MICRO™
on the Apple

Volume 3

INCLUDES DISKETTE

MICRO on the Apple 3

Notice

Preliminary article selection, Ford Cavallari; final article selection, program testing, debugging, and modification, Tim Osborn; technical assistance, Darryl Wright; copyediting, Marjorie Morse.

Cover Design and Graphics, Kate Winter

MICRO on the Apple Series ISSN: 0275-3537
MICRO on the Apple Volume 3 ISBN: 0-938222-08-2
Printed in the United States of America
Printing 10 9 8 7 6 5 4 3 2 1
Floppy disk produced in the United States of America

MICRO on the Apple 3

Edited by the staff of
MICRO, The 6502/6809 Journal

MICRO INK
P.O. Box 6502
Chelmsford, Massachusetts 01824

Contents

Introduction

MICRO Magazine is proud to present the third volume in our successful series, *MICRO on the Apple*. The programs MICRO publishes for the Apple are consistently among the best — programs that do interesting things in interesting ways on one of the best microcomputers in the world. Some of the programs that appear in this volume were originally published in MICRO Magazine; others are being published now for the first time. All have been thoroughly tested and debugged. Tim Osborn, our Apple expert, has spent many hours making sure that these programs are bug-free.

The programs in *MICRO on the Apple*, Volume 3, offer many hours of absorbing instruction and entertainment for every programmer:

- a carefully selected mix of programming aids for Applesoft and machine language
- impressive graphics programs
- invaluable reference articles
- I/O enhancements
- games

Many of these programs, designed to be used as subroutines, speed up execution. Others add features to your Apple. All will improve your own programming knowledge and ability.

1
APPLESOFT AIDS

Applesoft Aids

In this chapter we have included utilities to speed up execution and to help with program development.

"Amper-Search," by Alan Hill, will help speed up the task of searching a string array for a specified character string. An added bonus is the &DEALLOC, which will de-allocate a string or integer array. These two functions together will greatly increase the speed and efficiency of programs that deal with array processing.

"Applesoft Line Finder Routine" by Peter Meyer will give the user a hex dump of any Applesoft program line, allowing him to insert otherwise unavailable characters into the program text. "Variable Lister" could prove invaluable to someone maintaining a complex program; it dumps all variable values at any point requested without disturbing normal program execution. An added bonus is a 6502 assembly version of the famous Shell-Metzner sort.

Applesoft Line Finder Routine

by Peter J.G. Meyer

This 55-byte machine-language program will display the bytes constituting a specified line in an Applesoft program. This program also demonstrates how you can use the subroutines available in Applesoft and the Apple Monitor.

The Applesoft Interpreter (at \$D000-\$F7FF) and the Apple Monitor (\$F800-\$FFFF) contain many useful machine-language subroutines. One such subroutine, FNDLIN (at \$D61A), finds the location in memory of a given line of an Applesoft program.

To see why you might wish to do this, consider the following simple problem: how do you print "APPLE][PLUS" from within a program? This is easily reduced to two simpler problems: how to print "]" and "["? The former is available on the Apple keyboard in the guise of shift-M, but you cannot enter the latter from the keyboard. A solution is to include in your Applesoft program the line PRINT "APPLE]Z PLUS", and then replace the hexadecimal number which represents 'Z' (namely, \$5A) with the number which represents '[' (namely, \$5B). This requires examination of the region of memory containing the tokenized form of the PRINT statement, locating the \$5A, and replacing it with \$5B. In the case of an Applesoft program composed of only a few lines, this can be done by direct inspection of memory using the Monitor. But, if your program has hundreds of lines, then another method is called for.

Listing 1 is a short, machine-language program which is invoked (from BASIC command mode) by a statement of the form

```
CALL LOCATION, LINE
```

where LOCATION is the location (in decimal) of the machine-language routine (it is relocatable), and LINE is the number of the line in the program to be searched for. If the routine finds the line, then it will display the bytes constituting the line and leave you in Monitor mode. (To return to BASIC command mode, enter Control-C.) If there is no line of the specified number in the Applesoft program, then the only result is a beep.

Suppose the routine is loaded or assembled at $300 (decimal 768), your Applesoft program is in RAM, and you wish to find the location of line 3370, which is, say, PRINT "]X". If you enter CALL 768,3370 then the bytes constituting the line will be displayed as follows:

```
xxxx- yy zz 2A 0D BA 22 5D 5A 22 00
```

where xxxx is the address of the start of the line, yy zz is the pointer to the beginning of the next line (low-byte first), 2A 0D is the line number in hexadecimal (low-byte first), and 00 is the end-of-line token. The remaining five bytes are the tokenized form of the statement PRINT "]Z" (PRINT is represented by one byte: BA). If, for example, the address of the line is $1A92 then (from Monitor mode) you can enter:

```
1A99: 5B
```

which has the effect of replacing the byte '5A' with the byte '5B'. If (after Control-C-ing back to BASIC) the line is then LISTed, it will appear as PRINT "][", and will print accordingly.

For those readers without assemblers, the routine may be entered from Monitor mode by typing in 300: 20 BE DE 20 OC .... (See listing 1 for the remaining bytes.) Once entered, it may be saved to disk by entering BSAVE LINE FINDER, A$300, L$37. To use it, BLOAD LINE FINDER and proceed as above.

Apart from the utility, this routine is interesting because it relies almost entirely on subroutines in the Applesoft Interpreter and the Monitor, which is why it is only 55 bytes long. The five Applesoft subroutines and three Monitor subroutines which are used are given in listing 1 along with their addresses.

The routine works as follows: after you enter CALL 768,3370, this statement is placed in the buffer (at $200) and the zero page pointer TXTPTR is set to the first byte (the token for CALL). Upon invocation of the routine at location 768, TXTPTR is pointing to the comma, and the subroutine CHKCOM checks for this. (If there is no comma, a syntax error message results.) The routine then gets the line number using the subroutine LINGET, and places this (in hexadecimal form, low byte first) at LINNUM. The subroutine FNDLIN picks up this number and searches the Applesoft program for the line so numbered. If it does not find such a line, it returns with the carry flag clear. In this case the routine sounds the bell and returns to BASIC command mode.

If FNDLIN finds the line, then it returns with the carry flag set. It then deposits the address of the line at LOWTR (low byte first, as usual). The routine stores this address at A1, for later use by the subroutine XAM (eXAMine memory), which will display the bytes constituting the line.

Having found the address of the beginning of the line, the subroutines REMN and ADDON are used to find the address of the end. To use the subroutine REMN, which searches from the byte pointed to by TXTPTR until it finds an end-of-line token (00), the routine first sets TXTPTR to four places past the beginning of the line. This action skips the link pointer and the line number, since the line number may contain 00 (as in 0A 00, representing 10), which would mislead REMN. REMN is then invoked, and returns with the offset to the end-of-line in the Y register. ADDON adds this offset to TXTPTR, so that TXTPTR is then pointing to the end of the line. This address is stored at A2, and XAM is invoked to display the bytes from A1 to A2.

```
0800              ;************************
0800              ;*                      *
0800              ;*      LINE FINDER     *
0800              ;*          BY          *
0800              ;*      PETER MEYER     *
0800              ;*                      *
0800              ;*  COPYRIGHT (C) 1982  *
0800              ;*    MICRO INK, INC.   *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                      *
0800              ;************************
0800              ;
0800              ;
0800              ;
0300                       ORG $300          ;RELOCATABLE
0300                       OBJ $800
0300              ;
0300              ;
0300              ;
0300              ;APPLESOFT SUBROUTINES
0300              ;
0300              ;
D61A              FNDLIN   EQU $D61A
D998              ADDON    EQU $D998
D9A6              REMN     EQU $D9A6
DA0C              LINGET   EQU $DA0C
DEBE              CHKCOM   EQU $DEBE
0300              ;
0300              ;
0300              ;MONITOR SUBROUTINES
0300              ;
0300              ;
FDB3              XAM      EQU $FDB3
FF3A              BELL     EQU $FF3A
FF69              MONZ     EQU $FF69
0300              ;
0300              ;
0300              ;ZERO PAGE LOCATIONS
0300              ;
0300              ;
003C              A1       EPZ $3C
003E              A2       EPZ $3E
0050              LINNUM   EPZ $50
009B              LOWTR    EPZ $9B
00B8              TXTPTR   EPZ $B8
0300              ;
0300              ;
0300 20 BE DE              JSR CHKCOM        ;CHECK FOR COMMA
0303 20 0C DA              JSR LINGET        ;GET LINE NUMBER
0306 20 1A D6              JSR FNDLIN        ;SEARCH FOR LINE IN BASIC PROG
RAM
0309 B0 03                 BCS FOUND
030B 4C 3A FF              JMP BELL          ;NOT FOUND
030E A5 9B        FOUND    LDA LOWTR         ;STORE STARTING ADDRESS AT A1
0310 A4 9C                 LDY LOWTR+1
0312 85 3C                 STA A1
0314 84 3D                 STY A1+1
0316 A5 9B                 LDA LOWTR         ;SET TXTPTR TO STARTING
0318 18                    CLC               ;ADDRESS+4
0319 69 04                 ADC #$04
031B 85 B8                 STA TXTPTR
031D A5 9C                 LDA LOWTR+1
031F 69 00                 ADC #$00
0321 85 B9                 STA TXTPTR+1
0323 20 A6 D9              JSR REMN          ;FIND END OF LINE
0326 20 98 D9              JSR ADDON         ;SET TXTPTR TO END OF LINE
0329 A5 B8                 LDA TXTPTR
032B A4 B9                 LDY TXTPTR+1
032D 85 3E                 STA A2            ;STORE ENDING ADDRESS AT A2
032F 84 3F                 STY A2+1
0331 20 B3 FD              JSR XAM           ;DISPLAY MEMORY FROM A1 TO A2
0334 4C 69 FF              JMP MONZ          ;ENTER MONITOR MODE
0337                       END
```

Amper-Search

by Alan G. Hill

High speed machine language search routine finds character strings in BASIC arrays.

Amper-Search is a high-speed character search routine that will find and return the subscripts of all occurrences of a specified character string in a target string array. A search of a 2000 element array will take less than 1 second compared to about 90 seconds for an equivalent BASIC routine. Parameters are used to name the target string array, define the character string, define the bounds of the search, and name the variables to receive the subscripts and number of matches. An added bonus in the Amper-Search code is another routine called &DEALLOC. This routine gives your BASIC program the ability to de-allocate a string array or integer array when it's no longer needed. &DEALLOC can be used with any Applesoft BASIC program.

Let's look at the parameters and how they are passed between the Applesoft program and Amper-Search. The general form is:

```
&S[EARCH](NA$,L,H,ST$,PL,PH,I%,N%)
```

where:

[ ] bracket optional characters. The "&S" are required characters.

NA$ is the variable name of the single-dimensional string array to be searched.

L is a variable, constant, or expression specifying the value of the subscript of NA$ where the search is to begin; i.e. NA$(L).

H is a variable, constant, or expression specifying the value of the subscript of NA$ where the search is to end; i.e. NA$(H).

ST$ is the variable name of the simple string containing the "search" characters. A special case exists if the string contains a Control N character. See note 1.

PL is a variable, constant, or expression specifying the character position in the NA$(I) string where the search is to begin.

PH is a variable, constant, or expression specifying the character position in the NA$(I) string where the search is to end. PL and PH are equivalent to the MID$ statement of the form: MID$(NA$(I), PL,PH – PL + 1).

I% is the name of the single-dimensional integer array into which the subscripts of NA$ will be placed when a "match" is found. The first occurrence will be placed in I%(0). A special case exists if I% is a simple variable rather than an array variable. See note 5.

N% is the name of the simple integer variable into which the number of "matches" will be placed by Amper-Search. N% should be set to zero each time before Amper-Search is invoked. Setting N% <0 is a special case. See note 6.

After Amper-Search is invoked, the elements of NA$ which match the ST$ string may be listed with the statement: FOR I=0 TO N% – 1: PRINT NA$(I%(I)): NEXT I.

Notes

1. A match is defined as the consecutive occurrence of all characters in ST$ with those in NA$(L) through NA$(H) and within the PL and PH character positions of NA$(I). A Control N character in the ST$ string is a wild card. It will match any character in its corresponding NA$(I) position.

2. Any valid variable name may be used as a parameter. An "=" will match anything.

3. 0≤L≤H≤maximum number of elements in NA$. Elements of NA$ can be null strings.

4. 1≤PL≤PH≤255. A PH>LEN (NA$(I)) is allowed and will ensure that the entire NA$(I) string is searched.

5. I% must be dimensioned large enough to hold all matches; i.e. DIM I%(N%). Since you don't know the number of matches before Amper-Search is invoked, you have two alternatives. I% can be dimensioned the same size as NA$, thus assuring enough space to accommodate a complete match. This may waste memory or require more memory than is available. A second alternative is to first define I% as a *simple* variable before invoking Amper-Search. In this special case, Amper-Search will return the number of matches *only*. Your program can then DIM I%(N%), set N%=0, and re-invoke Amper-Search to return the subscripts. Its speed makes this option practical even for large arrays and will conserve memory by not allocating unused I% elements.
6. N% should be ≤0 prior to invoking Amper-Search. Set N% =0 if you want all matches. If N% =0 upon return, there were no matches. Set N% = – 1 if you only want the *first* occurrence of a match. In this special case, N% will be – 1 if there were no matches, or + 1 if a match were found. The subscript of the matching NA$ element will be found in I%(0).

Note 5 described a method for allocating the minimum size for I% that is large enough to hold the maximum number of matches. You could ask, "What if I use &SEARCH iteratively with a different ST$ string each time that has more matches than I% can hold? Won't that cause a BAD SUBSCRIPT ERROR?" Yes it will. Ideally, you would like to de-allocate I% and re-DIMension it at the new minimum size. The CLEAR command won't do the job because it will clear all variables. Now you should see the utility of yet another Amper-library routine called &DEALLOC which performs the needed function. The general form is:

```
&D[EALLOC] (A,B,N)
```

where A,B,N are the named variables of the integer and string arrays to be de-allocated.

[ ] bracket optional characters. "&D" are required.

For example: &D(I%) will de-allocate the I% integer array, &D(XY$,K%) will de-allocate the XY$ string array and the K% integer array.

To complete the de-allocation process, your program must follow the &D(XY$) statement with an X=FRE(0) housekeeping statement to regain the memory from character strings referred to only by the de-allocated string array. &DEALLOC cannot be used to increase the size of an array while preserving the current contents of the array.

Now let's look at some simple examples created by running the program in listing 1.

Listing 2 is a general BASIC demo you can experiment with to learn how Amper-Search can be used.

Some of the routines in Amper-Search can be adapted for use in other Amper-library machine language routines. The following routines may be useful:

GNAME retrieves the string or integer variable name from the "&" parameter list and places it in the NAME buffer in your machine language program. The A register is returned with a "$" or "%" character.

INTE converts the positive ASCII variable name in NAME to Applesoft's 2-character negative ASCII naming convention for integer variable names. If the A register does not contain a "%" upon entry, the carry flag will be set upon return.

STRING performs the same function for string variable names as INTE does for integer variables. The A register must contain a "$" upon entry.

FARRAY will search variable space for the array variable name contained in the NAME buffer. If found, its address will be returned in the X and Y registers. If not found, the carry flag will be set.

FSIMPL performs the same function for simple variables as FARRAY does for array variables.

