hughes, Tony "Applecore Disk of the Month", pg. 3.
The catalog of the first disk looks very impressive.

Hertzfeld, Tony "Volume MISMATCH matched", pg. 10.
A patch to disable the volume check on the Apple Disk.

Danielson, Larry "Pioneer Hardware Mod", pg. 12.
A modification for those who bought Apples before the color killer modification was put in.

446. Call Apple 2 No 2 (Feb., 1979)

Thyng, Mike "Volume Mismatch", pg. 6.
How to avoid volume mismatch on the Apple DOS.

Aldrich, Darrell "Programming Algorythm", pg. 6.
This is a program for linking routines in the COUT or the KEYIN EXIT when disk is in use on an Apple.

Golding, Val J. "Debugging as a Learning Aid", pg. 10.
Debugging with examples...6502 registers, TRACE, Control D before DOS commands, DSP, etc.

Aldrich, Ron "Disk-Disk Transfer Program", pg. 12.
This program will transfer Integer, Applesoft or Binary listings.

Golding, Val J. "Integer Basic Entry Points", pg. 14.
A program for Integer basic Command Entry Points formatted for Printer or screen.

Golding, Val and Huelsdonk, Bob "Applesoft Program Tokens", pg. 18.
A routine is given to display Applesoft program Tokens.

Golding, Val J. "Convert Catalog to 'C' ", pg. 18.
A routine is given to automatically change DOS commands on the Apple.

Thyng, Mike "Apple Mash", pg. 19.
Discussion of Volume mismatch error, the problem about the Apple DOS not reading or writing to disk if line number is over 255, etc.

Anon, "Apple Source", pg. 20.
DOS Version 3.2 can be expected to be available in March together with a new DOS manual! An UPDATE program will be made available to modify older disks. Pascal on disk and a RAM card will give the Apple 60K of Ram available.

Aldrich, Darrell "Disk Free Space", pg. 20.
A routine to print no of sectors and bytes free on your Apple disk.

447. 6502 User Notes No 13 (Jan., 1979)

Leedom, Robert C. "Kim Hexpawn", pg. 1-5.
Can be played on a 1 K KIM-1.

Butterfield, Jim "6502 OP CODES", pg. 6.
The author has grouped the codes logically so you can see how the codes are classified and decoded.

Tepperman, Dr. Barry "Tape Verify (II)", pg. 7.
Program is located in Kims page two rather than in the VEB as in the case of the earlier version of Verify.

Swank, Joel "Tape File Recovery Routine", pg. 8-9.
How to recover a tape with a dropout. Program for KIM.

Staff, "Language Lab: FOCAL", pg. 10.
Focal for the KIM

Staff, Micro-Z Co "KIM Basic Hint", pg. 11.
Fixes and Modifications for KIM Basic.

Herman, Harvey "Basic Renumber Program", pg. 12.
For those who use Microsoft Basic on KIM.

Day, Michael E. "Two Tiny Basic Mods", pg. 13.
Bugs and Fixes for Tiny Basic.

Rehnke, Eric "Forth", pg. 14.
All about Forth manuals, different types of Forth, etc.

Oliver, John P. "Forth Comments and Example", pg. 14.
Use of Forth on a PET in a telescope pointing program.

Rehnke, Eric C. "A 6522 I/O Board", pg. 16-17.
Room for four of the versatile 6522 PIA's.

Rehnke, Eric "KIM-4 Bus PINOUT", pg. 18.
Definition of the 44pin Standard KIMBUS.

Rehnke, Eric "Video Displays", pg. 19.
Standalone versus Memory Mapped displays are discussed.

Rehnke, Eric "Polymorphic Video Board Mods", pg. 20.
Some modifications before adding this board to the KIM system.

Leedom, Bob "Random comments about KIM and SYM", pg. 22.
Addition of an outboard risistor and A/D assists KIM in games such as ASTEROID. Some Mods are necessary in using KIM programs on the SYM.

Butterfield, Jim "Multi-Mode Adder", pg. 23.
This program adds and subtracts in either decimal or hex.

Zuber, Jim "ASCII Dump Program", pg. 24.
This program will dump ASCII data from memory of KIM to a printer.

Rubens, Thomas J. "Keyboard Debounce Routine", pg. 25.
A fix for noisy KIM keyboards.

Lyon, Douglas "Melodies for the Music Box", pg. 25.
Six new tunes for this popular music program.