&DEALLOC also uses several of the above routines. Similar routines which can be adapted reside somewhere in the Applesoft interpreter.

```
1  REM  ************************
2  REM  *                      *
3  REM  *     AMPER-SEARCH1    *
4  REM  *     ALAN G. HILL     *
5  REM  *                      *
6  REM  *  COPYRIGHT (C) 1982  *
7  REM  *     MICRO INK, INC.  *
8  REM  * CHELMSFORD, MA 01824 *
9  REM  *  ALL RIGHTS RESERVED *
10  REM *                      *
11  REM ************************
12  REM
13  REM
14  HIMEM: 9 * 4096 + 2 * 256
15 D$ =  CHR$ (4): PRINT D$"NOMONIC,I,O"
16  PRINT D$"BLOAD B.AMPER-SEARCH(48K)"
17  POKE 1013,76: POKE 1014,0: POKE 1015,146: REM 3F5:JMP $9200
18  DIM NA$(10),I%(10)
20 NA$(0) = "APPLE CORE"
21 NA$(1) = "CRAB APPLE"
22 NA$(2) = "APPLE&ORANGE"
23 NA$(3) = "APPLE/ORANGE"
24  LIST 18,23
100  REM  FIND ALL OCCURRENCES OF 'APPLE'
101 N% = 0:ST$ = "APPLE"
102  & SEARCH(NA$,0,10,ST$,1,255,I%,N%)
103  LIST 100,102: GOSUB 2000: GOSUB 3000
200  REM  FIND 'APPLE' IN NA$(0)->NA$(1) COLUMNS 1->5
201 N% = 0:ST$ = "APPLE"
202  & SEARCH(NA$,0,1,ST$,1,5,I%,N%)
203  LIST 200,202: GOSUB 2000: GOSUB 3000
300  REM  FIND 'APPLE ORANGE'
301 N% = 0:ST$ = "APPLE" +  CHR$ (14) + "ORANGE"
302  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)
303  LIST 300,302: GOSUB 2000: GOSUB 3000
400  REM  FIND 1ST 'ORANGE'
401 N% =  - 1:ST$ = "ORANGE"
402  & SEARCH(NA$,0,3,ST$,1,255,I%,N%)
403  LIST 400,402: GOSUB 2000: GOSUB 3000
490 ST$ = "CRAB"
492  REM  DYNAMICALLY ALLOCATE/DEALLOCATE M%
495  FOR J = 1 TO 2
500 N% = 0:K% = 0
501  & SEARCH(NA$,0,3,ST$,1,255,K%,N%)
502  DIM M%(N%):N% = 0
503  & SEARCH(NA$,0,3,ST$,1,255,M%,N%)
504  LIST 490,530: GOSUB 2100: GOSUB 3000
510  & DEALLOC(M%)
520 ST$ = "APPLE"
530  NEXT J
600  REM  FIND 'E' IN COLUMN 10
601 N% = 0:ST$ = "E"
602  & SEARCH(NA$,0,3,ST$,10,10,I%,N%)
603  LIST 600,602: GOSUB 2000
700  END
2000  IF N% = 0 THEN  PRINT "NONE FOUND": RETURN
2005  FOR I = 0 TO N% - 1
2010  HTAB 4: PRINT NA$(I%(I))
2020  NEXT I
2030  PRINT : RETURN
2100  IF N% = 0 THEN  PRINT "NONE FOUND": RETURN
2105  PRINT
2110  FOR I = 0 TO N% - 1
2120  HTAB 4: PRINT NA$(M%(I))
2130  NEXT I
2140  PRINT : RETURN
3000  FOR I = 1 TO 5000: NEXT I: RETURN

]
```

```
1  REM  ************************
2  REM  *                      *
3  REM  *     AMPER-SEARCH2    *
4  REM  *      ALAN G. HILL    *
5  REM  *                      *
6  REM  *  COPYRIGHT (C) 1982  *
7  REM  *     MICRO INK, INC.  *
8  REM  * CHELMSFORD, MA 01824 *
9  REM  *  ALL RIGHTS RESERVED *
10  REM *                      *
11  REM ************************
12  REM
13  REM
1000  GOSUB 10000
1010  POKE 32,20: POKE 33,19: HOME : VTAB 5: PRINT "DO YOU WANT TO": PRINT
     "SPECIFY SEARCH": PRINT "LIMITS(Y/N)?";: GET A$: PRINT
1020  IF A$ <  > "Y" THEN 1080
1030  VTAB 10: CALL  - 868: INPUT "LOWER SUBSCRIPT:";L: IF L < 0 OR L > 2
     1 THEN  PRINT B$: GOTO 1030
1040  VTAB 12: CALL  - 868: INPUT "UPPER SUBSCRIPT:";H: IF H < 0 OR H > 2
     1 OR H < L THEN  PRINT B$: GOTO 1040
1050  VTAB 14: CALL  - 868: INPUT "LOWER COLUMN:";PL: IF PL < 1 OR PL > 2
     55 THEN  PRINT B$: GOTO 1050
1060  VTAB 16: CALL  - 868: INPUT "UPPER COLUMN:";PH: IF PH < 1 OR PH > 2
     55 OR PH < PL THEN  PRINT B$: GOTO 1060
1065  VTAB 18: CALL  - 868: PRINT "FIRST/ALL?";: GET A$: PRINT : IF A$ =
     "F" THEN F% =  - 1
1070  GOTO 1120
1080 L = 0: REM  START AT NA$(0)
1090 H = I: REM  SEARCH ALL
1100 PL = 1: REM  START WITH 1ST COLUMN
1110 PH = 255: REM  MAXIMUM COLUMNS
1115 F% = 0: REM  FIND ALL
1120  POKE 32,0: POKE 33,39: VTAB 23: CALL  - 868
1130  INVERSE : PRINT "STRING:";: NORMAL : INPUT " ";ST$
1140  IF  LEN (ST$) = 0 THEN  END
1150 N% = F%: REM  INIT COUNTER
1160  REM  INVOKE 'AMPER-SEARCH'
1170  & SEARCH(NA$,L,H,ST$,PL,PH,I%,N%)
1180  REM  LIST FOUND STRINGS
1190  POKE 32,20: POKE 33,19: HOME
1200  IF N% <  = 0 THEN  PRINT "NONE FOUND": GOTO 1120
1210  FOR I = 0 TO N% - 1
1220  VTAB I%(I) + 1: PRINT NA$(I%(I))
1230  NEXT I
1240  GOTO 1120
10000  REM  HOUSEKEEPING
10010  HIMEM: 9 * 4096 + 2 * 256
10015  POKE 235,0
10020 D$ =  CHR$ (4)
10030 B$ =  CHR$ (7)
10040  PRINT D$"NOMONIC,I,O"
10050  POKE 1013,76: POKE 1014,0: POKE 1015,146: REM  SETUP '&' VECTOR AT
      $3F5 TO JMP $9200
10060  TEXT : HOME : VTAB 10: HTAB 12: PRINT "AMPER-SEARCH DEMO"
10070  HTAB 19: PRINT "BY": HTAB 14: PRINT "ALAN G. HILL"
10080  PRINT D$"BLOAD B.AMPER-SEARCH(48K)"
10090  FOR I = 1 TO 1000: NEXT I
10100  DIM NA$(22),I%(22)
10110 I = 0
10120  REM  INITIALIZE STRING ARRAY
10130  READ NA$(I)
10140  IF NA$(I) = "END" THEN 10160
10150 I = I + 1: GOTO 10130
10160 I = I - 1
10170  HOME
10180  FOR K = 0 TO I
10190  PRINT K; TAB( 4);NA$(K)
10200  NEXT K
```

```
10210  RETURN
11000  REM  SAMPLE STRINGS
11010  REM  NOTE: THIS DEMO IS SCREEN ORIENTED. DON'T PUT MORE THAN 22 IT
     EMS IN THE DATA STATEMENT LIST.
11020  DATA  APPLE II,APPLE SIDER,APPLE CIDER,APPLEVENTION,APPLE PI,APPLE
     SAUCE,APPLE TREE,APPLE ORCHARD
11030  DATA   APPLE II PLUS, APPLES & ORANGES ,APPLE BLOSSOM,CANDIED APPL
     ES,APPLE/ORANGE,APPLESOFT,APPLEODIAN,APPLEVISION
11040  DATA  APPLE STEM,APPLE CORE,APPLE-A-DAY,APPLE PIE,APPLE PEEL,APPLE
     -OF-MY-EYE
11050  DATA  END
```

```
0800              ;************************
0800              ;*                      *
0800              ;*     AMPER-SEARCH     *
0800              ;*          BY          *
0800              ;*     ALAN G. HILL     *
0800              ;*                      *
0800              ;*  COPYRIGHT (C) 1982  *
0800              ;*    MICRO INK, INC.   *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                      *
0800              ;************************
0800              ;
0800              ;
00D0              NAPTR    EPZ $D0
00D2              SAPTR    EPZ $D2
00D4              JAPTR    EPZ $D4
00D6              NPT      EPZ $D6
00D8              L        EPZ $D8
00DA              H        EPZ $DA
00DC              PL       EPZ $DC
00DD              PH       EPZ $DD
00DE              TEM6X    EPZ $DE
00E0              NAPTH    EPZ $E0
00E2              CNAPTR   EPZ $E2
00E4              CSAPTR   EPZ $E4
00E6              SAVEY    EPZ $E6
00E7              PS       EPZ $E7
00E8              LENNA    EPZ $E8
00E9              LENSA    EPZ $E9
00EA              SWITCH   EPZ $EA
00EB              SIZE     EPZ $EB
00D2              OFFSET   EPZ $D2
00D4              A1       EPZ $D4
0050              Z50      EPZ $50
00B7              CHRGOT   EPZ $B7
00B1              CHRGET   EPZ $B1
FDED              COUT     EQU $FDED
0800              ;            ROM              RAM
E6F8              GETBYT   EQU $E6F8            ;1EEF
DEC9              SYNERR   EQU $DEC9            ;16CC
DD67              FRMNUM   EQU $DD67            ;156A
E752              GETADR   EQU $E752            ;1F49
0800              ;
9200                       ORG $9200
9200                       OBJ $800
9200              ;
9200              ;PROCESS &
9200 48           BEGIN    PHA
9201 20 35 95              JSR SAVEZP           ;SAVE ZERO PG
9204 68                    PLA
9205 A2 02                 LDX #$02
9207 CA           CHRSFN   DEX
9208 30 53                 BMI ERRX
920A DD A9 95              CMP CHRTBL,X         ;'S' OR 'D'
920D D0 F8                 BNE CHRSFN           ;TRY AGAIN
920F 8A                    TXA
9210 0A                    ASL                  ;TIMES 2
9211 AA                    TAX
9212 20 B1 00     SR02     JSR CHRGET           ;NEXT CHAR
9215 F0 46                 BEQ ERRX
9217 C9 28                 CMP #$28             ; (
9219 D0 F7                 BNE SR02
```

```
921B BD A6 95          LDA LOC+01,X      ;JMP TO
921E 48                PHA               ;ROUTINE
921F BD A5 95          LDA LOC,X         ;VIA
9222 48                PHA               ;RTS
9223 60                RTS
9224           ;
9224           ; ** AMPER-SEARCH **
9224           ;
9224 20 22 94  SEARCH  JSR GNAME         ;GET NAME
9227 20 61 94          JSR STRING        ;CONVERT
922A 20 78 94          JSR FARRAY        ;FIND NAME
922D B0 34             BCS ERRV
922F 86 D0             STX NAPTR         ;NA$
9231 84 D1             STY NAPTR+01
9233 20 B1 00          JSR CHRGET
9236 20 67 DD          JSR FRMNUM
9239 20 52 E7          JSR GETADR
923C A5 50             LDA Z50
923E 85 D8             STA L             ;LOWER SUBSC
9240 A5 51             LDA Z50+01
9242 85 D9             STA L+01
9244 20 B1 00          JSR CHRGET
9247 20 67 DD          JSR FRMNUM
924A 20 52 E7          JSR GETADR
924D A5 50             LDA Z50
924F 85 DA             STA H             ;UPPER SUBSC
9251 A5 51             LDA Z50+01
9253 85 DB             STA H+01
9255 20 22 94          JSR GNAME
9258 20 61 94          JSR STRING
925B 90 1D             BCC SR20
925D           ;
925D           ; ** ERROR **
925D           ;
925D 20 5A 95  ERRX    JSR RSZP
9260 4C C9 DE          JMP SYNERR
9263           ;
9263           ; ** VARIABLE NOT FOUND MSG. **
9263           ;
9263 A2 00     ERRV    LDX #$00
9265 BD AB 95  SR18    LDA MSG1,X        ;ERROR MSG
9268 C9 C0             CMP #$C0          ;@ DELIMITER
926A F0 F1             BEQ ERRX
926C 09 80             ORA #$80
926E 20 ED FD          JSR COUT
9271 E0 0C             CPX #$0C
9273 D0 02             BNE SR19
9275 A2 19             LDX #$19
9277 E8        SR19    INX
9278 D0 EB             BNE SR18          ;ALWAYS
927A 20 B2 94  SR20    JSR FSIMPL        ;FIND NAME
927D B0 E4             BCS ERRV
927F 86 D2             STX SAPTR         ;ST$
9281 84 D3             STY SAPTR+01
9283 20 B1 00          JSR CHRGET
9286 20 F8 E6          JSR GETBYT
9289 86 DC             STX PL            ;FIRST POSITION
928B 20 B1 00          JSR CHRGET
928E 20 F8 E6          JSR GETBYT
9291 86 DD             STX PH            ;LAST POSITION
9293 20 22 94          JSR GNAME
9296 20 41 94          JSR INTE
9299 B0 C2             BCS ERRX
929B 20 78 94          JSR FARRAY
929E 90 09             BCC SR21
92A0 20 B2 94          JSR FSIMPL
92A3 B0 BE             BCS ERRV
92A5 A9 FF             LDA #$FF
92A7 85 EB             STA SIZE          ;# OF HITS ONLY
92A9 86 D4     SR21    STX JAPTR         ;I%
92AB 84 D5             STY JAPTR+01
92AD 20 22 94          JSR GNAME
92B0 20 41 94          JSR INTE
92B3 B0 A8             BCS ERRX
```

```
92B5 20 B2 94              JSR FSIMPL
92B8 B0 A9                 BCS ERRV
92BA 86 D6                 STX NPT              ;N%
92BC 84 D7                 STY NPT+01
92BE 20 B1 00              JSR CHRGET
92C1 D0 9A                 BNE ERRX
92C3             ;
92C3             ; ** FINISHED PARAMETERS **
92C3             ;
92C3             ; ** SET UP POINTERS **
92C3             ;
92C3 18                    CLC
92C4 A5 D4                 LDA JAPTR
92C6 69 07                 ADC #$07
92C8 85 D4                 STA JAPTR            ;I%
92CA A5 D5                 LDA JAPTR+01
92CC 69 00                 ADC #$00
92CE 85 D5                 STA JAPTR+01
92D0 A5 DA                 LDA H
92D2 85 50                 STA Z50
92D4 A5 DB                 LDA H+01
92D6 85 51                 STA Z50+01
92D8 A9 03                 LDA #$03
92DA 85 54                 STA $54
92DC A9 00                 LDA #$00
92DE 85 55                 STA $55
92E0 20 E9 94              JSR MPLY
92E3 86 E0                 STX NAPTH            ;NA$(H)
92E5 84 E1                 STY NAPTH+01
92E7 A5 D8                 LDA L
92E9 85 50                 STA Z50
92EB A5 D9                 LDA L+01
92ED 85 51                 STA Z50+01
92EF 20 E9 94              JSR MPLY
92F2 86 D0                 STX NAPTR            ;NA$(L)
92F4 84 D1                 STY NAPTR+01
92F6             ;
92F6 18                    CLC
92F7 A5 D2                 LDA SAPTR
92F9 69 02                 ADC #$02
92FB 85 D2                 STA SAPTR            ;ST$
92FD A5 D3                 LDA SAPTR+01
92FF 69 00                 ADC #$00
9301 85 D3                 STA SAPTR+01
9303 A0 00                 LDY #$00
9305 B1 D2                 LDA (SAPTR),Y
9307 D0 03                 BNE SR22
9309 4C 1E 94              JMP RETURN           ;NULL
930C 85 E9       SR22      STA LENSA
930E C8                    INY
930F B1 D2                 LDA (SAPTR),Y        ;SAVE
9311 85 E4                 STA CSAPTR           ;ADDRESS
9313 C8                    INY
9314 B1 D2                 LDA (SAPTR),Y
9316 85 E5                 STA CSAPTR+01
9318             ;
9318             ; ** START SEARCH **
9318             ;
9318 A0 00       NEXT      LDY #$00
931A B1 D0                 LDA (NAPTR),Y
931C F0 4E                 BEQ NEXTNA           ;NULL
931E 85 E8                 STA LENNA            ;LEN(NA$())
9320 C8                    INY
9321 B1 D0                 LDA (NAPTR),Y
9323 85 E2                 STA CNAPTR
9325 C8                    INY
9326 B1 D0                 LDA (NAPTR),Y
9328 85 E3                 STA CNAPTR+01
932A A4 DC                 LDY PL
932C 88                    DEY
932D C4 E8                 CPY LENNA
932F B0 3B                 BCS NEXTNA
9331 A9 00       NXTNAC    LDA #$00
9333 85 E7                 STA PS               ;CURRENT POSITION
```

```
9335 85 EA            STA SWITCH
9337 B1 E2    CONT    LDA (CNAPTR),Y
9339 C8               INY
933A 84 E6            STY SAVEY
933C A4 E7            LDY PS
933E D1 E4            CMP (CSAPTR),Y
9340 F0 0A            BEQ SR25          ;POSSIBLE MATCH
9342 B1 E4            LDA (CSAPTR),Y
9344 C9 3D            CMP #$3D          ;MATCH ANYTHING
9346 F0 45            BEQ MATCH
9348 C9 0E            CMP #$0E          ;CNTL N
934A D0 11            BNE SR26          ;NOT WILD CARD
934C                  ;
934C                  ; ** POSSIBLE MATCH **
934C                  ;
934C A9 FF    SR25    LDA #$FF
934E 85 EA            STA SWITCH
9350 C8               INY
9351 C4 E9            CPY LENSA         ;AT END?
9353 F0 38            BEQ MATCH         ;IT'S A MATCH!
9355 E6 E7            INC PS
9357 F0 13            BEQ NEXTNA
9359 A4 E6            LDY SAVEY
935B D0 DA            BNE CONT          ;ALWAYS
935D A4 E6    SR26    LDY SAVEY
935F 24 EA            BIT SWITCH
9361 10 01            BPL SR28
9363 88               DEY
9364 C4 E8    SR28    CPY LENNA         ;AT END?
9366 B0 04            BCS NEXTNA        ;BR YES
9368 C4 DD            CPY PH            ;LAST POSITION
936A 90 C5            BCC NXTNAC        ;NEXT CHARACTER
936C 18       NEXTNA  CLC               ;NEXT NA$(I)
936D A5 D0            LDA NAPTR
936F 69 03            ADC #$03
9371 85 D0            STA NAPTR
9373 A5 D1            LDA NAPTR+01
9375 69 00            ADC #$00
9377 85 D1            STA NAPTR+01
9379 E6 D8            INC L
937B D0 02            BNE SR33
937D E6 D9            INC L+01
937F 38       SR33    SEC
9380 A5 E0            LDA NAPTH
9382 E5 D0            SBC NAPTR
9384 A5 E1            LDA NAPTH+01
9386 E5 D1            SBC NAPTR+01
9388 B0 8E            BCS NEXT
938A 4C 1E 94         JMP RETURN        ;AT NA$(H)
938D                  ;
938D                  ; ** FOUND A MATCH **
938D                  ;
938D 24 EB    MATCH   BIT SIZE
938F 30 18            BMI SZONLY        ; # MATCHES ONLY
9391 A0 00            LDY #$00
9393 A5 D9            LDA L+01          ;SUBSCRIPT
9395 91 D4            STA (JAPTR),Y
9397 C8               INY
9398 A5 D8            LDA L
939A 91 D4            STA (JAPTR),Y
939C 18               CLC
939D A5 D4            LDA JAPTR
939F 69 02            ADC #$02
93A1 85 D4            STA JAPTR
93A3 A5 D5            LDA JAPTR+01
93A5 69 00            ADC #$00
93A7 85 D5            STA JAPTR+01
93A9 A0 03    SZONLY  LDY #$03
93AB 18               CLC
93AC B1 D6            LDA (NPT),Y
93AE 69 01            ADC #$01          ;N%=N%+1
93B0 91 D6            STA (NPT),Y
93B2 88               DEY
```

```
93B3 B1 D6                   LDA (NPT),Y
93B5 30 07                   BMI ONLY1          ;1ST OCCURRENCE
93B7 69 00                   ADC #$00
93B9 91 D6                   STA (NPT),Y
93BB 4C 6C 93                JMP NEXTNA
93BE A9 00          ONLY1    LDA #$00
93C0 91 D6                   STA (NPT),Y
93C2 C8                      INY
93C3 A9 01                   LDA #$01           ;N%=1
93C5 91 D6                   STA (NPT),Y
93C7                ;
93C7                ; ** FINISHED AMPER-SEARCH **
93C7                ;
93C7 4C 1E 94                JMP RETURN
93CA 4C 5D 92       ERRXX    JMP ERRX
93CD 4C 63 92       ERRVX    JMP ERRV
93D0                ;
93D0                ; ** DEALLOCATE **
93D0                ;
93D0 20 22 94       DEALLO   JSR GNAME          ;GET NAME
93D3 C9 24                   CMP #$24           ;$
93D5 F0 05                   BEQ RE50
93D7 20 41 94                JSR INTE           ;%
93DA D0 03                   BNE RE55           ;ALWAYS
93DC 20 61 94       RE50     JSR STRING
93DF B0 E9          RE55     BCS ERRXX
93E1 20 78 94                JSR FARRAY
93E4 B0 E7                   BCS ERRVX
93E6 86 D0                   STX NAPTR          ;NA$
93E8 84 D1                   STY NAPTR+01
93EA A0 02                   LDY #$02
93EC B1 D0                   LDA (NAPTR),Y
93EE 85 D2                   STA OFFSET
93F0 C8                      INY
93F1 B1 D0                   LDA (NAPTR),Y
93F3 85 D3                   STA OFFSET+01
93F5 18                      CLC
93F6 A5 D2                   LDA OFFSET
93F8 65 D0                   ADC NAPTR
93FA 85 D4                   STA A1
93FC A5 D3                   LDA OFFSET+01
93FE 65 D1                   ADC NAPTR+01
9400 85 D5                   STA A1+01
9402 20 18 95                JSR MOVE           ;MOVE VARIABLES
9405 38                      SEC
9406 A5 6D                   LDA $6D
9408 E5 D2                   SBC OFFSET
940A 85 6D                   STA $6D
940C A5 6E                   LDA $6E
940E E5 D3                   SBC OFFSET+01
9410 85 6E                   STA $6E
9412 20 B7 00                JSR CHRGOT
9415 C9 29                   CMP #$29           ; )
9417 D0 B7                   BNE DEALLO         ;NEXT VAR
9419 20 B1 00                JSR CHRGET
941C D0 AC                   BNE ERRXX
941E                ;
941E                ; ** FINISHED **
941E                ;
941E 20 5A 95       RETURN   JSR RSZP           ;RESTORE PAGE0
9421 60                      RTS
9422                ;
9422                ; ******************************
9422                ;     SUBROUTINES
9422                ; ******************************
9422                ;
9422                ; ** GET VARIABLE NAME **
9422                ;
9422 A2 00          GNAME    LDX #$00
9424 20 B1 00       GR01     JSR CHRGET
9427 C9 2C                   CMP #$2C           ; ,
9429 F0 11                   BEQ GR03
942B C9 29                   CMP #$29           ; )
942D F0 0D                   BEQ GR03
```

```
942F 9D B5 95           STA NAME,X        ;SAVE NAME
9432 E8                 INX
9433 E0 10              CPX #$10          ;16 IS ENOUGH
9435 D0 ED              BNE GR01
9437 68                 PLA
9438 68                 PLA               ;POP STACK
9439 4C 5D 92           JMP ERRX
943C CA        GR03     DEX
943D BD B5 95           LDA NAME,X        ; $ OR %
9440 60                 RTS
9441           ;
9441           ; ** INTEGER NAME **
9441           ;
9441 C9 25     INTE     CMP #$25          ;%
9443 D0 1A              BNE ERRI          ;NOT %
9445 8D B7 95           STA NAME+02       ;SAVE
9448 E0 01              CPX #$01          ;NAME
944A D0 04              BNE GR10          ;IN
944C A9 80              LDA #$80          ;APPLESOFT
944E D0 07              BNE GR14          ;FORMAT
9450 A2 01     GR10     LDX #$01
9452 A9 80     GR12     LDA #$80
9454 1D B5 95           ORA NAME,X
9457 9D B5 95  GR14     STA NAME,X
945A CA                 DEX
945B 10 F5              BPL GR12
945D 18                 CLC               ;CLEAR ERR
945E 60                 RTS
945F 38        ERRI     SEC               ;SET ERR
9460 60                 RTS
9461           ;
9461           ; ** STRING NAME **
9461           ;
9461 C9 24     STRING   CMP #$24          ;$
9463 D0 11              BNE ERRS
9465 8D B7 95           STA NAME+02
9468 A9 80              LDA #$80
946A E0 01              CPX #$01          ;SAVE
946C F0 03              BEQ GR18          ;NAME
946E 0D B6 95           ORA NAME+01
9471 8D B6 95  GR18     STA NAME+01
9474 18                 CLC
9475 60                 RTS
9476 38        ERRS     SEC               ;SET ERR
9477 60                 RTS
9478           ;
9478           ; ** FIND ARRAY NAME **
9478           ; ** IN VARIABLE SPACE **
9478           ;
9478 A5 6B     FARRAY   LDA $6B
947A 85 DE              STA TEM6X
947C A5 6C              LDA $6C
947E 85 DF              STA TEM6X+01
9480 A0 00     F02      LDY #$00
9482 B1 DE              LDA (TEM6X),Y
9484 CD B5 95           CMP NAME          ;1ST CHARACTER
9487 D0 08              BNE F04
9489 C8                 INY
948A B1 DE              LDA (TEM6X),Y
948C CD B6 95           CMP NAME+01       ;2ND CHARACTER
948F F0 1B              BEQ FOUND
9491 18        F04      CLC               ;LOOK AT
9492 A0 02              LDY #$02          ;NEXT NAME
9494 B1 DE              LDA (TEM6X),Y
9496 65 DE              ADC TEM6X
9498 48                 PHA
9499 C8                 INY
949A B1 DE              LDA (TEM6X),Y
949C 65 DF              ADC TEM6X+01
949E 85 DF              STA TEM6X+01
94A0 68                 PLA
94A1 85 DE              STA TEM6X
94A3 C5 6D              CMP $6D
94A5 A5 DF              LDA TEM6X+01
94A7 E5 6E              SBC $6E
```

```
94A9 90 D5                BCC F02             ;TRY NEXT ONE
94AB 60                   RTS                 ;NOT FOUND
94AC             ;
94AC A6 DE       FOUND    LDX TEM6X           ;RTN WITH
94AE A4 DF                LDY TEM6X+01        ;ADDRESS
94B0 18                   CLC
94B1 60                   RTS
94B2             ;
94B2             ; ** FIND SIMPLE NAME  **
94B2             ; ** IN VARIABLE SPACE **
94B2             ;
94B2 A5 69       FSIMPL   LDA $69
94B4 85 DE                STA TEM6X
94B6 A5 6A                LDA $6A
94B8 85 DF                STA TEM6X+01
94BA A0 00       FS2      LDY #$00
94BC B1 DE                LDA (TEM6X),Y
94BE CD B5 95             CMP NAME            ;1ST CHARACTER
94C1 D0 08                BNE FS4
94C3 C8                   INY
94C4 B1 DE                LDA (TEM6X),Y
94C6 CD B6 95             CMP NAME+01         ;2ND CHARACTER
94C9 F0 18                BEQ FOUNDS
94CB 18          FS4      CLC                 ;TRY NEXT ONE
94CC A5 DE                LDA TEM6X
94CE 69 07                ADC #$07            ;DISPLACEMENT
94D0 85 DE                STA TEM6X
94D2 A5 DF                LDA TEM6X+01
94D4 69 00                ADC #$00
94D6 85 DF                STA TEM6X+01
94D8 A5 DE                LDA TEM6X
94DA C5 6D                CMP $6D             ;AT END?
94DC A5 DF                LDA TEM6X+01
94DE E5 6E                SBC $6E
94E0 90 D8                BCC FS2             ;NEXT ONE
94E2 60                   RTS                 ;NOT FOUND
94E3             ;
94E3 A6 DE       FOUNDS   LDX TEM6X           ;RTN WITH
94E5 A4 DF                LDY TEM6X+01        ;ADDRESS
94E7 18                   CLC
94E8 60                   RTS
94E9             ;
94E9             ; ** MULTIPLY ROUTINE **
94E9             ;
94E9 18          MPLY     CLC
94EA A5 D0                LDA NAPTR
94EC 69 07                ADC #$07
94EE 85 52                STA $52
94F0 A5 D1                LDA NAPTR+01
94F2 69 00                ADC #$00
94F4 85 53                STA $53
94F6             ;
94F6             ; ** FROM 'RED' MANUAL **
94F6             ;
94F6 A0 10                LDY #$10
94F8 A5 50       MUL2     LDA $50
94FA 4A                   LSR
94FB 90 0C                BCC MUL4
94FD 18                   CLC
94FE A2 FE                LDX #$FE
9500 B5 54       MUL3     LDA $54,X
9502 75 56                ADC $56,X
9504 95 54                STA $54,X
9506 E8                   INX
9507 D0 F7                BNE MUL3
9509 A2 03       MUL4     LDX #$03
950B 76 50       MUL5     ROR $50,X
950D CA                   DEX
950E 10 FB                BPL MUL5
9510 88                   DEY
9511 D0 E5                BNE MUL2
9513 A6 50                LDX Z50
9515 A4 51                LDY Z50+01
9517 60                   RTS
```

```
9518                ;
9518                ; ** MOVE VARIABLES **
9518                ;
9518 A0 00          MOVE     LDY #$00
951A B1 D4          MV01     LDA (A1),Y
951C 91 D0                   STA (NAPTR),Y
951E E6 D0                   INC NAPTR
9520 D0 02                   BNE NXTA1
9522 E6 D1                   INC NAPTR+01
9524 A5 D4          NXTA1    LDA A1
9526 C5 6D                   CMP $6D
9528 A5 D5                   LDA A1+01
952A E5 6E                   SBC $6E
952C E6 D4                   INC A1
952E D0 02                   BNE MV02
9530 E6 D5                   INC A1+01
9532 90 E6          MV02     BCC MV01           ;NEXT ONE
9534 60                      RTS
9535                ;
9535                ; ** SAVE ZERO  **
9535                ; ** PAGE SPACE **
9535                ;
9535 A2 00          SAVEZP   LDX #$00
9537 B5 D0          SV02     LDA NAPTR,X
9539 9D D6 95                STA ZPSV,X
953C E8                      INX
953D E0 20                   CPX #$20           ;SAVE
953F D0 F6                   BNE SV02           ;32 SPOTS
9541 A2 00                   LDX #$00
9543 86 EB                   STX SIZE           ;INIT
9545 B5 50          SV04     LDA $50,X          ;ALSO $50, $55
9547 9D D0 95                STA SV50,X
954A E8                      INX
954B E0 06                   CPX #$06
954D D0 F6                   BNE SV04
954F A2 0F                   LDX #$0F
9551 A9 20                   LDA #$20           ;CLEAR
9553 9D B5 95       CLEAR    STA NAME,X         ;NAME AREA
9556 CA                      DEX
9557 10 FA                   BPL CLEAR
9559 60                      RTS
955A                ;
955A                ; ** RESTORE ZERO **
955A                ; ** PAGE SPACE   **
955A                ;
955A A2 00          RSZP     LDX #$00
955C BD D6 95       RS02     LDA ZPSV,X
955F 95 D0                   STA NAPTR,X
9561 E8                      INX
9562 E0 20                   CPX #$20
9564 D0 F6                   BNE RS02
9566 A2 00                   LDX #$00
9568 BD D0 95       RS04     LDA SV50,X
956B 95 50                   STA $50,X
956D E8                      INX
956E E0 06                   CPX #$06
9570 D0 F6                   BNE RS04
9572 60                      RTS
9573                ;
9573                ; ** DATA STORAGE **
9573                ;
9573 C1 CD D0                HEX C1CDD0C5D2ADD3C5C1D2C3C8
9576 C5 D2 AD
9579 D3 C5 C1
957C D2 C3 C8
957F C1 CC C1                HEX C1CCC1CEA0C7AEA0C8C9CCCC
9582 CE A0 C7
9585 AE A0 C8
9588 C9 CC CC
958B C3 CF CD                HEX C3CFCDCDC5D2C3C9C1CCA0D2C9C7C8D4D3A0
958E CD C5 D2
9591 C3 C9 C1
9594 CC A0 D2
9597 C9 C7 C8
959A D4 D3 A0
```

```
959D D2 C5 D3    574           HEX D2C5D3C5D2D6C5C4
95A0 C5 D2 D6
95A3 C5 C4
95A5 CF 93       575   LOC     HEX CF93              ;DEALLOC-1
95A7 23 92       576           HEX 2392              ;SEARCH-1
95A9 44          577   CHRTBL  HEX 44                ;D
95AA 53          578           HEX 53                ;S
95AB 8D          579   MSG1    HEX 8D
95AC D6 C1 D2    580           HEX D6C1D2C9C1C2CCC5A0
95AF C9 C1 C2
95B2 CC C5 A0
95B5 A0 A0 A0    581   NAME    HEX A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0A0
95B8 A0 A0 A0
95BB A0 A0 A0
95BE A0 A0 A0
95C1 A0 A0 A0
95C4 A0
95C5 8D          582           HEX 8D
95C6 CE CF D4    583           HEX CECFD4A0C6CFD5CEC4
95C9 A0 C6 CF
95CC D5 CE C4
95CF C0          584           HEX C0
95D0 A0 A0 A0    585   SV50    HEX A0A0A0A0A0A0
95D3 A0 A0 A0
95D6 A0          586   ZPSV    HEX A0                ; $20 SPACES
95D7             587           END

***** END OF ASSEMBLY

SYMBOL TABLE SORTED ALPHABETICALLY

  A1      00D4  BEGIN   9200  CHRGET  00B1  CHRGOT  00B7  CHRSFN  9207
  CHRTBL  95A9  CLEAR   9553  CNAPTR  00E2  CONT    9337  COUT    FDED
  CSAPTR  00E4  DEALLO  93D0  ERRI    945F  ERRS    9476  ERRV    9263
  ERRVX   93CD  ERRX    925D  ERRXX   93CA  F02     9480  F04     9491
  FARRAY  9478  FOUND   94AC  FOUNDS  94E3  FRMNUM  DD67  FS2     94BA
  FS4     94CB  FSIMPL  94B2  GETADR  E752  GETBYT  E6F8  GNAME   9422
  GR01    9424  GR03    943C  GR10    9450  GR12    9452  GR14    9457
  GR18    9471  H       00DA  INTE    9441  JAPTR   00D4  L       00D8
  LENNA   00E8  LENSA   00E9  LOC     95A5  MATCH   938D  MOVE    9518
  MPLY    94E9  MSG1    95AB  MUL2    94F8  MUL3    9500  MUL4    9509
  MUL5    950B  MV01    951A  MV02    9532  NAME    95B5  NAPTH   00E0
  NAPTR   00D0  NEXT    9318  NEXTNA  936C  NPT     00D6  NXTA1   9524
  NXTNAC  9331  OFFSET  00D2  ONLY1   93BE  PH      00DD  PL      00DC
  PS      00E7  RE50    93DC  RE55    93DF  RETURN  941E  RS02    955C
  RS04    9568  RSZP    955A  SAPTR   00D2  SAVEY   00E6  SAVEZP  9535
  SEARCH  9224  SIZE    00EB  SR02    9212  SR18    9265  SR19    9277
  SR20    927A  SR21    92A9  SR22    930C  SR25    934C  SR26    935D
  SR28    9364  SR33    937F  STRING  9461  SV02    9537  SV04    9545
  SV50    95D0  SWITCH  00EA  SYNERR  DEC9  SZONLY  93A9  TEM6X   00DE
  Z50     0050  ZPSV    95D6

SYMBOL TABLE SORTED BY ADDRESS

  Z50     0050  CHRGET  00B1  CHRGOT  00B7  NAPTR   00D0  SAPTR   00D2
  OFFSET  00D2  JAPTR   00D4  A1      00D4  NPT     00D6  L       00D8
  H       00DA  PL      00DC  PH      00DD  TEM6X   00DE  NAPTH   00E0
  CNAPTR  00E2  CSAPTR  00E4  SAVEY   00E6  PS      00E7  LENNA   00E8
  LENSA   00E9  SWITCH  00EA  SIZE    00EB  BEGIN   9200  CHRSFN  9207
  SR02    9212  SEARCH  9224  ERRX    925D  ERRV    9263  SR18    9265
  SR19    9277  SR20    927A  SR21    92A9  SR22    930C  NEXT    9318
  NXTNAC  9331  CONT    9337  SR25    934C  SR26    935D  SR28    9364
  NEXTNA  936C  SR33    937F  MATCH   938D  SZONLY  93A9  ONLY1   93BE
  ERRXX   93CA  ERRVX   93CD  DEALLO  93D0  RE50    93DC  RE55    93DF
  RETURN  941E  GNAME   9422  GR01    9424  GR03    943C  INTE    9441
  GR10    9450  GR12    9452  GR14    9457  ERRI    945F  STRING  9461
  GR18    9471  ERRS    9476  FARRAY  9478  F02     9480  F04     9491
  FOUND   94AC  FSIMPL  94B2  FS2     94BA  FS4     94CB  FOUNDS  94E3
  MPLY    94E9  MUL2    94F8  MUL3    9500  MUL4    9509  MUL5    950B
  MOVE    9518  MV01    951A  NXTA1   9524  MV02    9532  SAVEZP  9535
  SV02    9537  SV04    9545  CLEAR   9553  RSZP    955A  RS02    955C
  RS04    9568  LOC     95A5  CHRTBL  95A9  MSG1    95AB  NAME    95B5
  SV50    95D0  ZPSV    95D6  FRMNUM  DD67  SYNERR  DEC9  GETBYT  E6F8
  GETADR  E752  COUT    FDED
```

Applesoft Variable Lister

by Richard Albright

The ability to dump the values of all variables can be immensely helpful in Applesoft program development. The Applesoft Variable Lister provides this ability and can be used with any program, located anywhere in memory.

This Lister may be attached to any Applesoft program by simply merging its Applesoft subroutine with the main program. This can be accomplished using the standard Apple RENUMBER program. Any unused space in which the 71 lines will fit without affecting the normal operation of the program will do, but the end of the program is the recommended location.

Once installed within the program, the Lister can be invoked like any Applesoft subroutine; that is, by means of a GOSUB n statement where n is the number of the first line of the subroutine within the program. This GOSUB can be issued by the main program or from the keyboard.

The Lister will operate under both ROM and RAM Applesoft, but requires the use of a disk drive. The disk drive last accessed before the Lister was invoked must contain a diskette on which the Lister's two machine language routines are stored under the names SHELL-METZNER SORT and APPLESOFT VARIABLE LISTER OBJ. In addition, one file buffer must be available.

Using the Lister

The output from the Lister will appear on both a printer and the screen if the printer is open at the time the Lister is invoked. Otherwise, the output goes to the screen only. The output format for the printer is slightly different from the screen format.

Figure 1 is an example of the printed output format. User responses to prompts have been underlined. When the Lister is invoked, it first queries you for

ALPHA SORT, MEMORY SORT OR QUIT?

with the double-underlined letters appearing in inverse on the screen. A 'Q' response at this point simply terminates the Lister with no further ado. An 'A' response results in an alphabetical listing of variables while an 'M' response will cause variables to be listed in the order stored. After either an 'A' or an 'M' response, the disk drive will activate briefly while a temporary file is created. (More on this later.)

Next, the Lister asks if you would like to display

VALUES OR LOCATIONS?

A 'V' response will give you the current value for each simple variable (as shown in figure 1); an 'L' response produces a display of locations at which the values are stored in memory.

At this point the disk drive will again activate while the APPLESOFT VARIABLE LISTER OBJ and (if ALPHA SORT has been selected) the SHELL-METZNER SORT files are read and another temporary file is created. If sorting is performed, a

SORTING VARIABLE NAMES . . .

message is displayed while the names are being sorted. Usually the sorting process takes only a few seconds.

After a slight pause, the first page of variables will be displayed (and printed if the printer is on). A two-column format is used for all combinations of display options. Numeric values are displayed to full precision, but strings longer than 14 characters are truncated. Forty variables appear on a full page. The message

HIT SPACE BAR TO CONTINUE; 'ESC' TO QUIT

appears on the screen (*not* on the printer) after each page. Pressing the ESC key results in the termination of the Lister (after some more disk activity). Pressing the space bar, on the other hand, causes the next page of simple variables to be displayed. If all simple variables have been displayed, the first page of array variables is produced. Notice that array variable values *cannot* be displayed; only the location of the start of each array is provided — even if VALUES is the selected display mode.

Following the last array page, the Lister is terminated by pressing either the space bar or the ESC key. At this point the disk drive will again briefly activate. If the Lister was invoked from the keyboard, an error message will be encountered and can be ignored. If invoked from the main program, execution continues normally with the statement following the GOSUB.

***Figure 1:* Example of Printed Output**

```
APPLESOFT VARIABLE LISTER
ALPHA SORT, MEMORY SORT OR QUIT? A
VALUES OR LOCATIONS? V
SORTING VARIABLE NAMES...
```

Simple Variables; Alpha Order

Var	Value	Var	Value
A	0	LB$	[
A $		ML%	3
B1$	3 LETTERS	MQ	99
B2$	0	MR	99
BS$		NL%	12
CA$		NQ%	9
CL$	CLOSE	NR	12
CR$			
		NR%	0
D	0	NS%	10
D $		O0$	OPEN SURVEY C
EQ	1	O1$	OPEN SURVEY T
ER%	1	O2$	OPEN INTERVIE
F1$	TEST1	OP$	OPEN SURVEY T
FQ%	9	PP	0
I	96	Q	1
J	0	QQ	1
K	0	R	0
L	0	R0$	READ SURVEY C
L1	9	R1$	READ SURVEY T
L2	1	RE$	READ SURVEY C
RR	1		
RT%	4		
SS	0		
T $			
UN	2048		
V	5		
W2$	WRITE INTERVI		
XR%	1		
ZZ	1		

Array Variables; Alpha Order

Var	Hex	Dec	Var	Hex	Dec
CT	$2DB2	11698			
DT$	$33F5	13301			
QC	$2F99	12185			
R $	$3194	12692			

The Source Code

The Applesoft Variable Lister consists of an Applesoft subroutine (listing 1), a machine language setup routine (listing 2), and a machine language sort routine (listing 3). The Applesoft subroutine can be entered and SAVEd under an arbitrary name. The machine language routines may be entered into memory either directly using the monitor or indirectly using an assembler, then BSAVEd under the names APPLESOFT VARIABLE LISTER OBJ (for the setup routine) and SHELL-METZNER SORT (for the sort routine).

Technical Notes

The Lister's Applesoft subroutine occupies about 3500 bytes of memory. In addition, execution of the Lister requires a certain amount of free space: five bytes per variable if the ALPHA SORT option is chosen and ten bytes per variable if the MEMORY SORT option is selected. The Lister does *not* verify that this space is available. If insufficient space exists, the result is unpredictable.

If the addition of the Lister to a program using Hi-Res graphics causes the program to overflow into the Hi-Res memory area, then the merged program should be saved and reloaded above the Hi-Res memory. If only Hi-Res page one is used, this move is accomplished by executing the following POKEs between the SAVE and the LOAD:

```
POKE 103,1:POKE 104,64:POKE 16384,0
```

To move the program above Hi-Res page two, use the following POKEs:

```
POKE 103,1:POKE 104,96:POKE 24576,0
```

The Lister's Applesoft subroutine itself uses three simple variables (ZZ, ZZ% and ZZ$) and one array variable (ZZ). These variable names should be avoided in the main program: if they appear in the main program, execution of the Lister subroutine will reset their values. ZZ will always appear in the simple variable listing, but ZZ%, ZZ$, and the ZZ array variable will appear only if the Lister is executed more than once between CLEARs or RUNs.

Both the SHELL-METZNER SORT and APPLESOFT VARIABLE LISTER OBJ routines use page three of memory. However, the contents of page three at the time the Lister is invoked are saved on diskette in a temporary file named PAGE 3 SAVE. The original page three is restored as part of the Lister termination processing.

Both machine language routines make extensive use of page zero, but again, a temporary file (PAGE 0 SAVE) is used to save the initial values and they are restored when the Lister finishes. However, only part of page zero is restored, leaving some page zero values altered after running the Lister. Specifically, locations 24 to 31 ($18 to $1F) are altered. These locations are not normally used by an Applesoft program.

A third temporary file (PAGE 0 SAVE2) is used if ALPHA SORT is selected. It is used to restore page zero values after the sorting has been completed. All temporary files are deleted by the Lister if it terminates normally. Both the SHELL-METZNER SORT and the APPLESOFT VARIABLE LISTER OBJ routines are fully relocatable.

The sorting routine uses the Shell-Metzner algorithm and is designed to sort fixed-length records so that the one with the lowest key value appears highest in the memory. Up to 32,767 records occupying contiguous locations may be sorted with this routine, space permitting. Each record may be up to 255 bytes in length and must have a sort key field that may be as short as one byte or as long as the entire record. The key is evaluated as an unsigned binary integer field and the sorting is performed on that basis.

The sort routine uses memory locations 25 to 31 ($19 to $1F) as an input argument list, interpreted as follows:

25	($19):	record length
26	($1A):	key offset (i.e., record characters preceding the key)
27	($1B):	key length
28-29	($1C-$1D):	number of records
30-31	($1E-$1F):	pointer to 1st byte of 1st record

The last two items are two-byte binary integers, presented in the usual low byte/high byte format. The sorting routine does not alter the values placed in any of these locations, nor does it verify their consistency.

Although the sort routine can handle thousands of records, the setup routine can handle a maximum of 255 variables of any type (simple or array). If more than 255 simple or array variables exist, the operation of the Lister is unpredictable.