Firth, Mike "Camera Speed Tester", pg. 26.
With a minimum of hardware and software timing KIM can time the shutter.

Hawkins, Geo. W. "Power-On Reset", pg. 27.
Very simple hardware for this task.

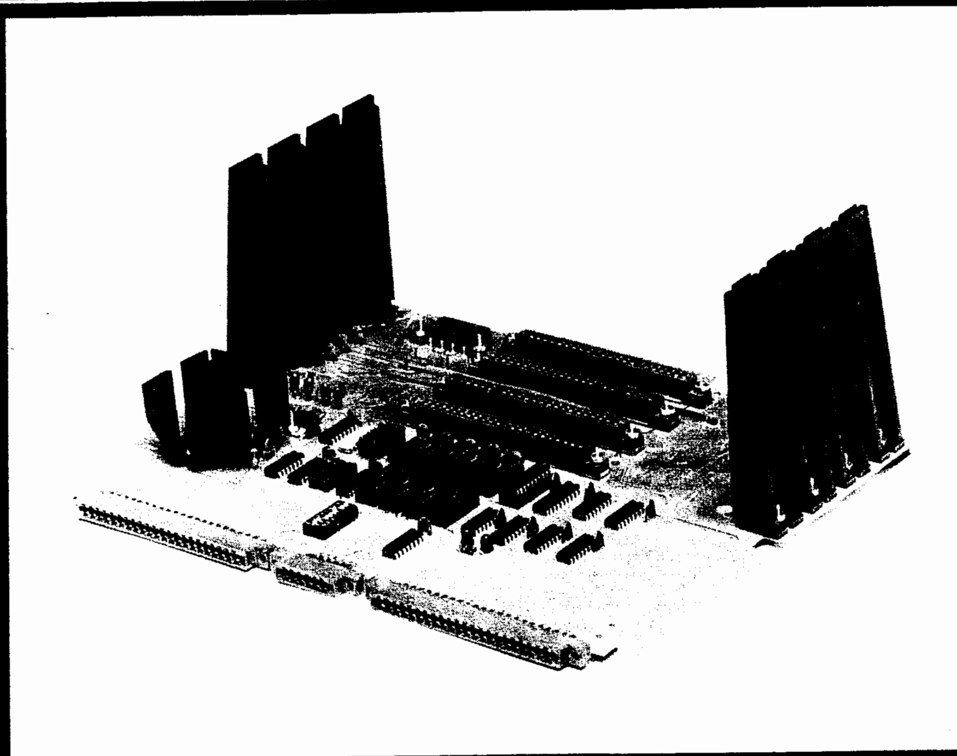
Rehnke, Eric "The Outside World Connection", pg. 27.
Use of OPTO-Isolators in interfaces to the outside world (KIM).

Egbert, Dwight D. "More on the OPTO-Isolator", pg. 27.
KIM-1 to RS232 using opto-isolators.

448. Dr Dobb's Journal 3 Iss 3 No 33 (March 1979)

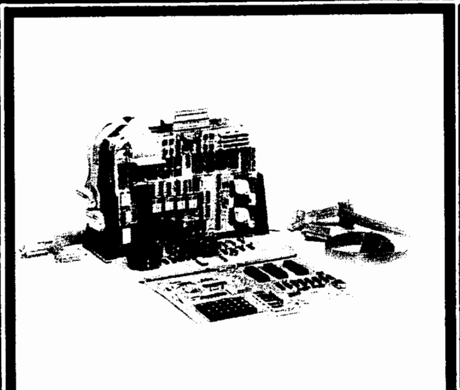
Swank, HJoel "PIA's for KIM", pg. 41-42.
Connect a Motorola 6820 PIA to your KIM.