Strings containing one or more carriage return characters (ASCII 13) cause formatting problems on both the screen and the printer. If the value appears in the left column on the screen, then one variable may be omitted from the right column. On the printer, one or more blank lines may be introduced. This problem is exemplified in figure 1: the CR$ string consists of a single carriage return character, resulting in the unexpected gap between the CR$ and D variables in the left column and the NR and NR% variables in the right column.

```
10  REM  ************************
20  REM  *                      *
30  REM  *   VARIABLE LISTER    *
40  REM  *   RICHARD ALBRIGHT   *
50  REM  *                      *
60  REM  *  COPYRIGHT (C) 1982  *
70  REM  *     MICRO INK, INC.  *
80  REM  * CHELMSFORD, MA 01824 *
90  REM  *  ALL RIGHTS RESERVED *
100  REM *                      *
110  REM ************************
120  REM
130  REM
140  FOR ZZ = 32 TO 35: POKE 715 + ZZ, PEEK (ZZ): NEXT ZZ
150  POKE 32,0: POKE 33,40: POKE 34,0: POKE 35,24: TEXT : NORMAL
160  PRINT : INVERSE : PRINT  SPC( 7);"APPLESOFT VARIABLE LISTER"; SPC( 8
     ): NORMAL
170  FOR ZZ = 0 TO 9: POKE 752 + ZZ,48 + ZZ: NEXT ZZ: FOR ZZ = 10 TO 15: POKE
     752 + ZZ,55 + ZZ: NEXT ZZ
180  PRINT : INVERSE : PRINT "A";: NORMAL : PRINT "LPHA SORT, ";: INVERSE
     : PRINT "M";: NORMAL : PRINT "EMORY SORT OR ";: INVERSE : PRINT "Q";
     : NORMAL : PRINT "UIT? ";
190 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 190
200  POKE  - 16368,0: PRINT  CHR$ (ZZ): IF ZZ <  > 193 AND ZZ <  > 205 AND
     ZZ <  > 209 THEN  PRINT  CHR$ (7): GOTO 180
210  IF ZZ = 209 THEN 830
220 ZZ = ZZ - 192: IF ZZ > 1 THEN ZZ = 2
230  POKE 250,ZZ: INVERSE : PRINT "V";: NORMAL : PRINT "ALUES OR ";: INVERSE
     : PRINT "L";: NORMAL : PRINT "OCATIONS?";
240 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 240
250  POKE  - 16368,0: PRINT  CHR$ (ZZ): IF ZZ <  > 204 AND ZZ <  > 214 THEN
      PRINT  CHR$ (7): GOTO 250
260 ZZ = ZZ - 204: IF ZZ > 0 THEN ZZ = 2
270 ZZ = ZZ +  PEEK (250)
280  PRINT  CHR$ (4);"BSAVE PAGE 3 SAVE,A$300,L$100": PRINT  CHR$ (4);"BS
     AVE PAGE 0 SAVE,A$C0,L$40"
290  PRINT  CHR$ (4);"BLOAD APPLESOFT VARIABLE LISTER OBJ": PRINT  CHR$ (
     4)
300  POKE 250,ZZ:ZZ =  FRE (0): CALL 768
310  POKE 251, PEEK (111): POKE 252, PEEK (112): IF  PEEK (250) = 2 OR  PEEK
     (250) = 4 THEN 390
320  PRINT  CHR$ (4);"BSAVE PAGE 0 SAVE2,A$C0,L$40": PRINT  CHR$ (4)
330  PRINT  CHR$ (4);"BLOAD SHELL-METZNER SORT": PRINT  CHR$ (4)
340  PRINT : PRINT "SORTING VARIABLE NAMES . . .": PRINT
350  POKE 25,5: POKE 26,0: POKE 27,3
360 ZZ =  PEEK (251) + 256 *  PEEK (252) + 5 *  PEEK (254): POKE 28, PEEK
     (253): POKE 29,0: POKE 31, INT (ZZ / 256): POKE 30,ZZ - 256 *  PEEK
     (31):ZZ =  PEEK (254): CALL 768
370  POKE 28,ZZ: POKE 29,0:ZZ =  PEEK (30) + 256 *  PEEK (31) - 5 * ZZ: POKE
     31, INT (ZZ / 256): POKE 30,ZZ - 256 *  PEEK (31): CALL 768
380  PRINT  CHR$ (4);"BLOAD PAGE 0 SAVE2": PRINT  CHR$ (4);"DELETE PAGE 0
      SAVE2": PRINT  CHR$ (4)
390  HOME : INVERSE : PRINT  SPC( 5);"SIMPLE VARIABLES; ";: IF  PEEK (250
     ) = 1 OR  PEEK (250) = 3 THEN  PRINT "ALPHA ORDER"; SPC( 6);
400  IF  PEEK (250) = 2 OR  PEEK (250) = 4 THEN  PRINT "MEMORY ORDER"; SPC(
     5);
410  PRINT : NORMAL : IF  PEEK (253) = 0 THEN  PRINT : PRINT "NO SIMPLE V
     ARIABLES": GOSUB 530: GOTO 450
420 ZZ(0) =  PEEK (253):ZZ(1) =  PEEK (251) + 256 *  PEEK (252) + 5 * ( PEEK
     (253) +  PEEK (254))
430  IF  PEEK (250) > 2 THEN ZZ = ZZ: POKE 25, PEEK (131): POKE 26, PEEK
     (132):ZZ$ = ZZ$: POKE 27, PEEK (131): POKE 28, PEEK (132):ZZ% = ZZ%:
      POKE 29, PEEK (131): POKE 30, PEEK (132)
440  GOSUB 580
450  IF  PEEK (250) > 2 THEN  POKE 250, PEEK (250) - 2
460  HOME : INVERSE : PRINT  SPC( 6);"ARRAY VARIABLES; ";: IF  PEEK (250)
      = 1 THEN  PRINT "ALPHA ORDER"; SPC( 6);
470  IF  PEEK (250) = 2 THEN  PRINT "MEMORY ORDER"; SPC( 5);
480  PRINT : NORMAL : IF  PEEK (254) = 0 THEN  PRINT : PRINT "NO ARRAY VA
     RIABLES": GOSUB 530: GOTO 500
490 ZZ(0) =  PEEK (254):ZZ(1) =  PEEK (251) + 256 *  PEEK (252) + 5 *  PEEK
     (254): GOSUB 580
500  GOTO 790
510  VTAB 2: PRINT "VAR HEX   DEC      * VAR HEX   DEC": PRINT "--- -----
      -----    * --- ----- -----": RETURN
```

```
520  VTAB 2: PRINT "VAR VALUE          * VAR VALUE": PRINT "--- ---------
     ----- * --- --------------": RETURN
530 ZZ$ = "HIT" +  CHR$ (96) + "SPACE" +  CHR$ (96) + "BAR" +  CHR$ (96) +
     "TO" +  CHR$ (96) + "CONTINUE" +  CHR$ (123) +  CHR$ (96) +  CHR$ (1
     03) + "ESC" +  CHR$ (103) +  CHR$ (96) + "TO" +  CHR$ (96) + "QUIT"
540  FOR ZZ = 1 TO  LEN (ZZ$): POKE ZZ + 1999, ASC ( MID$ (ZZ$,ZZ,1)) - 6
     4: NEXT ZZ
550 ZZ =  PEEK ( - 16384): IF ZZ < 128 THEN 550
560  POKE  - 16368,0: IF ZZ <  > 155 THEN  PRINT : PRINT : RETURN
570  POP : POP : GOTO 790
580  REM  PRINT VARIABLE NAMES & LOCATIONS
590 ZZ(10) =  INT (( PEEK (250) + 1) / 2): ON ZZ(10) GOSUB 510,520: POKE
     34,3
600 ZZ(3) = 0:ZZ(1) = ZZ(1) - 5
610 ZZ(2) = ZZ(3) + 1: IF ZZ(2) > ZZ(0) THEN  POKE 34,0: RETURN
620 ZZ(3) = ZZ(2) + 19: IF ZZ(3) > ZZ(0) THEN ZZ(3) = ZZ(0)
630 ZZ(6) = ZZ(2) - 1
640 ZZ(6) = ZZ(6) + 1: IF ZZ(6) > ZZ(3) THEN ZZ(1) = ZZ(1) - 100:ZZ(3) =
     ZZ(3) + 20: GOSUB 530: HOME : GOTO 610
650  VTAB ZZ(6) - ZZ(2) + 4:ZZ(8) = ZZ(1): GOSUB 670: PRINT  SPC( 19 -  POS
     (0));"* ";: IF ZZ(6) + 20 <  = ZZ(0) THEN ZZ(8) = ZZ(1) - 100: GOSUB
     670
660  PRINT :ZZ(1) = ZZ(1) - 5: GOTO 640
670  PRINT  CHR$ ( PEEK (ZZ(8))); CHR$ ( PEEK (ZZ(8) + 1)); CHR$ ( PEEK (
     ZZ(8) + 2));" ";: IF ZZ(10) = 2 THEN 730
680  PRINT "$";:ZZ(5) =  PEEK (ZZ(8) + 4):ZZ(4) =  PEEK (ZZ(8) + 3):ZZ(7)
     =  INT (ZZ(5) / 16): PRINT  CHR$ ( PEEK (752 + ZZ(7))); CHR$ ( PEEK
     (752 + ZZ(5) - 16 * ZZ(7)));
690 ZZ(7) =  INT (ZZ(4) / 16): PRINT  CHR$ ( PEEK (752 + ZZ(7))); CHR$ ( PEEK
     (752 + ZZ(4) - 16 * ZZ(7)));
700 ZZ$ =  STR$ (256 * ZZ(5) + ZZ(4))
710  PRINT  SPC( 6 -  LEN (ZZ$));ZZ$;
720  RETURN
730 ZZ(9) =  PEEK (ZZ(8) + 3) + 256 *  PEEK (ZZ(8) + 4):ZZ =  PEEK (ZZ(8)
     + 2) - 31: IF ZZ > 1 THEN ZZ = ZZ - 3
740  ON ZZ GOTO 750,770,780
750 ZZ(7) =  PEEK (25) + 256 *  PEEK (26) - 2: POKE ZZ(7) + 2, PEEK (ZZ(9
     ) + 2): POKE ZZ(7) + 3, PEEK (ZZ(9) + 3): POKE ZZ(7) + 4, PEEK (ZZ(9
     ) + 4): POKE ZZ(7) + 5, PEEK (ZZ(9) + 5)
760  POKE ZZ(7) + 6, PEEK (ZZ(9) + 6): PRINT ZZ;: RETURN
770 ZZ(7) =  PEEK (27) + 256 *  PEEK (28) - 2: FOR ZZ = 2 TO 4: POKE ZZ(7
     ) + ZZ, PEEK (ZZ(9) + ZZ): NEXT ZZ: PRINT  LEFT$ (ZZ$,14);: RETURN
780 ZZ(7) =  PEEK (29) + 256 *  PEEK (30) - 2: FOR ZZ = 2 TO 3: POKE ZZ(7
     ) + ZZ, PEEK (ZZ(9) + ZZ): NEXT ZZ: PRINT ZZ%;: RETURN
790  IF ZZ = 209 THEN 830
800  HOME : PRINT : PRINT  CHR$ (4);"BLOAD PAGE 0 SAVE": PRINT  CHR$ (4);
     "DELETE PAGE 0 SAVE": PRINT  CHR$ (4)
810  PRINT  CHR$ (4);"BLOAD PAGE 3 SAVE"
820  PRINT  CHR$ (4);"DELETE PAGE 3 SAVE": PRINT  CHR$ (4)
830  FOR ZZ = 32 TO 35: POKE ZZ, PEEK (715 + ZZ): NEXT ZZ
840  HOME : RETURN

]
```

```
0800                ;************************
0800                ;*                      *
0800                ;* VARIABLE LISTER OBJ  *
0800                ;*   RICHARD ALBRIGHT   *
0800                ;*                      *
0800                ;*        LISTER        *
0800                ;*                      *
0800                ;*  COPYRIGHT (C) 1982  *
0800                ;*    MICRO INK, INC.   *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                      *
0800                ;************************
0800                ;
00A5                VNAME    EPZ $A5        ;CURRENT VARIABLE NAME
00A8                VLOC     EPZ $A8        ;CURRENT VARIABLE LOCATION
00AA                VTYPE    EPZ $AA        ;VARIABLE TYPE (0=SIMPLE;1=ARR
AY)
00FD                NSIMPL   EPZ $FD        ;COUNT OF SIMPLE VARIABLES
00FE                NARRAY   EPZ $FE        ;COUNT OF ARRAY VARIABLES
0800                ;
0800                ;
0300                         ORG $300
0300                         OBJ $800
0300                ;
0300 A5 69                   LDA $69        ;INITIALIZE VARIABLE POINTER T
O
0302 85 A8                   STA VLOC       ;START OF SIMPLE VARIABLE
0304 A5 6A                   LDA $6A        ;SPACE
0306 85 A9                   STA VLOC+1
0308 A9 00                   LDA #$00       ;INITIALIZE VARIABLE COUNTERS
030A 85 FD                   STA NSIMPL     ;TO ZERO
030C 85 FE                   STA NARRAY
030E 85 AA                   STA VTYPE      ;START WITH SIMPLE VARIABLES
0310 A5 AA          TOP      LDA VTYPE      ;TOP OF MAIN LOOP
0312 18                      CLC
0313 65 AA                   ADC VTYPE      ;SET X TO 2 TIMES THE
0315 AA                      TAX            ;VARIABLE INDEX
0316 A5 A9                   LDA VLOC+1     ;IF CURRENT  VARIABLE IS NOT
0318 D5 6C                   CMP $6C,X      ;BEYOND THE END OF THE
031A 90 11                   BCC STRTVP     ;STORAGE SPACE FOR THE
031C D0 06                   BNE INCVT      ;CURRENT VARIABLE TYPE,
031E A5 A8                   LDA VLOC       ;THEN GO ON TO VARIABLE
0320 D5 6B                   CMP $6B,X      ;PROCESSING
0322 90 09                   BCC STRTVP
0324 E6 AA          INCVT    INC VTYPE      ;INCREMENT VARIABLE TYPE
0326 A4 AA                   LDY VTYPE
0328 C0 02                   CPY #$02
032A D0 E4                   BNE TOP        ;GO BACK TO THE TOP IF INDEX<>
2
032C 60                      RTS            ;QUIT IF INDEX=2
032D A6 AA          STRTVP   LDX VTYPE      ;START OF VARIABLE PROCESSING
032F F6 FD                   INC NSIMPL,X   ;INCREMENT VARIABLE COUNT
0331 A2 00                   LDX #$00       ;BLANK OUT CURRENT VARIABLE
0333 A9 20                   LDA #$20       ;NAME
0335 95 A5          BLNKVN   STA VNAME,X
0337 E8                      INX
0338 E0 03                   CPX #$03
033A D0 F9                   BNE BLNKVN
033C A0 00                   LDY #$00       ;IF BIT 7 IS OFF, THEN
033E B1 A8                   LDA (VLOC),Y   ;SKIP INTEGER PROCESSING
0340 C9 7F                   CMP #$7F
0342 90 18                   BCC SAVE1
0344 A2 25                   LDX #$25       ;ATTACH '%' TO NAME
0346 86 A7                   STX VNAME+2
0348 29 7F                   AND #$7F       ;SAVE 1ST CHARACTER
034A 85 A5                   STA VNAME
034C C8                      INY            ;STRIP BIT 7 FROM 2ND CHARACTE
R
034D B1 A8                   LDA (VLOC),Y   ;AND SAVE IF NOT $00
034F 29 7F                   AND #$7F
0351 C9 00                   CMP #$00
0353 F0 1C                   BEQ LOWER
0355 85 A6                   STA VNAME+1
0357 18                      CLC            ;SKIP STRING PROCESSING
```

```
0358 90 17               BCC LOWER
035A 90 B4     RELAY     BCC TOP           ;RELAY RETURN TO TOP
035C 85 A5     SAVE1     STA VNAME         ;SAVE 1ST CHARACTER
035E C8                  INY               ;GET 2ND
035F B1 A8               LDA (VLOC),Y
0361 C9 7F               CMP #$7F          ;IF BIT 7 IS OFF, THEN
0363 90 06               BCC SAVE2         ;SKIP STRING PROCESSING
0365 A2 24               LDX #$24          ;ATTACH '$' TO NAME
0367 86 A7               STX VNAME+2
0369 29 7F               AND #$7F          ;STRIP BIT 7
036B C9 00     SAVE2     CMP #$00          ;SAVE 2ND CHARACTER IF NOT
                                            ZERO
036D F0 02               BEQ LOWER
036F 85 A6               STA VNAME+1
0371 38        LOWER     SEC               ;LOWER START OF STRING
0372 A5 6F               LDA $6F           ;STORAGE AREA BY 5
0374 E9 05               SBC #$05
0376 85 6F               STA $6F
0378 A5 70               LDA $70
037A E9 00               SBC #$00
037C 85 70               STA $70
037E A0 00               LDY #$00          ;MOVE VARIABLE DESCRIPTION
0380 B9 A5 00  MOVE      LDA VNAME,Y       ;TO STRING STORAGE
0383 91 6F               STA ($6F),Y
0385 C8                  INY
0386 C0 05               CPY #$05
0388 D0 F6               BNE MOVE
038A A5 AA               LDA VTYPE         ;IF CURRENT VARIABLE TYPE=1
038C C9 01               CMP #$01          ;(I.E., AN ARRAY VARIABLE)
038E F0 10               BEQ INCPTR        ;SKIP SIMPLE VARIABLE
0390 18                  CLC               ;INCREMENT CURRENT VARIABLE
0391 A5 A8               LDA VLOC          ;LOCATION BY 7 AND GO ON
0393 69 07               ADC #$07          ;TO THE NEXT VARIABLE
0395 85 A8               STA VLOC
0397 A5 A9               LDA VLOC+1
0399 69 00               ADC #$00
039B 85 A9               STA VLOC+1
039D 18                  CLC
039E 90 11               BCC GETNXT
03A0 A0 02     INCPTR    LDY #$02          ;INCREMENT CURRENT VARIABLE
03A2 18                  CLC               ;LOCATION BY THE LENGTH
03A3 A5 A8               LDA VLOC          ;OF THE CURRENT ARRAY
03A5 71 A8               ADC (VLOC),Y      ;AND GO ON TO THE
03A7 AA                  TAX               ;NEXT ARRAY VARIABLE
03A8 A5 A9               LDA VLOC+1
03AA C8                  INY
03AB 71 A8               ADC (VLOC),Y
03AD 85 A9               STA VLOC+1
03AF 86 A8               STX VLOC
03B1 18        GETNXT    CLC               ;GO ON TO THE NEXT
03B2 90 A6               BCC RELAY         ;VARIABLE
03B4                     END

***** END OF ASSEMBLY
```

```
SYMBOL TABLE SORTED ALPHABETICALLY

  BLNKVN   0335  GETNXT  03B1  INCPTR  03A0  INCVT   0324  LOWER   0371
  MOVE     0380  NARRAY  00FE  NSIMPL  00FD  RELAY   035A  SAVE1   035C
  SAVE2    036B  STRTVP  032D  TOP     0310  VLOC    00A8  VNAME   00A5
  VTYPE    00AA

SYMBOL TABLE SORTED BY ADDRESS

  VNAME    00A5  VLOC    00A8  VTYPE   00AA  NSIMPL  00FD  NARRAY  00FE
  TOP      0310  INCVT   0324  STRTVP  032D  BLNKVN  0335  RELAY   035A
  SAVE1    035C  SAVE2   036B  LOWER   0371  MOVE    0380  INCPTR  03A0
  GETNXT   03B1
```

```
0800                ;************************
0800                ;*                      *
0800                ;*  SHELL-METZNER SORT  *
0800                ;*   RICHARD ALBRIGHT   *
0800                ;*                      *
0800                ;*        SORTER        *
0800                ;*                      *
0800                ;*  COPYRIGHT(C), 1982  *
0800                ;*    MICRO INK, INC.   *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                      *
0800                ;************************
0800                ;
0800                ;
0019                RL       EPZ $19        ;RECORD LENGTH
001A                KEYOFF   EPZ $1A        ;KEY OFFSET FROM START OF RECO
RD
001B                KEYLEN   EPZ $1B        ;KEY LENGTH
001C                N        EPZ $1C        ;NUMBER OF RECORDS IN $1C-$1D
001E                ARRAY    EPZ $1E        ;POINTER TO ARRAY IN $1E-$1F
00C9                KEYEND   EPZ $C9        ;OFFSET OF LAST KEY BYTE
00CA                I        EPZ $CA        ;INDEX I IN $CA-$CB
00CC                L        EPZ $CC        ;INDEX L IN $CC-$CD
00CE                M        EPZ $CE        ;INDEX M IN $CE-$CF
00DA                K        EPZ $DA        ;INDEX K IN $DA-$DB
00DC                J        EPZ $DC        ;INDEX J IN $DC-$DD
00FA                CNT1     EPZ $FA        ;TEMPORARY COUNTERS IN $FA-$FF

00FC                CNT2     EPZ $FC
00FE                CNT3     EPZ $FE
0800                ;
0300                         ORG $300
0300                         OBJ $800
0300                ;
0300 18                      CLC            ;ESTABLISH OFFSET OF LAST
0301 A5 1A                   LDA KEYOFF     ;KEY BYTE
0303 65 1B                   ADC KEYLEN
0305 85 C9                   STA KEYEND
0307 A5 1C                   LDA N          ;INITIALIZE M TO N
0309 85 CE                   STA M
030B A5 1D                   LDA N+1
030D 85 CF                   STA M+1
030F 18             LOOP1    CLC            ;TOP OF MAIN LOOP
0310 66 CF                   ROR M+1        ;M:=M/2
0312 66 CE                   ROR M
0314 A5 CE                   LDA M          ;STOP IF M=0
0316 D0 05                   BNE MORE
0318 A5 CF                   LDA M+1
031A D0 01                   BNE MORE
031C 60                      RTS
031D A2 00          MORE     LDX #$00       ;K:=N-M
031F 38                      SEC
0320 A5 1C                   LDA N
0322 E5 CE                   SBC M
0324 85 DA                   STA K
0326 A5 1D                   LDA N+1
0328 E5 CF                   SBC M+1
032A 85 DB                   STA K+1
032C A9 01                   LDA #$01       ;J:=1
032E 85 DC                   STA J
0330 A9 00                   LDA #$00
0332 85 DD                   STA J+1
0334 A5 DC          LOOP2    LDA J          ;I:=J
0336 85 CA                   STA I
0338 A5 DD                   LDA J+1
033A 85 CB                   STA I+1
033C 18             LOOP3    CLC            ;L:=I+M
033D A5 CA                   LDA I
033F 65 CE                   ADC M
0341 85 CC                   STA L
0343 A5 CB                   LDA I+1
0345 65 CF                   ADC M+1
0347 85 CD                   STA L+1
0349 A2 00                   LDX #$00       ;SET X REGISTER TO 0
```

```
034B A4 19      GETLOC   LDY RL          ;SET Y REGISTER TO RECORD LENG
TH
034D 38                  SEC             ;INITIALIZE CNT2 TO I-1
034E B5 CA               LDA I,X         ;IF X=0
0350 E9 01               SBC #$01        ;INITIALIZE CNT3 TO L-1
0352 85 FA               STA CNT1        ;IF X=2
0354 95 FC               STA CNT2,X      ;AND STORE THE SAME
0356 B5 CB               LDA I+1,X       ;VALUE IN CNT1
0358 E9 00               SBC #$00
035A 85 FB               STA CNT1+1
035C 95 FD               STA CNT2+1,X
035E 88         GETOFF   DEY             ;MULTIPLY BY RECORD LENGTH TO
035F F0 16               BEQ GETABS      ;GET THE OFFSET OF THE
0361 18                  CLC             ;(I-1)TH RECORD (IF X=0) OR TH
E
0362 A5 FA               LDA CNT1        ;(L-1)TH RECORD (IF X=2) FROM
0364 75 FC               ADC CNT2,X      ;THE START OF THE ARRAY
0366 95 FC               STA CNT2,X
0368 A5 FB               LDA CNT1+1
036A 75 FD               ADC CNT2+1,X
036C 95 FD               STA CNT2+1,X
036E 90 EE               BCC GETOFF
0370 00                  BRK             ;BREAK ON OVERFLOW
0371 D0 C9      RELAY3   BNE LOOP3       ;RELAY RETURNS
0373 90 BF      RELAY2   BCC LOOP2
0375 D0 98      RELAY1   BNE LOOP1
0377 18         GETABS   CLC             ;ADD LOCATION OF START
0378 A5 1E               LDA ARRAY       ;OF ARRAY TO GET ABSOLUTE
037A 75 FC               ADC CNT2,X      ;LOCATION OF (I-1)TH OR
037C 95 FC               STA CNT2,X      ;(L-1)TH RECORD
037E A5 1F               LDA ARRAY+1
0380 75 FD               ADC CNT2+1,X
0382 95 FD               STA CNT2+1,X
0384 E8                  INX             ;ADD 2 TO X REGISTER
0385 E8                  INX
0386 E0 04               CPX #$04        ;GO GET (L-1)TH RECORD
0388 D0 C1               BNE GETLOC      ;IF X=2
038A A4 1A               LDY KEYOFF      ;SET Y REGISTER TO KEY OFFSET
038C B1 FC      COMPAR   LDA (CNT2),Y    ;COMPARE (I-1)TH AND
038E D1 FE               CMP (CNT3),Y    ;(L-1)TH KEY VALUES;
0390 90 09               BCC SWITCH      ;SWITCH RECORDS IF THE
0392 D0 2F               BNE INCJ        ;(L-1)TH KEY IS > THE
0394 C8                  INY             ;(I-1)TH KEY
0395 C4 C9               CPY KEYEND
0397 D0 F3               BNE COMPAR
0399 F0 28               BEQ INCJ
039B A4 19      SWITCH   LDY RL
039D 88         SW1      DEY
039E B1 FC               LDA (CNT2),Y
03A0 AA                  TAX
03A1 B1 FE               LDA (CNT3),Y
03A3 91 FC               STA (CNT2),Y
03A5 8A                  TXA
03A6 91 FE               STA (CNT3),Y
03A8 C0 00               CPY #$00
03AA D0 F1               BNE SW1
03AC 38                  SEC             ;I:=I-M
03AD A5 CA               LDA I
03AF E5 CE               SBC M
03B1 85 CA               STA I
03B3 A5 CB               LDA I+1
03B5 E5 CF               SBC M+1
03B7 85 CB               STA I+1
03B9 A5 CB               LDA I+1         ;BRANCH ON I<1
03BB 30 06               BMI INCJ
03BD D0 B2               BNE RELAY3


03BF A5 CA               LDA I
03C1 D0 AE               BNE RELAY3
03C3 E6 DC      INCJ     INC J           ;J:=J+1
03C5 D0 02               BNE INCJ2
03C7 E6 DD               INC J+1
03C9 A5 DD      INCJ2    LDA J+1         ;BRANCH ON J>K
03CB C5 DB               CMP K+1
```

```
03CD 90 A4          BCC RELAY2
03CF D0 A4          BNE RELAY1
03D1 A5 DC          LDA J
03D3 C5 DA          CMP K
03D5 90 9C          BCC RELAY2
03D7 18             CLC
03D8 F0 99          BEQ RELAY2
03DA D0 99          BNE RELAY1
03DC                END
```

SYMBOL TABLE SORTED ALPHABETICALLY

ARRAY	001E	CNT1	00FA	CNT2	00FC	CNT3	00FE	COMPAR	038B
GETABS	0376	GETLOC	034B	GETOFF	035E	I	00CA	INCJ	03C2
INCJ2	03C8	J	00EB	K	00D6	KEYEND	00C9	KEYLEN	001B
KEYOFF	001A	L	00CC	LOOP1	030F	LOOP2	0334	LOOP3	033C
M	00CE	MORE	031D	N	001C	RELAY1	0374	RELAY2	0372
RELAY3	0370	RL	0019	SW1	039C	SWITCH	039A		

SYMBOL TABLE SORTED BY ADDRESS

RL	0019	KEYOFF	001A	KEYLEN	001B	N	001C	ARRAY	001E
KEYEND	00C9	I	00CA	L	00CC	M	00CE	K	00D6
J	00EB	CNT1	00FA	CNT2	00FC	CNT3	00FE	LOOP1	030F
MORE	031D	LOOP2	0334	LOOP3	033C	GETLOC	034B	GETOFF	035E
RELAY3	0370	RELAY2	0372	RELAY1	0374	GETABS	0376	COMPAR	038B
SWITCH	039A	SW1	039C	INCJ	03C2	INCJ2	03C8		

2
MACHINE-LANGUAGE AIDS

Machine-Language Aids

This chapter contains four utility programs designed to make life a little easier for the assembly-language programmer.

David Rosenberg's "Double Barrelled Disassembler" not only prints a hard copy disassembly two abreast, but also gives the user the ability to specify a precise range of memory without disassembling in increments of 20 instructions. "Cross Referencing 6502 Programs" by Cornelis Bongers is an indispensible tool for anyone interested in analyzing code through disassembly. With this program the user can study all address references within a range of code, either external or internal.

Wes Huntress's "Fast Fractional Math Package" provides the assembly language programmer with a tool bag of fractional math functions. For the BASIC programmer, it gives Integer BASIC the ability to perform complex functions and speeds up Applesoft at the cost of some accuracy. "Applesoft Error Messages from Machine Language" by Steve Cochard describes how to access Applesoft error messages from machine language and offers programming examples on interfacing with error message subroutines.

Double Barrelled Disassembler

by David L. Rosenberg

This short utility makes it easier to create disassembly listings. It not only lists from starting to ending addresses, but also formats the listing into two columns for easier reading and less paper usage.

How many L's are there between $BD00 and $BFFF? What seems at first to be a ridiculous question actually points out one of the few flaws in the Apple II's ROM Monitor: the disassembler routine only prints twenty lines at a time. This can be a major annoyance if you are printing many long listings.

This program attacks the problem and formats the listing into two columns to minimize wasted paper and make the disassembly easier to follow. Once the program has been BRUN the disassembly function is called by typing "beginning address"."ending address" (CTRL – Y) return. This sequence will disassemble the code from the beginning address through the ending address and print it in two column per page format (see listing 1).

How Does it Work?

The program divides the first part of the object code into two segments, each containing the same number of instructions as there are lines on a page. Then it takes one instruction from each piece and calls the Monitor disassembly routine to print them on the same line. Next, the pointers to the instructions are incremented and the program loops to the disassembly portion again. When all the instructions in each segment are done, a form-feed is printed and the next portion of the code is segmented, and the process is repeated until the ending address is reached.

The only problem I encountered was that the Monitor disassembly routine prints a carriage return as the first character each time it is called. Obviously this is not desirable after we go to the trouble of positioning the printer to the start of the second column. To circumvent this the disassembler is called in four separate pieces.

PR1 is called to print the address in the Program Counter ($3A,$3B) as four ASCII bytes followed by a dash. PR2 points PC at the length of the instruction and forms an index into the Monitor's op-code mnemonic table. PR3 actually prints the mnemonic along with the appropriate address or hex literal. At this point we must push a $01 onto the stack to indicate that this is the last instruction to disassemble. PR4 increments PC to point to the next instruction then pulls the top value from the stack, decrements it by one and if it is equal to zero, does a return. Since PR4 is jumped to, this return will take us back to the mainline where the program sets up to disassemble the corresponding instruction from column two.

Before calling the Monitor disassembler, PC must contain the address of the instruction to be disassembled. Since we are disassembling and printing two non-sequential instructions on each line, a large part of the program is concerned with swapping instruction addresses in and out of PC. A4 ($42,$43) is used as a work byte to store the column one address when the second column is being disassembled. A3 ($40,$41) serves a similar function when the first column is being disassembled. A2 ($3E,$3F) always contains the ending address of the code to be disassembled.

The subroutine INITA3 calls a Monitor routine at $F88E to return the length of an instruction. The whole purpose of the routine is to find the address of the *nth* + 1 instruction, where *n* is the number of lines per page. This is also the start of column two, so we want this address to wind up in A3. To accomplish this we will call INSDS2 *n* times and add the resulting length to the address at A3. Note that the length returned is actually one less than the actual instruction length, and therefore, we must increment LEN before adding it to A3. Invalid op-codes are not flagged, but are returned as one-byte length instructions.

To end execution, routine CMPCA2 compares the current value of PC to the value of A2 (the end address). If it is equal to, or greater than A2, we pop the last return address from the stack and jump to UNHOOK. This effectively disconnects from the mainline and resets the stack to the condition it was at when the disassembler was first invoked. Because the program is called from monitor, the RTS in UNHOOK will result in a return to monitor.

Making it Work

This program should be used with an AIO serial card in slot #1 and a Texas Instruments 810 printer. The routine STHOOK sets the DOS output hooks and disables the serial card's video echo. If your interface is in a different slot, change the LDX instruction at line 89. It is of the format C*n*, where *n* is the slot number. For printers with a software-selectable line width this would be the best place to include the code for this function. The routine UNHOOK, always the last one executed, is where you should reset the line width.

The first instruction in the routine TAB controls how far over (in print positions) the second column will start. This can be changed to ½ of the line width that you are using (i.e., $28 for an 80-column line). The number of lines per page is set in two places, line 118 and line 177. It can be set to suit your needs, but just be sure it is the same in both places.

If your printer does not recognize $0C as a form-feed character or does not have a formfeed, the routine FFE ED will have to be changed. This routine makes the printer skip to the top of the next page.

Since the program uses standard Apple output routines it can be used, as is, with any printer card (serial or parallel) that does not require a software driver. If you use a print driver routine, change the JSRs at lines 66, 79, 85 and 93 to go to your driver entry point. The character to be printed will reside in the Accumulator prior to these calls.

Editor's note: Listing 2 is an example of how the Double Barrelled Disassembler can be modified for other card/printer combinations. Here the program was modified to work with Apple's serial Interface Card and a Bedford Computer Systems, Inc., daisy wheel printer.

Listing 1

```
0800              ;************************
0800              ;*                      *
0800              ;*   DOUBLE BARRELLED   *
0800              ;*     DISASSEMBLER     *
0800              ;*          BY          *
0800              ;*  DAVID L. ROSENBURG  *
0800              ;*                      *
0800              ;*       DISSASMB       *
0800              ;*                      *
0800              ;*  COPYRIGHT (C) 1982  *
0800              ;*    MICRO INK, INC.   *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                      *
0800              ;************************
0800              ;
0800              ;
0024              CH        EPZ $24          ;CURSOR HORIZONTAL POSN
002F              LEN       EPZ $2F          ;INSTRUCTION LENGTH
003A              PC        EPZ $3A          ;ADDRESS TO DISSASSEMBLE
003E              A2        EPZ $3E          ;ENDING ADDRESS
0040              A3        EPZ $40          ;ADDRESS TO DISASSEMBLE
0042              A4        EPZ $42          ;WORK BYTE
0045              A5        EPZ $45          ;LINE COUNTER
03F8              VECTOR    EQU $3F8         ;CTRL-Y VECTOR ADDRESS
0579              NOVID     EQU $579         ;AIO SERIAL CARD NOVID FLG
05F9              COL       EQU $5F9         ;SERIAL INTER. CARD COLUMN NO.
06F9              PWDTH     EQU $6F9         ;SERIAL INTER. CARD LINE WIDTH
AA53              HOOKS     EQU $AA53        ;OUTPUT HOOK
F88E              INSDS2    EQU $F88E        ;ROUTINE FOR INSTRUC. LENGTH
FDED              PRINT     EQU $FDED        ;MONITOR COUT ROUTINE
FD99              PR1       EQU $FD99        ;PART OF DISASSEMBLER (ROM)
F889              PR2       EQU $F889        ;PART OR DISASSEMBLER (ROM)
F8D3              PR3       EQU $F8D3        ;PART OF DISASSEMBLER (ROM)
FE67              PR4       EQU $FE67        ;PART OF DISASSEMBLER (ROM)
0800              ;
0800                        ORG $800
0800              ;
0800              ;***********************
0800              ;THIS ROUTINE SETS THE APPLE'S CTRL-Y VECTOR ADDRESS
0800              ;TO POINT TO THE START OF THE DISASSEMBLER CODE
0800              ;IT IS EXECUTED WHEN THE PROGRAM IS BRUN
0800              ;***********************
0800              ;
0800 A9 4C        INIT      LDA #$4C         ;OP CODE FOR JUMP
0802 8D F8 03               STA VECTOR       ;STORE AT CTRL-Y VECTOR
0805 A9 10                  LDA #START       ;GET LOW BYTE OF ENTRY LOCATIO
N
0807 8D F9 03               STA VECTOR+1     ;STORE AT VECTOR
080A A9 08                  LDA /START       ;GET HI BYTE OF ENTRY LOCATION

080C 8D FA 03               STA VECTOR+2
080F 60                     RTS
0810              ;
0810              ;******************************
0810              ;    START OF DISASSEMBLER
0810              ;******************************
0810              ;
0810 20 62 08     START     JSR STHOOK       ;SET OUTPUT HOOKS FOR PRINTER
0813 20 87 08     MAIN      JSR SETPC        ;SET PC TO A3
0816 20 99 08               JSR SETA5        ;SET A5 TO # OF LINES PER PAGE

0819 20 E9 08               JSR INITA3       ;SET A3 TO START OF COLUMN 2
081C 20 9E 08     LOOP      JSR CMPCA2       ;COMPARE PC TO END ADDRESS
081F 20 D6 08               JSR DISASM       ;DISASSEMBLE INSTRUCTION AT PC

0822 20 B7 08               JSR CMA3A2       ;COMPARE A3 TO END ADDRESS
0825 B0 12                  BCS LOOP2        ;DON'T PRINT 2ND COLUMN IF >
0827 20 C4 08               JSR STORPC       ;SAVE PC AT A4
082A 20 87 08               JSR SETPC        ;SET PC TO A3
082D 20 48 08               JSR TAB
0830 20 D6 08               JSR DISASM       ;DISASSEMBLE INSTRUCTION AT PC
 (=A3)
```

```
0833 20 90 08                JSR SETA3          ;SET A3 TO PC
0836 20 CD 08                JSR RSTRPC         ;SET PC TO A4
0839 A9 0D        LOOP2      LDA #$0D
083B 20 ED FD                JSR PRINT          ;PRINT CARRIAGE RETURN
083E C6 45                   DEC A5             ;DECREMENT LINE COUNTER
0840 D0 DA                   BNE LOOP           ;IF NOT END OF PAGE
0842 20 5C 08                JSR FFEED          ;ADVANCE TO NEXT PAGE
0845 4C 13 08                JMP MAIN
0848 A9 42        TAB        LDA #$42           ;SET X-REG TO
084A 38                      SEC                ;66-CURSOR POSITION
084B E5 24                   SBC CH             ;I.E. #OF SPACES TO PRINT
084D AA                      TAX                ;TILL MIDDLE OF PAGE
084E F0 0B        T1         BEQ TX
0850 30 09                   BMI TX
0852 A9 A0                   LDA #$A0
0854 20 ED FD                JSR PRINT          ;PRINT SPACES TILL
0857 CA                      DEX                ;X-REG=0
0858 4C 4E 08                JMP T1
085B 60           TX         RTS
085C A9 0C        FFEED      LDA #$0C           ;PRINT FORM FEED
085E 20 ED FD                JSR PRINT
0861 60                      RTS
0862 A0 00        STHOOK     LDY #$00           ;SET THE OUTPUT HOOK
0864 A2 C1                   LDX #$C1           ;TO C100 (SLOT 1)
0866 8E 54 AA                STX HOOKS+1
0869 8C 53 AA                STY HOOKS
086C A9 8D                   LDA #$8D           ;PRINT CARRIAGE RETURN TO
086E 20 ED FD                JSR PRINT          ;INITIALIZE SERIAL CARD
0871 A9 80                   LDA #$80
0873 8D 79 05                STA NOVID          ;NO VIDEO MOD
0876 60                      RTS
0877 A9 00        UNHOOK     LDA #$00           ;RESET VIDEO MOD
0879 A0 F0                   LDY #$F0           ;AND RESTORE OUTPUT
087B A2 FD                   LDX #$FD           ;HOOKS TO SCREEN
087D 8D 79 05                STA NOVID
0880 8C 53 AA                STY HOOKS
0883 8E 54 AA                STX HOOKS+1
0886 60                      RTS
0887 A5 40        SETPC      LDA A3             ;SET PC TO A3
0889 85 3A                   STA PC
088B A5 41                   LDA A3+1
088D 85 3B                   STA PC+1
088F 60                      RTS
0890 A5 3A        SETA3      LDA PC             ;SET A3 TO PC
0892 85 40                   STA A3
0894 A5 3B                   LDA PC+1
0896 85 41                   STA A3+1
0898 60                      RTS
0899 A9 3C        SETA5      LDA #$3C           ;INITIALIZE LINE COUNTER TO
089B 85 45                   STA A5             ;60 --- COUNTS DOWN
089D 60                      RTS
089E A5 3B        CMPCA2     LDA PC+1           ;COMPARE HI BYTE OF PC TO
08A0 C5 3F                   CMP A2+1           ;HI BYTE OF A2 (END ADDR)
08A2 90 12                   BCC C2             ;<RETURN
08A4 F0 05                   BEQ C1             ;=COMPARE LOW BYTES
08A6 68                      PLA                ;POP RETURN ADDRESS
08A7 68                      PLA                ;OFF THE STACK
08A8 4C 77 08                JMP UNHOOK         ;RESET HOOKS AND QUIT
08AB A5 3A        C1         LDA PC             ;COMPARE LOW BYTES
08AD C5 3E                   CMP A2
08AF 90 05                   BCC C2             ;RETURN
08B1 68                      PLA                ;POP STACK
08B2 68                      PLA
08B3 4C 77 08                JMP UNHOOK         ;RESET AND QUIT
08B6 60           C2         RTS
08B7 A5 41        CMA3A2     LDA A3+1           ;COMPARE A3 AND A2
08B9 C5 3F                   CMP A2+1           ;RETURN WITH CARRY BIT
08BB 90 06                   BCC CMA2           ;SET OR CLEAR TO
08BD D0 04                   BNE CMA2           ;INDICATE STATUS

08BF A5 40                   LDA A3
08C1 C5 3E                   CMP A2
08C3 60           CMA2       RTS
08C4 A5 3A        STORPC     LDA PC
08C6 85 42                   STA A4             ;SAVE CURRENT VALUE OF PC
08C8 A5 3B                   LDA PC+1
```

```
08CA 85 43              STA A4+1
08CC 60                 RTS
08CD A5 42    RSTRPC    LDA A4              ;RESTORE PC FROM CURRENT
08CF 85 3A              STA PC              ;VALUE OF A4
08D1 A5 43              LDA A4+1
08D3 85 3B              STA PC+1
08D5 60                 RTS
08D6 A6 3A    DISASM    LDX PC              ;DISASSEMBLE 1 INSTRUCTION
08D8 A4 3B              LDY PC+1            ;AT PC USING MONITOR
08DA 20 99 FD           JSR PR1             ;DISASSEMBLE ROUTINE
08DD 20 89 F8           JSR PR2             ;IN FOUR PARTS
08E0 20 D3 F8           JSR PR3
08E3 A9 01              LDA #$01            ;SET COUNTER ON STACK FOR
08E5 48                 PHA                 ;NUMBER OF INSTRUCTIONS
08E6 4C 67 FE           JMP PR4             ;ROUTINE SUPPLIES RTS
08E9          ;
08E9          ;*****************************
08E9          ;THIS ROUTINE CALCULATES THE ADDRESS OF THE
08E9          ;FIRST INSTRUCTION IN COLUMN TWO
08E9          ;*****************************
08E9          ;
08E9 A2 3C    INITA3    LDX #$3C            ;NUMBER OF INSTRUCTIONS
08EB A0 00    INIT41    LDY #$00            ;SET INDEX POINTER
08ED 8A                 TXA                 ;SAVE NUMBER OF
08EE 48                 PHA                 ;INSTRUCTIONS ON STACK
08EF B1 40              LDA (A3),Y          ;GET OP CODE
08F1 20 8E F8           JSR INSDS2          ;MONITOR ROUTINE FOR LENGTH
08F4 E6 2F              INC LEN
08F6 A5 40              LDA A3              ;GET A3 AND
08F8 18                 CLC                 ;INCREMENT BY
08F9 65 2F              ADC LEN             ;LENGTH OF INSTRUCTION
08FB 85 40              STA A3              ;SAVE IN A3
08FD 90 02              BCC INIT42          ;INCREMENT HI BYTE
08FF E6 41              INC A3+1            ;IF NECESSARY
0901 68       INIT42    PLA                 ;GET NUMBER OF INSTRUCTIONS
0902 AA                 TAX
0903 CA                 DEX                 ;SUBTRACT 1
0904 D0 E5              BNE INIT41          ;LOOP IF NOT DONE
0906 60                 RTS
0907                    END
```