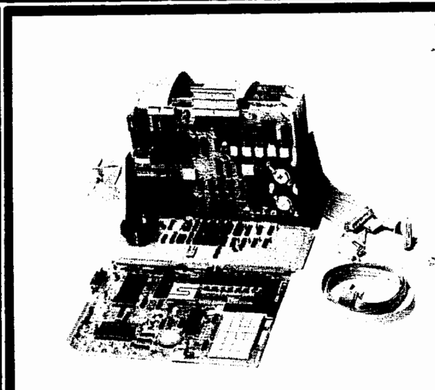
Introducing SEAWELL's

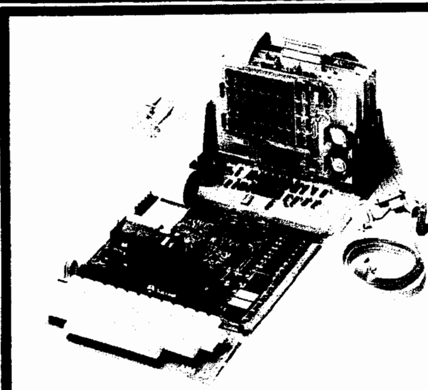


Little Buffered Mother

The ultimate Motherboard for any KIM-1, SYM-1, or AIM-65 system







Features:

- 4K Static RAM on board
- +5V, +12V, and −12V regulators on board
- 4 +1 buffered expansion slots
- Accepts KIM-4 compatible boards
- Full access to application & expansion connector
- LED indicators for IRQ, NMI, and power-on
- Also compatible with SEA-1, SEA-16, the PROMMER, SEA-PROTO, SEA-ISDC, and more
- Onboard hardware for optional use of a 17 (128K addressing limit)
- Mounts like KIM-4 or with CPU board standing up
- 10 slot Motherboard expansion available - SEAWELL's Maxi Mother

Introduction price $99.
Regular price $119.
with 4K Static RAM installed $169.
Assembled — 3 month warranty

For further information contact:

SEAWELL Marketing Inc.
P.O. Box 17006
Seattle, WA 98107

SEAWELL Marketing Inc.
315 N.W. 85th
Seattle, WA 98117
(206) 782-9480

Computer Forum

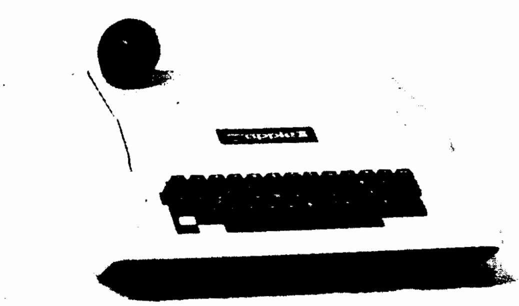
14052 EAST FIRESTONE BOULEVARD
SANTA FE SPRINGS, CALIFORNIA 90670
(213) 921-2111 · (714) 739-0711

· BUSINESS
· EDUCATIONAL
· PERSONAL

APPLE SPECIAL INTEREST GROUP

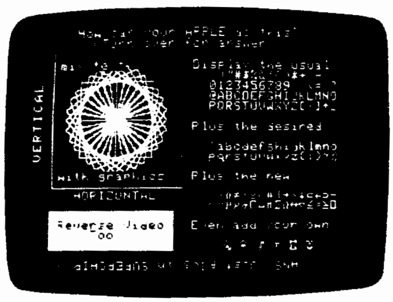
If you have a special interest in using the Apple send us your Name, Address, Phone No. and what your special interest is. We will put you on our special interest mailing list. Each time the list gains enough new members for your special interest, we will send a copy to everyone on the list. This way you will be in contact with others that have the same Apple interest as yourself.

Your special interest could be dress pattern making, RTTY application, NC (numerical control) or anything else that you may want to hear about from others.



List Price	Sale Price
16K $1195.00	$1095.00
32K $1345.00	$1195.00
48K $1495.00	$1295.00
Modulator to connect to color T.V.	$ 29.95

SUPER CHIP

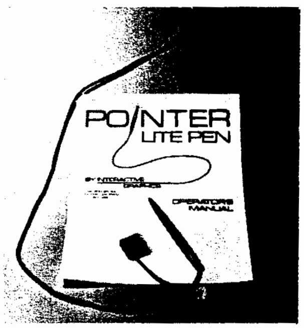


- Full ASCII character set (lower case)
- 31 other, useful non-ASCII characters
- User-defined characters with our Character Edit Cassette
- Enhanced editing capabilities for program and data modification

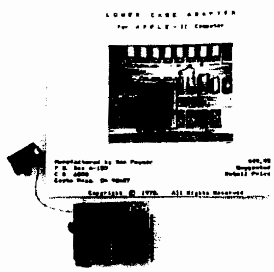
And, it works with your existing APPLE programs—Integer BASIC and Applesoft.