Listing 2

```
0800              ;************************
0800              ;*                      *
0800              ;*   DOUBLE BARRELLED   *
0800              ;*     DISASSEMBLER     *
0800              ;*          BY          *
0800              ;*  DAVID L. ROSENBURG  *
0800              ;*                      *
0800              ;*  MODIFIED BY T.S.O.  *
0800              ;*     TO WORK WITH     *
0800              ;*   THE APPLE SERIAL   *
0800              ;*    INTERFACE CARD    *
0800              ;*                      *
0800              ;*     DISASSMB-SC      *
0800              ;*                      *
0800              ;*  COPYRIGHT (C) 1982  *
0800              ;*    MICRO INK, INC.   *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                      *
0800              ;************************
0800              ;
0024              CH       EPZ $24         ;CURSOR HORIZONTAL POSN
002F              LEN      EPZ $2F         ;INSTRUCTION LENGTH
003A              PC       EPZ $3A         ;ADDRESS TO DISSASSEMBLE
003E              A2       EPZ $3E         ;ENDING ADDRESS
0040              A3       EPZ $40         ;ADDRESS TO DISASSEMBLE
0042              A4       EPZ $42         ;WORK BYTE
0045              A5       EPZ $45         ;LINE COUNTER
03F8              VECTOR   EQU $3F8        ;CTRL-Y VECTOR ADDRESS
0579              NOVID    EQU $579        ;AIO SERIAL CARD NOVID FLG
05F9              COL      EQU $5F9        ;SERIAL INTER. CARD COLUMN NO.

06F9              PWDTH    EQU $6F9        ;SERIAL INTER. CARD LINE WIDTH

AA53              HOOKS    EQU $AA53       ;OUTPUT HOOK
F88E              INSDS2   EQU $F88E       ;ROUTINE FOR INSTRUC. LENGTH
FDED              PRINT    EQU $FDED       ;MONITOR COUT ROUTINE
FD99              PR1      EQU $FD99       ;PART OF DISASSEMBLER (ROM)
F889              PR2      EQU $F889       ;PART OR DISASSEMBLER (ROM)
F8D3              PR3      EQU $F8D3       ;PART OF DISASSEMBLER (ROM)
FE67              PR4      EQU $FE67       ;PART OF DISASSEMBLER (ROM)
0800              ;
0800                       ORG $800
0800              ;
0800              ;************************
0800              ;THIS ROUTINE SETS THE APPLE'S CTRL-Y VECTOR ADDRESS
0800              ;TO POINT TO THE START OF THE DISASSEMBLER CODE
0800              ;IT IS EXECUTED WHEN THE PROGRAM IS BRUN
0800              ;************************
0800              ;
0800 A9 4C        INIT     LDA #$4C        ;OP CODE FOR JUMP
0802 8D F8 03              STA VECTOR      ;STORE AT CTRL-Y VECTOR
0805 A9 10                 LDA #START      ;GET LOW BYTE OF ENTRY LOCATIO
N
0807 8D F9 03              STA VECTOR+1    ;STORE AT VECTOR
080A A9 08                 LDA /START      ;GET HI BYTE OF ENTRY LOCATION

080C 8D FA 03              STA VECTOR+2
080F 60                    RTS
0810              ;
0810              ;******************************
0810              ;    START OF DISASSEMBLER
0810              ;******************************
0810              ;
0810 20 71 08     START    JSR STHOOK      ;SET OUTPUT HOOKS FOR PRINTER
0813 20 96 08     MAIN     JSR SETPC       ;SET PC TO A3
0816 20 A8 08              JSR SETA5       ;SET A5 TO # OF LINES PER PAGE

0819 20 F8 08              JSR INITA3      ;SET A3 TO START OF COLUMN 2
081C 20 AD 08     LOOP     JSR CMPCA2      ;COMPARE PC TO END ADDRESS
081F 20 E5 08              JSR DISASM      ;DISASSEMBLE INSTRUCTION AT PC

0822 20 C6 08              JSR CMA3A2      ;COMPARE A3 TO END ADDRESS
```

```
0825 B0 12                  BCS LOOP2      ;DON'T PRINT 2ND COLUMN IF >
0827 20 D3 08               JSR STORPC     ;SAVE PC AT A4
082A 20 96 08               JSR SETPC      ;SET PC TO A3
082D 20 52 08               JSR TAB
0830 20 E5 08               JSR DISASM     ;DISASSEMBLE INSTRUCTION AT
                                            PC (=A3)
0833 20 9F 08               JSR SETA3      ;SET A3 TO PC
0836 20 DC 08               JSR RSTRPC     ;SET PC TO A4
0839 A9 0D         LOOP2    LDA #$0D
083B 20 ED FD               JSR PRINT      ;PRINT CARRIAGE RETURN
083E A9 0A                  LDA #$0A
0840 20 ED FD               JSR PRINT      ;PRINT LINE FEED
0843 A9 00                  LDA #$00
0845 8D F9 05               STA COL        ;RESET COLUMN PARAMETER
0848 C6 45                  DEC A5         ;DECREMENT LINE COUNTER
084A D0 D0                  BNE LOOP       ;IF NOT END OF PAGE
084C 20 6B 08               JSR FFEED      ;ADVANCE TO NEXT PAGE
084F 4C 13 08               JMP MAIN
0852 AD F9 05      TAB      LDA COL        ;GET COLUMN FROM SERIAL CARD
0855 85 24                  STA CH
0857 A9 42                  LDA #$42       ;SET X-REG TO
0859 38                     SEC            ;66-CURSOR POSITION
085A E5 24                  SBC CH         ;I.E. #OF SPACES TO PRINT
085C AA                     TAX            ;TILL MIDDLE OF PAGE
085D F0 0B         T1       BEQ TX
085F 30 09                  BMI TX
0861 A9 A0                  LDA #$A0
0863 20 ED FD               JSR PRINT      ;PRINT SPACES TILL
0866 CA                     DEX            ;X-REG=0
0867 4C 5D 08               JMP T1
086A 60            TX       RTS
086B A9 0C         FFEED    LDA #$0C       ;PRINT FORM FEED
086D 20 ED FD               JSR PRINT
0870 60                     RTS
0871 A0 00         STHOOK   LDY #$00       ;SET THE OUTPUT HOOK
0873 A2 C1                  LDX #$C1       ;TO C100 (SLOT 1)
0875 8E 54 AA               STX HOOKS+1
0878 8C 53 AA               STY HOOKS
087B A9 8D                  LDA #$8D       ;PRINT CARRIAGE RETURN TO
087D 20 ED FD               JSR PRINT      ;INITIALIZE SERIAL CARD
0880 20 ED FD               JSR PRINT
0883               ;        LDA #$80
0883               ;        STA NOVID      ;NO VIDEO MOD
0883 A9 00                  LDA #$00       ;SHUT OFF FORCED CR'S
0885 8D F9 06               STA PWDTH      ;FROM SERIAL CARD
0888 60                     RTS
0889 A9 00         UNHOOK   LDA #$00       ;RESET VIDEO MOD
088B A0 F0                  LDY #$F0       ;AND RESTORE OUTPUT
088D A2 FD                  LDX #$FD       ;HOOKS TO SCREEN
088F               ;        STA NOVID
088F 8C 53 AA               STY HOOKS
0892 8E 54 AA               STX HOOKS+1
0895 60                     RTS
0896 A5 40         SETPC    LDA A3         ;SET PC TO A3
0898 85 3A                  STA PC
089A A5 41                  LDA A3+1
089C 85 3B                  STA PC+1
089E 60                     RTS
089F A5 3A         SETA3    LDA PC         ;SET A3 TO PC
08A1 85 40                  STA A3
08A3 A5 3B                  LDA PC+1
08A5 85 41                  STA A3+1
08A7 60                     RTS
08A8 A9 3C         SETA5    LDA #$3C       ;INITIALIZE LINE COUNTER TO
08AA 85 45                  STA A5         ;60 --- COUNTS DOWN
08AC 60                     RTS
08AD A5 3B         CMPCA2   LDA PC+1       ;COMPARE HI BYTE OF PC TO
08AF C5 3F                  CMP A2+1       ;HI BYTE OF A2 (END ADDR)
08B1 90 12                  BCC C2         ;<RETURN
08B3 F0 05                  BEQ C1         ;=COMPARE LOW BYTES

08B5 68                     PLA            ;POP RETURN ADDRESS
08B6 68                     PLA            ;OFF THE STACK
08B7 4C 89 08               JMP UNHOOK     ;RESET HOOKS AND QUIT
08BA A5 3A         C1       LDA PC         ;COMPARE LOW BYTES
```

```
08BC C5 3E              CMP A2
08BE 90 05              BCC C2            ;RETURN
08C0 68                 PLA               ;POP STACK
08C1 68                 PLA
08C2 4C 89 08           JMP UNHOOK        ;RESET AND QUIT
08C5 60        C2       RTS
08C6 A5 41     CMA3A2   LDA A3+1          ;COMPARE A3 AND A2
08C8 C5 3F              CMP A2+1          ;RETURN WITH CARRY BIT
08CA 90 06              BCC CMA2          ;SET OR CLEAR TO
08CC D0 04              BNE CMA2          ;INDICATE STATUS
08CE A5 40              LDA A3
08D0 C5 3E              CMP A2
08D2 60        CMA2     RTS
08D3 A5 3A     STORPC   LDA PC
08D5 85 42              STA A4            ;SAVE CURRENT VALUE OF PC
08D7 A5 3B              LDA PC+1
08D9 85 43              STA A4+1
08DB 60                 RTS
08DC A5 42     RSTRPC   LDA A4            ;RESTORE PC FROM CURRENT
08DE 85 3A              STA PC            ;VALUE OF A4
08E0 A5 43              LDA A4+1
08E2 85 3B              STA PC+1
08E4 60                 RTS
08E5 A6 3A     DISASM   LDX PC            ;DISASSEMBLE 1 INSTRUCTION
08E7 A4 3B              LDY PC+1          ;AT PC USING MONITOR
08E9 20 99 FD           JSR PR1           ;DISASSEMBLE ROUTINE
08EC 20 89 F8           JSR PR2           ;IN FOUR PARTS
08EF 20 D3 F8           JSR PR3
08F2 A9 01              LDA #$01          ;SET COUNTER ON STACK FOR
08F4 48                 PHA               ;NUMBER OF INSTRUCTIONS
08F5 4C 67 FE           JMP PR4           ;ROUTINE SUPPLIES RTS
08F8           ;
08F8           ;******************************
08F8           ;THIS ROUTINE CALCULATES THE ADDRESS OF THE
08F8           ;FIRST INSTRUCTION IN COLUMN TWO
08F8           ;******************************
08F8           ;
08F8 A2 3C     INITA3   LDX #$3C          ;NUMBER OF INSTRUCTIONS
08FA A0 00     INIT41   LDY #$00          ;SET INDEX POINTER
08FC 8A                 TXA               ;SAVE NUMBER OF
08FD 48                 PHA               ;INSTRUCTIONS ON STACK
08FE B1 40              LDA (A3),Y        ;GET OP CODE
0900 20 8E F8           JSR INSDS2        ;MONITOR ROUTINE FOR LENGTH
0903 E6 2F              INC LEN
0905 A5 40              LDA A3            ;GET A3 AND
0907 18                 CLC               ;INCREMENT BY
0908 65 2F              ADC LEN           ;LENGTH OF INSTRUCTION
090A 85 40              STA A3            ;SAVE IN A3
090C 90 02              BCC INIT42        ;INCREMENT HI BYTE
090E E6 41              INC A3+1          ;IF NECESSARY
0910 68        INIT42   PLA               ;GET NUMBER OF INSTRUCTIONS
0911 AA                 TAX
0912 CA                 DEX               ;SUBTRACT 1
0913 D0 E5              BNE INIT41        ;LOOP IF NOT DONE
0915 60                 RTS
0916                    END
```

Cross Referencing 6502 Programs

by Cornelis Bongers

This cross reference program facilitates the analysis of 6502 programs by constructing a cross reference table that relates each address that is used to its point of reference.

The variety and quality of software for 6502 systems continues to grow. Now it is not attractive to write certain programs yourself, such as a word processor or advanced game, since the market offers most programs of this kind at reasonable prices. However, there is one flaw in this argument. After you buy a program, you almost always discover that it would have suited your needs if those two (missing) options had been included, or if that nasty bug had been left out.

An example is the flight simulator, a well-known program to Apple owners. The first time I tried to hold that plane in the air while keeping the Germans off my tail, it all seemed very difficult, even impossible. However, after many (entertaining) hours, I finally mastered the game. Then I wondered why there wasn't a second level of play — one for aces. For instance, in the latter version, a restriction could be put on the vertical velocity when the plane is landing. In the current version you land safely, whatever the vertical speed. A plane crash would be more realistic if the vertical velocity exceeds a certain speed at the moment of touchdown.

If you're satisfied with a program except for a few points, you have three options: 1. Do nothing and just live with it. If this is your choice, stop reading and skip the rest of this article. 2. Write your own program and include the missing options while omitting the bugs. However, this decision will not in general be very wise because it will cost you at least a few months (probably much longer) to write. 3. Analyze the program you have and build in the extra options with patches. As you'll see, the crucial part is the analysis of the program. A cross reference table is useful for this, and although it doesn't answer all questions, it saves hours of work on Applesoft analysis.

Analyzing Programs

I will describe some of the experiences I had during the analysis of Applesoft. Since I knew Applesoft started at $E000, I started analyzing at $E000 too. I kept track of all zero page addresses that were used and the values that were stored in them. I also made a list of the called subroutines. Soon I discovered that this process would drive me crazy. After several hours of working, I had a zero page table full of meaningless numbers, not to mention an enormous table of subroutines that were called for unknown reasons.

Just before I decided to give up, I remembered something an experienced programmer once told me: a large program somehow has to analyze its input and thus must have a keyword table. Furthermore, large programs usually contain a number of subroutines which handle the keyword functions. Since keywords can be recognized in a disassembly listing by repeated question marks, I found the keyword table of Applesoft. Because program control must go to a routine handling the keyword function when a keyword is detected, it seemed logical that there also had to be a table of subroutine entry addresses.

After scanning through the listing, I found this table right before the keyword table. A few hours later I had the subroutine entries in the listing marked with the keywords and called it a day, thinking that the rest would be simple. Wrong! The next day I discovered the addresses where the SPEED and ROT bytes are stored. There was no progress because I still couldn't keep track of the program flow since too many subroutines were called from the keyword handling routines.

A similar problem arose with the zero page addresses. Often I suspected that a certain zero page location was used in connection with a specific function only, but I could not check this since it is absolutely impossible to find all the references to a certain address in a listing of 96 pages.

A colleague who had written an x-ref assembler on a Nova 820 computer made a cross reference table of Applesoft for me which solved most of my problems. An x-ref assembler lists all references that are made to an address. By using these references it is possible to trace the program flow in reverse order, making it easy to find the driver (main program) in the program you want to analyze. Furthermore, the references show where in the program a certain zero page (or other) location is used. (This helps to find out the meaning of the values stored in such a location.)

Apart from the references themselves, useful information provided by a cross reference table includes the number of references at a certain address. For instance, if you find a subroutine with more than five references, it is bound to be an important one and it certainly will be worth the trouble to find out what it does. As an example, the cross reference table of a small program is listed in figure 1.

When executing this program it will ask for a BASE. After typing in a number between one and nine, the program will display a counter on the screen which starts counting at zero in a notation with base equal to BASE + 1. The ASCII values listed behind the mnemonics show that the last part of the program contains the text 'BASE (1-9) ?' Since the 'B' from BASE is referred to by address $200D, this will be an address within a routine that displays text.

Another important point is the empty line at $2024. Because this line is referred to by the instruction BNE $2024 at $202D, there must be a hidden instruction at $2024. Hidden instructions are sometimes used (among others in Applesoft) to save a few bytes.

Note that the x-ref assembler lists addresses $A0, $AD, $D3 and $A0A9 as addresses used by the program. However, these addresses appear in the text "BASE (1-9) ?" since the disassembler has translated some text to valid opcodes. The x-ref assembler is thus not able to distinguish opcodes that are more or less randomly generated within text or tables from real opcodes. This means that some of the references listed by the x-ref assembler may not be valid.

Figure 1

```
0005-                                2025
00A0-                                203B
00AD-                                203E
00D3-                                2039
00FE-                                2023
040D-                                2020
05D5-                                2007
05D6-                                202A 2031
2000-2058FC JSR $FC58     X|
2003-A222   LDX #$22     ""
2005-A9B0   LDA #$B0     )0
2007-9DD505 STA $05D5,X   U     200B
200A-CA     DEX          J
200B-D0FA   BNE $2007    Pz
200D-BD3820 LDA $2038,X =8     2016
2010-20EDFD JSR $FDED     m}
2013-E8     INX          h
2014-E00C   CPX #$0C     `
2016-D0F5   BNE $200D    Pu
2018-AC00C0 LDY $C000    , @   201B
201B-10FB   BPL $2018     {
201D-8C10C0 STY $C010      @
2020-8C0D04 STY $040D
2023-A5FE   LDA $FE      %~
2024-                           202D
2025-D605   DEC $05,X    V
2027-A221   LDX #$21     "!
2029-98     TYA                 2035
202A-DDD605 CMP $05D6,X ]V
202D-D0F5   BNE $2024    Pu
202F-A9B0   LDA #$B0     )0
2031-9DD605 STA $05D6,X  V
2034-CA     DEX          J
2035-10F2   BPL $2029     r
2037-60     RTS          `
2038-C2     ???          B      200D
2039-C1D3   CMP ($D3,X) AS
203B-C5A0   CMP $A0      E
203D-A8     TAY          (
203E-B1AD   LDA ($AD),Y 1-
2040-B9A9A0 LDA $A0A9,Y 9)
2043-BF     ???          ?
A0A9-                           2040
C000-                           2018
C010-                           201D
FC58-                           2000
FDED-                           2010
```

The X-REF Assembler Program

Since I consider an x-ref assembler an indispensable software tool for my Apple, I wrote one. The text file of the progam is listed in figure 2.

To run the x-ref assembler, BRUN CROSS ASSEMBLER and give the (monitor) command 800G to initialize the control Y vector. Next, load the binary program that has to be x-reffed, starting at a user-defined location. In the sequel it will be assumed that this is location $1000. In case you load from tape, the monitor MOVE command can be used to "move" the program to this location. After having performed these steps, the x-ref assembler can be executed by the command,

XXXX < YYYY.ZZZZcontrol Y

where

XXXX is the origin of the program that has to be x-reffed.

YYYY is the start address (i.e. $1000) of the program in memory

ZZZZ is the end address of the program in memory.

For instance, if you want to make a cross reference table of ROM Applesoft, it first has to be moved to location $1000 by the command 1000< D000 .F800M. The x-ref assembler then can be executed by the command D000 < 1000.3800 control Y. After having typed in this command (followed by a carriage return) the display should show five figures after a few moments. These are:

Pass The number of passes (including print passes) made thus far.

SAR The start of the address range that is x-reffed during the current pass.

EAR The end of the address range that is x-reffed during the current pass.

TSP A table pointer.

PCU The user's program counter.

To explain these figures, it is necessary to give a brief description of the way the x-ref assembler works. The program starts (after the control Y command) by initializing a table which begins just behind the program that has to be x-reffed and ends at a user-defined location. This table is used to store the references and consists of the format shown in table 1.

Table 1

Memory Location	Address	(Next Address)	(Previous Address)	References
ZZZZ+1	0000	ZZZZ+1+1*0FF1	FFFF	
ZZZZ+1+1*0FF1	FFFF	FFFF	ZZZZ+1	

The table is initialized with the values shown and each entry has a (user-defined) length of 0FF1 bytes. Next, the x-ref assembler starts x-reffing the program, thereby keeping up two program counters. The first program counter (PC) points to the subsequent addresses of the instructions that have to be disassembled in the program starting at $1000, while the second program counter (PCU) points to the corresponding addresses in the original program. The PC and PCU therefore differ by a constant with the value $XXXX-$1000.

Suppose now that the first instruction that is being disassembled is LDA $00. The x-ref assembler then searches the table to see whether address $00 is present already. Since this is the case, it stores the current value of PCU, say $3000, as a reference at the entry of address $00. If the second instruction is LDX $03, the table is searched again, but this time no entry for address $03 is found. Therefore this entry is added to the table and the pointers to the next and previous addresses are updated. After adding address $03, the table appears as in table 2.

Table 2

Memory Location	Address	(Next Address)	(Previous Address)	References
ZZZZ+1	0000	ZZZZ+1+2*0FF1	FFFF	3000
ZZZZ+1+1*0FF1	FFFF	FFFF	ZZZZ+1+2*0FF1	
ZZZZ+1+2*0FF1	0003	ZZZZ+1+1*0FF1	ZZZZ+1	3002

When x-reffing a large program, the table eventually becomes full. If the x-ref assembler detects this, it narrows its search range by neglecting (in the current pass) all addresses larger than the largest address found so far. The largest address of the search range is displayed on the screen under the heading EAR. Thus, as soon as the table is full, this address will change to a smaller value.

Suppose now that the table is full and the x-ref assembler finds an address, say QQQQ, that is in the search range but not in the table. In that case, an entry for this address has to be merged into the table. The program does this by first changing the value of the largest address in the search range (EAR) to the next largest address in the search range. Note that this address can be found by using the "previous address" pointer that is stored at each entry. The address QQQQ is then stored in the entry of the previous largest address which is empty now. Finally, the 'next address' pointer of the largest address smaller than QQQQ, the 'next address' and 'previous address' pointer of QQQQ, and the 'previous address' pointer of the smallest address larger than QQQQ, are updated to link QQQQ to the chain of addresses.

If the x-ref pass has been completed, the results are displayed or printed up to the largest address in the search range. In case all addresses could not be stored in the previous x-ref pass, the program puts the smallest address of the (new) search range (SAR) equal to the largest address of the (previous) search range plus one (i.e., SAR = EAR + 1) whereas EAR is put equal to FFFF. Next, another x-ref pass is made and this process continues until the references to all addresses have been displayed or printed.

Finally, I'll discuss some of the program parameters that can be changed by the user. These parameters can be found in the DATA SECTION of the listing.

The first four parameters are used to inform the program about your printer configuration. If you don't have a printer, put PRFLG equal to $00 and neglect the three parameters: PNTL, PNTH and CSND. If PRFLG equals zero, all output will be directed to the video screen. Since the output may run a little bit too fast on the screen to make notes, you can display one address (plus references) at a time by repeatedly pressing the escape key. Any other key will continue the output at normal speed.

I have distinguished three ways that printers can be connected to the Apple:

1. You may have an interface card, say in slot 2. In that case, put PNTL equal to the slot number (i.e. 2) and put PNTH as well as CSND equal to zero.

2. If you use a subroutine that drives the printer, put the low byte of this subroutine in PNTL and the high byte in PNTH, and put CSND equal to zero.

3. If the printer routine already has been connected before execution of the x-ref assembler, a special character to activate or deactivate the printer can be sent by storing the "printer off" character in PNTL, the "printer on" character in PNTH and by putting CSND equal to $FF.

Editor's note: For serial interface card in slot zero BRUN CROSS-SLOT-ZERO in place of CROSS ASSEMBLER to send output to the printer.

The next parameter is E0T1. This parameter contains the highest memory address used by the x-ref assembler (HIMEM). In the listing, E0T1 is put equal to $8FFF since my printer routine starts at $9000, but if you don't use a printer or disk, E0T1 can be put equal to the highest RAM address. The parameter 0FF1 equals the length of a table entry. Because 6 bytes per entry are needed to store the address and the pointers, 0FF1 equals $34 which means that (52-6)/2 = 23 references can be stored per entry. The last parameter, AMAX, is the maximal number of addresses that will be printed per line, if a printer has been connected.

I hope these parameters will offer you sufficient selection possibilities to make the type of cross reference table you need. If they do not, just x-ref the x-ref assembler, analyze it and make a version suited to your needs.