$94.95

List . . . $225.00



$34.95



$49.95

$359.00

Apple II EPROM

Expand Your ROM Software

Add capability to your system monitor or BASIC for business or other applications. Add commands to the operating system. New operating systems can be put into EPROM memory with our EPROM programmer and plugged directly into your APPLE II board with our EPROM socket adaptor. The **MICROPRODUCTS** EPROM Programmer will program INTEL 2716s and other 5-volt replacements for 2716s.

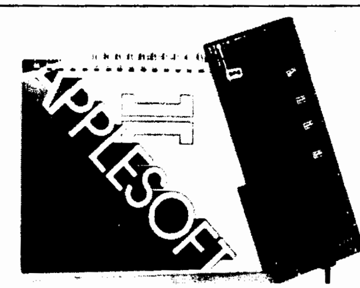
Add to or replace existing APPLE II ROM software with operating systems of your own design. Other software systems similar to PASCAL, FORTH, LISP, APL, FORTRAN, COBOL, ALGOL, other BASICs etc. may be incorporated into your APPLE II ROM space.

The EPROM Programmer looks just like memory to the computer and can be configured to program memory locations from 8000 to FFFF for a total of 32K bytes. This means that the EPROMs can be used in computer applications other than the APPLE II, i.e. the **MICROPRODUCTS** Superkim, etc. This turns your APPLE II into a very low cost powerful, software development system.

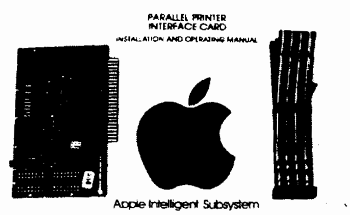
FEATURES:
- Fully assembled
- Completely self-contained
- Textool Zero insertion force socket for EPROM
- Onboard 25 volt power supply
- Double sided plated through holes on fiberglass PC board
- Gold plated edge connector
- Fully socketed
- Latest low-power Schottky IC's
- Solder mask

ADVANTAGES:
- Put memory in two empty ROM slots in APPLE II
- Replace memory in existing APPLE II ROM slots
- Add new operating systems to APPLE II
- Program INTEL 2716 2K byte EPROMs, 2758 1K byte EPROMs and other compatible 5-volt EPROMS.
- Put peripheral drivers in permanent memory
- Use APPLE II to program EPROMs for other computers

$94.95



List . . . $200.00



List . . . $180.00

NAME ____________________
ADDRESS ____________________
CITY ____________________ STATE ____________________
ZIP ____________________ PHONE ____________________

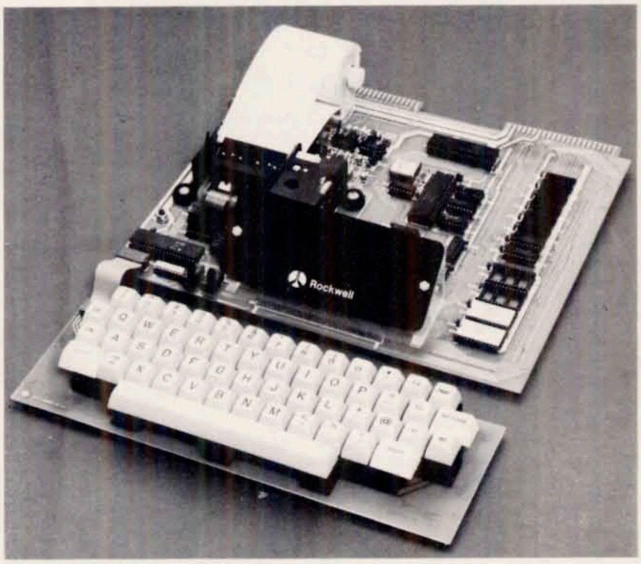
DESCRIPTION	PRICE	QUANTITY	AMOUNT

☐ B of A Card ☐ Mastercharge

Expiration Date ____________________
Card # ____________________
Signature ____________________

Subtotal	
Calif. Residents 6% tax	
Shipping	
TOTAL AMOUNT ENCLOSED	

AIM 65 BY ROCKWELL INTERNATIONAL



AIM 65 is fully assembled, tested and warranted. With the addition of a low cost, readily available power supply, it's ready to start working for you.

AIM 65 features on-board thermal printer and alphanumeric display, and a terminal-style keyboard. It has an addressing capability up to 65K bytes, and comes with a user-dedicated 1K or 4K RAM. Two installed 4K ROMS hold a powerful Advanced Interface Monitor program, and three spare sockets are included to expand on-board ROM or PROM up to 20K bytes.

An Application Connector provides for attaching a TTY and one or two audio cassette recorders, and gives external access to the user-dedicated general purpose I/O lines.

Also included as standard are a comprehensive AIM 65 User's Manual, a handy pocket reference card, an R6500 Hardware Manual, an R6500 Programming Manual and an AIM 65 schematic.

AIM 65 is packaged on two compact modules. The circuit module is 12 inches wide and 10 inches long, the keyboard module is 12 inches wide and 4 inches long. They are connected by a detachable cable.

THERMAL PRINTER

Most desired feature on low-cost microcomputer systems . . .

- Wide 20-column printout
- Versatile 5 x 7 dot matrix format
- Complete 64-character ASCII alphanumeric format
- Fast 120 lines per minute
- Quite thermal operation
- Proven reliability

FULL-SIZE ALPHANUMERIC KEYBOARD

Provides compatibility with system terminals . . .

- Standard 54 key, terminal-style layout
- 26 alphabetic characters
- 10 numeric characters
- 22 special characters
- 9 control functions
- 3 user-defined functions

TRUE ALPHANUMERIC DISPLAY

Provides legible and lengthy display . . .

- 20 characters wide
- 16-segment characters
- High contrast monolithic characters
- Complete 64-character ASCII alphanumeric format

PROVEN R6500 MICROCOMPUTER SYSTEM DEVICES

Reliable, high performance NMOS technology . . .

- R6502 Central Processing Unit (CPU), operating at 1 MHz. Has 65K address capability, 13 addressing modes and true index capability. Simple but powerful 56 instructions.
- Read/Write Memory, using R2114 Static RAM devices. Available in 1K byte and 4K byte versions.
- 8K Monitor Program Memory, using R2332 Static ROM devices. Has sockets to accept additional 2332 ROM or 2532 PROM devices, to expand on-board Program memory up to 20K bytes.
- R6532 RAM-Input/Output-Timer (RIOT) combination device. Multipurpose circuit for AIM 65 Monitor functions.
- Two R6522 Versatile Interface Adapter (VIA) devices, which support AIM 65 and user functions. Each VIA has two parallel and one serial 8-bit, bidirectional I/O ports, two 2-bit peripheral handshake control lines and two fully-programmable 16-bit interval timer/event counters.

BUILT-IN EXPANSION CAPABILITY

- 44-Pin Application Connector for peripheral add-ons
- 44-Pin Expansion Connector has full system bus
- Both connectors are KIM-1 compatible

TTY AND AUDIO CASSETTE INTERFACES

Standard interface to low-cost peripherals . . .

- 20 ma. current loop TTY interface
- Interface for two audio cassette recorders
- Two audio cassette formats: ASCII KIM-1 compatible and binary, blocked file assembler compatible

ROM RESIDENT ADVANCED INTERACTIVE MONITOR

Advanced features found only on larger systems . . .

- Monitor-generated prompts
- Single keystroke commands
- Address independent data entry
- Debug aids
- Error messages
- Option and user interface linkage

ADVANCED INTERACTIVE MONITOR COMMANDS

- Major Function Entry
- Instruction Entry and Disassembly
- Display/Alter Registers and Memory
- Manipulate Breakpoints
- Control Instruction/Trace
- Control Peripheral Devices
- Call User-Defined Functions
- Comprehensive Text Editor

LOW COST PLUG-IN ROM OPTIONS

- 4K Assembler—symbolic, two-pass
- 8K BASIC Interpreter

POWER SUPPLY SPECIFICATIONS

- +5 VDC ± 5% regulated @ 2.0 amps (max)
- +24 VDC ±15% unregulated @ 2.5 amps (peak) 0.5 amps average

PRICE: $375.00 (1K RAM)

Plus $4.00 UPS (shipped in U.S. must give ***street*** address), $10 parcel post to APO's, FPO's, Alaska, Hawaii, Canada, $25 air mail to all other countries

We manufacture a complete line of high quality expansion boards. Use reader service card to be added to our mailing list, or U.S. residents send $1.00 (International send $3.00 U.S.) for airmail delivery of our complete catalog.

RNB ENTERPRISES INCORPORATED

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

master charge VISA