Figure 2

```
0800    ;************************
0800    ;*                      *
0800    ;*   CROSS  ASSEMBLER   *
0800    ;*          BY          *
0800    ;*   CORNELIS BONGERS   *
0800    ;*                      *
0800    ;*        CROSS         *
0800    ;*                      *
0800    ;*  COPYRIGHT (C) 1982  *
0800    ;*    MICRO INK, INC.   *
0800    ;* CHELMSFORD, MA 01824 *
0800    ;*  ALL RIGHTS RESERVED *
0800    ;*                      *
0800    ;************************
0800    ;
0800    ;
0800    ;
0800             ORG $800
009B    ESC      EPZ $9B            ;ESCAPE
008C    FF       EPZ $8C            ;FORM FEED
C000    KBD      EQU $C000
C010    CLKBD    EQU $C010
002D    RM       EPZ $2D            ;MONITOR LOCATIONS
002C    LM       EPZ $2C
0036    CSWL     EPZ $36
003C    A1L      EPZ $3C
003E    A2L      EPZ $3E
0042    A4L      EPZ $42
003A    PC       EPZ $3A
0024    CH       EPZ $24
002F    LEN      EPZ $2F
002E    FORM     EPZ $2E
F9C0    MNML     EQU $F9C0
FA00    MNMR     EQU $FA00
0800    ;
0800    ;
F88C    FMIN     EQU $F88C          ;MONITOR SUBROUTINES
FC58    CLSC     EQU $FC58
FD96    PRYX2    EQU $FD96
FDDA    PRBYT    EQU $FDDA
F94A    PRBL2    EQU $F94A
F948    PRBAK    EQU $F948
F9B4    CHAR1    EQU $F9B4
F9BA    CHAR2    EQU $F9BA
FDED    COUT     EQU $FDED
FD8E    CROUT    EQU $FD8E
FE95    OUTP     EQU $FE95
F953    PCADJ    EQU $F953
0800    ;
0800    ;
007E    AST      EPZ $7E            ;DATA REGISTERS
0080    PC1      EPZ $80
0082    PC2      EPZ $82
0084    PCU1     EPZ $84
0086    PCU2     EPZ $86
0000    PASS     EPZ $00
0001    SAR      EPZ $01            ;START ADDRESS RANGE
0003    EAR      EPZ $03            ;END ADDRESS RANGE
0005    TSP      EPZ $05            ;TABLE POINTER
0007    PCU      EPZ $07            ;USERS PROGRAM POINTER
0088    SOT      EPZ $88            ;START OF TABLE
008A    EOT      EPZ $8A            ;END OF TABLE
008C    RUNT     EPZ $8C            ;GENERAL USE
008E    END      EPZ $8E            ;POINTER TO LARGEST ADDRESS
0090    HULP     EPZ $90            ;TEMPORARY STORAGE
0092    SAVX     EPZ $92            ;STORAGE X REGISTER
0093    CNTR     EPZ $93            ;BYTE COUNTER
0094    OFF      EPZ $94            ;LENGTH OF TABLE ENTRY
0095    EOP      EPZ $95            ;END OF PROGRAM FLAG
0096    TFUL     EPZ $96            ;TABLE FULL FLAG
0097    ACNT     EPZ $97            ;COUNTER
0098    ETBL     EPZ $98            ;END OF TABLE FLAG
0800    ;
```

```
0800               ;
0800               ;*** START OF PROGRAM ***
0800               ;
0800               ;   * INITIALIZATION *
0800               ;
0800 A9 4C                 LDA #$4C
0802 8D F8 03              STA $3F8
0805 A9 2E                 LDA #BEGIN           ;BRUN THIS PART
0807 8D F9 03              STA $3F9             ;INIT CONTROL Y VECTOR
080A A9 08                 LDA /BEGIN
080C 8D FA 03              STA $3FA
080F 60                    RTS
0810               ;
0810               ;   * DATA SECTION *
0810               ;
0810 FF            PRFLG   BYT $FF              ;SET PRINTER FLAG
0811 8E            PNTL    BYT $8E              ;^N TO DEACTIVATE PRINTER

0812 90            PNTH    BYT $90              ;^P TO ACTIVATE PRINTER
0813 FF            CSND    BYT $FF              ;SET CHAR SEND BYTE
0814 FF 8F         EOT1    ADR $8FFF            ;END OF MEMORY USED (HIMEM)
0816 34            OFF1    BYT $34              ;LENGTH OF A TABLE ENTRY
0817 0A            AMAX    BYT $0A              ;MAXIMUM NO. OF
                                                 ADDRESSES/LINE
0818               ;
0818 50 41 53      HEAD    ASC 'PASSSAR EAR TSP PCU'
081B 53 53 41
081E 52 20 45
0821 41 52 20
0824 54 53 50
0827 20 50 43
082A 55
082B 8D                    BYT $8D
082C 00 00         SAR1    ADR $0000            ;START ADDRESS RANGE
082E               ;
082E               ;   * INIT PROGRAM PARAMETERS *
082E               ;
082E A2 FE         BEGIN   LDX #$FE
0830 38                    SEC
0831 B5 40         IFR     LDA A2L+$02,X
0833 69 00                 ADC #$00             ;INIT START OF TABLE
0835 95 8A                 STA SOT+$02,X        ;TO SOT=A2L+1
0837 B5 3E                 LDA A1L+$02,X
0839 95 82                 STA PC1+$02,X        ;INIT PROGRAM COUNTERS
083B 95 84                 STA PC2+$02,X
083D B5 44                 LDA A4L+$02,X
083F 95 86                 STA PCU1+$02,X       ;INIT USERS PROGRAM COUNTER
0841 95 88                 STA PCU2+$02,X
0843 BD 2E 07              LDA SAR1-$FE,X       ;INIT START ADDRESS RANGE
0846 95 03                 STA SAR+$02,X
0848 E8                    INX
0849 D0 E6                 BNE IFR
084B 86 95                 STX EOP              ;INIT END OF PROGRAM FLAG
084D 86 98                 STX ETBL             ;AND END OF TABLE FLAG
084F AD 16 08              LDA OFF1             ;INIT LENGTH TABLE ENTRY
0852 85 94                 STA OFF
0854 AD 14 08              LDA EOT1
0857 E5 94                 SBC OFF              ;END INIT END OF TABLE
0859 85 8A                 STA EOT              ;TO END OF MEMORY-1-OFF
085B AD 15 08              LDA EOT1+1
085E E9 00                 SBC #$00
0860 85 8B                 STA EOT+1
0862 A9 80                 LDA #$80
0864 85 00                 STA PASS             ;INIT PASS
0866 20 58 FC              JSR CLSC             ;CLEAR SCREEN
0869               ;
0869               ;   * MAIN PROGRAM *
0869               ;
0869 A2 00         AIT     LDX #$00             ;CLEAR TABLE FULL POINTER
086B 86 96                 STX TFUL
086D E8                    INX
086E 20 09 0C              JSR SPCN             ;INIT PROGRAM COUNTERS
0871 A9 FF                 LDA #$FF             ;INIT END ADDRESS RANGE
0873 85 03                 STA EAR
0875 85 04                 STA EAR+1
0877 20 8E 08              JSR POUT             ;DEACTIVATE PRINTER
```

```
087A 20 A1 08                JSR XASM              ;CROSS ASSEMBLE
087D A2 01                   LDX #$01
087F 20 29 0C                JSR PONOF             ;ACTIVATE PRINTER
0882 20 26 0B                JSR PRINT             ;PRINT RESULTS
0885 A5 96                   LDA TFUL              ;WAS TABLE FULL?
0887 D0 0A                   BNE NRDY              ;YES, BRANCH
0889 A9 8C                   LDA #FF               ;READY, GIVE FORM FEED
088B 20 ED FD                JSR COUT
088E A2 00          POUT     LDX #$00              ;DEACTIVATE PRINTER
0890 4C 29 0C                JMP PONOF             ;RTS TO MONITOR
0893 38             NRDY     SEC
0894 A2 FE                   LDX #$FE
0896 B5 05          NLA      LDA EAR+$02,X
0898 69 00                   ADC #$00              ;PUT SAR=EAR+1
089A 95 03                   STA SAR+$02,X
089C E8                      INX
089D D0 F7                   BNE NLA
089F F0 C8                   BEQ AIT               ;ALWAYS
08A1                ;
08A1                ;  * CROSS ASSEMBLE SUBROUTINE *
08A1                ;
08A1 20 5D 09       XASM     JSR IRUNT             ;INIT RUNT
08A4 A9 FF                   LDA #$FF
08A6 A2 00                   LDX #$00
08A8 86 8C                   STX RUNT
08AA A6 8B                   LDX EOT+1
08AC C8             MFUR     INY
08AD D0 02                   BNE STOR
08AF E6 8D                   INC RUNT+1
08B1 91 8C          STOR     STA (RUNT),Y          ;FILL TABLE WITH FF'S
08B3 E4 8D                   CPX RUNT+1
08B5 D0 F5                   BNE MFUR
08B7 C4 8A                   CPY EOT
08B9 D0 F1                   BNE MFUR
08BB A0 00                   LDY #$00
08BD 98                      TYA
08BE 91 88                   STA (SOT),Y           ;INIT FIRST TABLE ENTRY
08C0 C8                      INY
08C1 91 88                   STA (SOT),Y
08C3 18                      CLC
08C4 A2 FE                   LDX #$FE
08C6 A5 94                   LDA OFF
08C8 75 8A          NLP      ADC SOT+$02,X
08CA 95 07                   STA TSP+$02,X         ;TSP=SOT+OFF
08CC 95 90                   STA END+$02,X         ;END=TSP
08CE C8                      INY
08CF 91 88                   STA (SOT),Y           ;NEXT ENTRY IS TSP
08D1 A9 00                   LDA #$00
08D3 E8                      INX
08D4 D0 F2                   BNE NLP
08D6 C8                      INY
08D7 A5 88                   LDA SOT               ;INIT SECOND TABLE ENTRY
08D9 91 05                   STA (TSP),Y           ;PREVIOUS ENTRY IS SOT
08DB A5 89                   LDA SOT+1
08DD C8                      INY
08DE 91 05                   STA (TSP),Y
08E0 20 67 09                JSR ATSP1             ;SET TSP TO FIRST EMPTY ENTRY
08E3 20 8E FD                JSR CROUT             ;CARRIAGE RETURN
08E6 A0 00                   LDY #$00              ;PRINT HEADING
08E8 A2 04          CP1      LDX #$04
08EA B9 18 08       CPR      LDA HEAD,Y
08ED 20 ED FD                JSR COUT
08F0 C8                      INY
08F1 CA                      DEX
08F2 D0 F6                   BNE CPR
08F4 20 48 F9                JSR PRBAK
08F7 C0 14                   CPY #$14
08F9 D0 ED                   BNE CP1
08FB 20 6F 0A                JSR APASS             ;ADJUST PASS
08FE A5 00          DNI      LDA PASS
0900 29 7F                   AND #$7F
0902 A0 01                   LDY #$01
0904 84 24                   STY CH                ;INIT CURSOR
0906 88                      DEY
0907 F0 09                   BEQ PRT               ;ALWAYS
```

```
0909 B9 00 00   ANP     LDA PASS,Y      ;DISPLAY PASS,SAR,EAR,
090C 20 DA FD           JSR PRBYT       ;TSP AND PCU
090F B9 FF FF           LDA PASS-1,Y
0912 20 DA FD   PRT     JSR PRBYT
0915 20 48 F9           JSR PRBAK
0918 C8                 INY
0919 C8                 INY
091A C0 0A              CPY #$A
091C D0 EB              BNE ANP
091E A9 FF              LDA #$FF
0920 85 93              STA CNTR        ;INIT COUNTER
0922 20 8F 09           JSR DISA        ;DISASSEMBLE ONE INSTRUCTION
0925 A6 93              LDX CNTR
0927 30 2E              BMI ENF         ;BRANCH IF NO ADDRESS
0929 D0 06              BNE NZP         ;BRANCH IF NO ZERO PAGE ADDRESS

092B A5 7E              LDA AST         ;ADJUST AST FOR ZP INSTRUCTIONS

092D 85 7F              STA AST+1
092F 86 7E              STX AST
0931 20 78 0A   NZP     JSR RANGE       ;CHECK WHETHER ADDRESS IS IN
0934 90 21              BCC ENF         ;TABLE RANGE AND BRANCH IF NOT

0936 20 8C 0A           JSR SEAR        ;SEARCH ADDRESS IN TABLE
0939 B0 03              BCS FOUN        ;BRANCH IF EXACT MATCH
093B 20 B2 0A           JSR MERGE       ;MERGE IF NOT
093E A0 05      FOUN    LDY #$05
0940 C8         NEF     INY             ;SEARCH ROOM IN TABLE ENTRY
0941 C4 94              CPY OFF         ;TO STORE USERS PROGRAM COUNTER

0943 F0 12              BEQ ENF         ;BRANCH IF ENTRY FULL
0945 B1 8C              LDA (RUNT),Y
0947 C8                 INY
0948 31 8C              AND (RUNT),Y
094A 49 FF              EOR #$FF
094C D0 F2              BNE NEF
094E A5 07              LDA PCU         ;ROOM FOUND, STORE
0950 91 8C              STA (RUNT),Y    ;USERS PROGRAM COUNTER
0952 88                 DEY
0953 A5 08              LDA PCU+1
0955 91 8C              STA (RUNT),Y
0957 20 72 09   ENF     JSR ENFL        ;ADJUST PROGRAM COUNTERS
095A 90 A2              BCC DNI
095C 60                 RTS             ;RTS IF END OF PROGRAM REACHED

095D            ;
095D            ;  * IRUNT/ATSP2 : PUT RUNT=SOT OR TSP=TSP+OFF
095D            ;
095D A4 89      IRUNT   LDY SOT+1
095F 84 8D              STY RUNT+1
0961 A4 88              LDY SOT
0963 84 8C              STY RUNT
0965 60                 RTS
0966 18         ATSP2   CLC
0967 A5 05      ATSP1   LDA TSP
0969 65 94              ADC OFF
096B 85 05              STA TSP
096D 90 02              BCC RETR
096F E6 06              INC TSP+1
0971 60         RETR    RTS
0972            ;
0972            ;  * ENFL:SUBROUTINE ADJUST PROGRAM COUNTERS
0972            ;  * AND CHECK ON END OF PROGRAM
0972            ;
0972 20 60 0A   ENFL    JSR PCAD        ;ADJUST USERS PROGRAM COUNTER
0975 85 07              STA PCU
0977 84 08              STY PCU+1
0979 20 53 F9           JSR PCADJ       ;ADJUST PROGRAM COUNTER
097C 85 3A              STA PC
097E 84 3B              STY PC+1
```

```
0980 38                   SEC
0981 E5 88                SBC SOT        ;END OF PROGRAM REACHED?
0983 98                   TYA
0984 E5 89                SBC SOT+1
0986 24 00                BIT PASS
0988 10 04                BPL KLR        ;YES, RETURN WITH CARRY SET
098A 90 02                BCC KLR        ;IF X-ASSEMBLE PASS
098C E6 95                INC EOP        ;IF PRINT PASS, SET EOP FLAG
098E 60         KLR       RTS
098F            ;
098F            ;  * DISA:DISASSEMBLE ONE INSTRUCTION (X-ASSEMBLE PASS
098F            ;  * IF PASS POSITIVE, PRINT PASS IF PASS NEGATIVE)
098F            ;  * SEE ALSO APPLE MONITOR FOR COMMENTS
098F            ;
098F A2 00      DISA      LDX #$00
0991 20 8C F8             JSR FMIN       ;DETERMINE FORMAT INSTRUCTION
0994 24 00                BIT PASS       ;IF PASS IS POS,
0996 10 4A                BPL XASS       ;CROSS ASSEMBLE
0998 48                   PHA
0999 A6 07                LDX PCU
099B A4 08                LDY PCU+1
099D 20 96 FD             JSR PRYX2      ;PRINT USERS PROGRAM COUNTER
09A0 A0 00                LDY #$00
09A2 B1 3A      PROP      LDA (PC),Y
09A4 20 DA FD             JSR PRBYT      ;PRINT OPCODES
09A7 4C AD 09             JMP NOBL       ;IN A 8 CHAR. FIELD
09AA 20 4A F9   PRBK      JSR PRBL2
09AD C4 2F      NOBL      CPY LEN
09AF C8                   INY
09B0 90 F0                BCC PROP
09B2 A2 02                LDX #$02
09B4 C0 03                CPY #$03
09B6 90 F2                BCC PRBK
09B8 CA                   DEX
09B9 20 4A F9             JSR PRBL2      ;PRINT BLANK
09BC A2 03                LDX #$03
09BE 68                   PLA
09BF A8                   TAY
09C0 B9 C0 F9             LDA MNML,Y     ;FETCH MNEMONIC
09C3 85 2C                STA LM
09C5 B9 00 FA             LDA MNMR,Y
09C8 85 2D                STA RM
09CA A9 00      M1        LDA #$00
09CC A0 05                LDY #$05
09CE 06 2D      M2        ASL RM
09D0 26 2C                ROL LM
09D2 2A                   ROL
09D3 88                   DEY
09D4 D0 F8                BNE M2
09D6 69 BF                ADC #$BF
09D8 20 ED FD             JSR COUT
09DB CA                   DEX
09DC D0 EC                BNE. M1
09DE E8                   INX
09DF 20 4A F9             JSR PRBL2      ;PRINT BLANK AFTER MNEMONIC
09E2 A4 2F      XASS      LDY LEN
09E4 A2 06                LDX #$06
09E6 E0 03      PRAD1     CPX #$03
09E8 F0 27                BEQ PRAD5      ;FETCH ADDRESS IF PRESENT
09EA 06 2E      PRAD2     ASL FORM
09EC 90 18                BCC PRAD3
09EE BD B3 F9             LDA CHAR1-1,X
09F1 24 00                BIT PASS
09F3 30 06                BMI OUT
09F5 C9 A3                CMP #$A3       ;IS CHAR "#" ?
09F7 F0 66                BEQ RTS3       ;YES, STOP DISASSEMBLING
09F9 D0 0B                BNE PRAD3      ;NO, CONTINUE
09FB 20 ED FD   OUT       JSR COUT
09FE BD B9 F9             LDA CHAR2-1,X
0A01 F0 03                BEQ PRAD3
0A03 20 ED FD             JSR COUT
0A06 CA         PRAD3     DEX
0A07 D0 DD                BNE PRAD1
0A09 F0 1E                BEQ EMNO       ;ALWAYS
0A0B 88         PRAD4     DEY
```

```
0A0C 30 DC              BMI PRAD2
0A0E 20 4E 0A           JSR MPBY         ;PRINT ADDRESS BYTE
0A11 A5 2E     PRAD5    LDA FORM
0A13 C9 E8              CMP #$E8
0A15 B1 3A              LDA (PC),Y
0A17 90 F2              BCC PRAD4
0A19 20 63 0A           JSR PCA3
0A1C AA                 TAX
0A1D E8                 INX
0A1E D0 01              BNE PRYX
0A20 C8                 INY
0A21 98        PRYX     TYA
0A22 20 4E 0A           JSR MPBY
0A25 8A                 TXA
0A26 20 4E 0A           JSR MPBY
0A29 24 00     EMNO     BIT PASS
0A2B 10 32              BPL RTS3         ;IF NO PRINT PASS, RTS
0A2D A9 18              LDA #$18
0A2F 85 24              STA CH           ;TAB FOR PRINTER OR VIDEO
0A31 A0 00              LDY #$00
0A33 B1 3A     OOP      LDA (PC),Y
0A35 09 80              ORA #$80
0A37 C9 FF              CMP #$FF         ;MY PRINTER DOESN'T ACCEPT FF
0A39 F0 04              BEQ SPTA
0A3B C9 A0              CMP #$A0         ;OUTPUT ASCII VALUES OPCODES
0A3D B0 02              BCS CHT
0A3F A9 A0     SPTA     LDA #$A0
0A41 20 ED FD  CHT      JSR COUT
0A44 C4 2F              CPY LEN
0A46 C8                 INY
0A47 90 EA              BCC OOP
0A49 A9 1E              LDA #$1E         ;SET TAB FOR PRINTER OR VIDEO
0A4B 4C D1 0B           JMP ITAB1
0A4E           ;
0A4E           ;  * MPBY:PRINT BYTE IF PRINT PASS
0A4E           ;  * STEAL (ADDRESS) BYTE IF X-PASS
0A4E           ;
0A4E 24 00     MPBY     BIT PASS
0A50 10 03              BPL NPR
0A52 4C DA FD           JMP PRBYT        ;PRINT BYTE AND RTS
0A55 E6 93     NPR      INC CNTR         ;COUNT ADDRESS BYTES
0A57 86 92              STX SAVX         ;SAVE X-REGISTER
0A59 A6 93              LDX CNTR
0A5B 95 7E              STA AST,X        ;SAVE ADDRESS BYTE
0A5D A6 92              LDX SAVX
0A5F 60        RTS3     RTS
0A60           ;
0A60           ;  * PCAD:ADJUST USERS PROGRAM COUNTER
0A60           ;  * OR CALCULATE TARGET FOR RELATIVE BRANCH
0A60           ;
0A60 38        PCAD     SEC
0A61 A5 2F              LDA LEN
0A63 A4 08     PCA3     LDY PCU+1
0A65 AA                 TAX
0A66 10 01              BPL PCA4
0A68 88                 DEY
0A69 65 07     PCA4     ADC PCU
0A6B 90 01              BCC RTS4
0A6D C8                 INY
0A6E 60        RTS4     RTS
0A6F           ;
0A6F           ;  * APASS:ADJUST PASS
0A6F           ;
0A6F A4 00     APASS    LDY PASS
0A71 C8                 INY
0A72 98                 TYA
0A73 49 80              EOR #$80
0A75 85 00              STA PASS
0A77 60                 RTS
0A78           ;
0A78           ;  * RANGE:CHECK WHETHER ADDRESS IS IN SEARCH RANGE
0A78           ;  * RETURN WITH CARRY CLEAR IF NOT, ELSE WITH CARRY SET
0A78           ;
0A78 38        RANGE    SEC
```

```
0A79 A5 03              LDA EAR
0A7B E5 7F              SBC AST+1
0A7D A5 04              LDA EAR+1
0A7F E5 7E              SBC AST
0A81 90 08              BCC NIR
0A83 A5 7F              LDA AST+1
0A85 E5 01              SBC SAR
0A87 A5 7E              LDA AST
0A89 E5 02              SBC SAR+1
0A8B 60         NIR     RTS
0A8C            ;
0A8C            ;   * SEAR:SEARCH ADDRESS IN TABLE, RETURN WITH CARRY SET
0A8C            ;   * IF FOUND, ELSE WITH CARRY CLEAR
0A8C            ;
0A8C 20 5D 09   SEAR    JSR IRUNT          ;PUT RUNT=SOT
0A8F 38                 SEC
0A90 A0 01      CON     LDY #$01
0A92 A5 7F              LDA AST+1
0A94 F1 8C              SBC (RUNT),Y
0A96 AA                 TAX
0A97 88                 DEY
0A98 A5 7E              LDA AST
0A9A F1 8C              SBC (RUNT),Y
0A9C 90 05              BCC RTS5           ;RTS IF TABLE ENTRY
                                            ADDRESS IS
0A9E D0 04              BNE NFND           ;LARGER THAN ADDRESS
                                            SEARCHED FOR
0AA0 8A                 TXA
0AA1 D0 01              BNE NFND
0AA3 60         RTS5    RTS
0AA4 A0 02      NFND    LDY #$02
0AA6 B1 8C              LDA (RUNT),Y       ;GET NEXT TABLE ENTRY IN RUNT
0AA8 AA                 TAX
0AA9 C8                 INY
0AAA B1 8C              LDA (RUNT),Y
0AAC 85 8D              STA RUNT+1
0AAE 86 8C              STX RUNT
0AB0 B0 DE              BCS CON            ;ALWAYS
0AB2            ;
0AB2            ;   * MERGE:MERGE OR ADD ADDRESS (IN) TO TABLE IF
0AB2            ;   * NO EXACT MATCH FOUND
0AB2            ;
0AB2 A5 96      MERGE   LDA TFUL           ;IS TABLE FULL?
0AB4 F0 1C              BEQ NFUL           ;BRANCH IF NOT
0AB6 A5 8E              LDA END            ;ELSE RESERVE ENTRY LARGEST
0AB8 85 05              STA TSP            ;ADDRESS FOR CURRENT
0ABA A5 8F              LDA END+1           ADDRESS
0ABC 85 06              STA TSP+1
0ABE A0 04              LDY #$04           ;STORE NEXT BUT LARGEST
                                            ADDRESS
0AC0 20 1C 0C           JSR CFF1           ;ENTRY IN END FOR LATER USE
0AC3 85 8F              STA END+1
0AC5 86 8E              STX END
0AC7 A4 94              LDY OFF
0AC9 A9 FF              LDA #$FF
0ACB 88         PR      DEY                ;STORE FF'S IN TABLE ENTRY
0ACC 91 05              STA (TSP),Y
0ACE C0 06              CPY #$06
0AD0 D0 F9              BNE PR
0AD2 A0 05      NFUL    LDY #$05
0AD4 A2 01              LDX #$01
0AD6 B1 8C      RET     LDA (RUNT),Y       ;ADJUST POINTERS
0AD8 91 05              STA (TSP),Y        ;(TSP),4,5=(RUNT),4,5
0ADA 95 90              STA HULP,X         ;HULP,0,1=(RUNT),4,5
0ADC B5 05              LDA TSP,X
0ADE 91 8C              STA (RUNT),Y       ;(RUNT),4,5=TSP,0,1
0AE0 88                 DEY
0AE1 CA                 DEX
0AE2 10 F2              BPL RET
0AE4 A2 01              LDX #$01
0AE6 B1 90      REK     LDA (HULP),Y
0AE8 91 05              STA (TSP),Y
0AEA B5 05              LDA TSP,X
0AEC 91 90              STA (HULP),Y       ;(HULP),2,3=TSP,0,1
0AEE 95 8C              STA RUNT,X         ;RUNT,0,1=TSP,0,1
```

```
0AF0 88                   DEY
0AF1 CA                   DEX
0AF2 10 F2                BPL REK
0AF4 A5 7F                LDA AST+1          ;(TSP),0,1=ADDRESS
0AF6 91 05                STA (TSP),Y
0AF8 A5 7E                LDA AST
0AFA 88                   DEY
0AFB 91 05                STA (TSP),Y
0AFD A5 96                LDA TFUL           ;IS TABLE FULL?
0AFF D0 0E                BNE YFULL          ;YES, BRANCH
0B01 20 66 09             JSR ATSP2          ;NO, ADJUST TSP
0B04 38                   SEC
0B05 E5 8A                SBC EOT            ;CHECK WHETHER TABLE
                                              IS FULL NOW
0B07 A5 06                LDA TSP+1
0B09 E5 8B                SBC EOT+1
0B0B 90 18                BCC RTSS           ;NO, BRANCH
0B0D E6 96                INC TFUL           ;YES, SET TABLE FULL POINTER
0B0F A0 04       YFULL    LDY #$04
0B11 B1 8E                LDA (END),Y        ;SET END ADDRESS RANGE
0B13 85 90                STA HULP
0B15 C8                   INY
0B16 B1 8E                LDA (END),Y
0B18 85 91                STA HULP+1
0B1A A0 01                LDY #$01
0B1C B1 90                LDA (HULP),Y
0B1E 85 03                STA EAR
0B20 88                   DEY
0B21 B1 90                LDA (HULP),Y
0B23 85 04                STA EAR+1
0B25 60          RTSS     RTS
0B26             ;
0B26             ;  * PRINT: PRINT PASS
0B26             ;
0B26 20 6F 0A    PRINT    JSR APASS
0B29 A2 03                LDX #$03
0B2B 20 09 0C             JSR SPCN           ;RESTORE PRINT COUNTERS
0B2E B5 88       NEL      LDA SOT,X
0B30 95 05                STA TSP,X          ;TSP=SOT
0B32 CA                   DEX
0B33 10 F9                BPL NEL
0B35 20 1A 0C             JSR CFFF           ;(TSP),6,7=$FFFF ?
0B38 D0 03                BNE PZER           ;BRANCH IF NOT
0B3A 20 DF 0B             JSR CHKR           ;YES, NEGLECT FIRST ENTRY
0B3D AD 00 C0    PZER     LDA KBD            ;KEYBOARD INPUT = ESCAPE
                                              CHAR.?
0B40 C9 9B                CMP #ESC
0B42 D0 08                BNE PZERT          ;NO, BRANCH
0B44 8D 10 C0             STA CLKBD          ;CLEAR STROBE
0B47 AD 00 C0    ASK      LDA KBD            ;WAIT FOR ANOTHER STROBE
0B4A 10 FB                BPL ASK
0B4C A5 95       PZERT    LDA EOP            ;END OF PROGRAM FLAG ON?
0B4E D0 31                BNE PAI            ;YES, DO NOT DISASSEMBLE
0B50 A5 98                LDA ETBL           ;IF END OF TABLE THEN
0B52 D0 20                BNE PAD            ;DISASSEMBLE ONLY
0B54 A0 01                LDY #$01
0B56 38                   SEC
0B57 B1 05                LDA (TSP),Y
0B59 E5 07                SBC PCU
0B5B AA                   TAX
0B5C 88                   DEY
0B5D B1 05                LDA (TSP),Y
0B5F E5 08                SBC PCU+1          ;ADDRESS EQUAL TO PCU?
0B61 90 1E                BCC PAI            ;NO, ADDRESS IS SMALLER
0B63 D0 0F                BNE PAD            ;NO, ADDRESS IS LARGER
0B65 8A                   TXA
0B66 D0 0C                BNE PAD
0B68 20 8F 09             JSR DISA           ;ADDRESS=PCU HERE, PRINT
0B6B 20 95 0B             JSR INFO           ;ADDRESS, DISASS. AND
                                              TABLE INFO
0B6E 20 72 09             JSR ENFL           ;ADJUST PROGRAM COUNTERS
0B71 4C 90 0B             JMP CHEK
0B74 20 8F 09    PAD      JSR DISA           ;PRINT ADDRESS AND
                                              DISASSEMBLE
0B77 20 72 09             JSR ENFL
0B7A A5 95                LDA EOP
```

```
0B7C 25 98            AND ETBL          ;RTS EOP=1 AND ETBL=1
0B7E F0 BD            BEQ PZER
0B80 60               RTS
0B81 A0 01     PAI    LDY #$01          ;PRINT ADDRESS AND INFO
0B83 B1 05            LDA (TSP),Y
0B85 AA               TAX
0B86 88               DEY
0B87 B1 05            LDA (TSP),Y
0B89 A8               TAY
0B8A 20 96 FD         JSR PRYX2
0B8D 20 95 0B         JSR INFO
0B90 20 DF 0B  CHEK   JSR CHKR
0B93 D0 A8            BNE PZER          ;ALWAYS
0B95           ;
0B95           ;  * INFO:PRINT TABLE ENTRY INFORMATION
0B95           ;
0B95 A0 06     INFO   LDY #$06
0B97 20 CF 0B         JSR ITAB          ;SET TAB FOR PRINTER
0B9A 20 1C 0C  NPRQ   JSR CFF1
0B9D F0 2F            BEQ RS            ;IF ADDRESS=$FFFF THEN READY
0B9F 48        PINT   PHA               ;SAVE ACCU
0BA0 8A               TXA
0BA1 48               PHA
0BA2 A5 97            LDA ACNT
0BA4 2D 10 08         AND PRFLG         ;MAXIMUM NO. OF ADDRESSES
0BA7 CD 17 08         CMP AMAX          ;PER LINE PRINTED?
0BAA 90 06            BCC NCRT          ;BRANCH IF NOT
0BAC 20 8E FD         JSR CROUT         ;CARRIAGE RETURN IF YES
0BAF 20 CF 0B         JSR ITAB          ;SET TAB AND INIT ACNT
0BB2 98        NCRT   TYA
0BB3 E6 97            INC ACNT
0BB5 0D 10 08         ORA PRFLG
0BB8 C9 07            CMP #$07
0BBA F0 05            BEQ NBLK
0BBC A2 01            LDX #$01          ;PRINT BLANK
0BBE 20 4A F9         JSR PRBL2
0BC1 68        NBLK   PLA               ;FOLLOWED BY THE ADDRESS
0BC2 20 DA FD         JSR PRBYT
0BC5 68               PLA
0BC6 20 DA FD         JSR PRBYT
0BC9 C8               INY
0BCA C4 94            CPY OFF
0BCC D0 CC            BNE NPRQ
0BCE 60        RS     RTS               ;RTS IF END OF ENTRY REACHED
0BCF           ;
0BCF           ;  * ITAB:SET TAB FOR PRINTER OR VIDEO AND
0BCF           ;  * INIT ADDRESS COUNTER
0BCF           ;
0BCF A5 24     ITAB   LDA CH
0BD1 2C 10 08  ITAB1  BIT PRFLG         ;PRINTER ON?
0BD4 10 02            BPL STCH          ;BRANCH IF NOT
0BD6 A9 1C            LDA #$1C          ;SET TAB FOR PRINTER
0BD8 85 24     STCH   STA CH
0BDA A9 00            LDA #$00          ;INIT ADDRESS COUNTER
0BDC 85 97            STA ACNT
0BDE 60               RTS
0BDF           ;
0BDF           ;  * CHKR:CHECK IF END OF TABLE REACHED AND RTS IF SO
0BDF           ;  * IF NOT, ADJUST TSP
0BDF           ;
0BDF A0 02     CHKR   LDY #$02
0BE1 20 1C 0C         JSR CFF1
0BE4 C5 8F            CMP END+1
0BE6 D0 1C            BNE ATSP
0BE8 E4 8E            CPX END
0BEA D0 18            BNE ATSP          ;BRANCH ON END OF TABLE
0BEC A5 96            LDA TFUL          ;CHECK WHETHER TABLE FULL
0BEE 05 95            ORA EOP           ;OR END OF PROGRAM REACHED
0BF0 F0 10            BEQ ITBL          ;BRANCH IF NOT
0BF2 68               PLA
0BF3 68               PLA
0BF4 A2 01            LDX #$01
0BF6 B5 3A     APCR   LDA PC,X
0BF8 95 82            STA PC2,X
0BFA B5 07            LDA PCU,X
```

```
0BFC 95 86                   STA PCU2,X        ;ELSE
0BFE CA                      DEX               ;SAVE PRINT COUNTERS
0BFF 10 F5                   BPL APCR          ;AND RTS TO MAIN PROGRAM
0C01 60                      RTS
0C02 E6 98          ITBL     INC ETBL          ;SET END OF TABLE FLAG
0C04 86 05          ATSP     STX TSP
0C06 85 06                   STA TSP+1
0C08 60                      RTS
0C09                ;
0C09                ;  * SPCN:INIT PROGRAM COUNTERS
0C09                ;
0C09 A0 01          SPCN     LDY #$01
0C0B B5 80          SPC1     LDA PC1,X
0C0D 99 3A 00                STA PC,Y
0C10 B5 84                   LDA PCU1,X
0C12 99 07 00                STA PCU,Y
0C15 CA                      DEX
0C16 88                      DEY
0C17 10 F2                   BPL SPC1
0C19 60                      RTS
0C1A                ;
0C1A                ;  * CFFF:CHECK WHETHER (TSP),6,7=$FFFF
0C1A                ;
0C1A A0 06          CFFF     LDY #$06
0C1C B1 05          CFF1     LDA (TSP),Y
0C1E AA                      TAX
0C1F C8                      INY
0C20 B1 05                   LDA (TSP),Y
0C22 C9 FF                   CMP #$FF
0C24 D0 02                   BNE REET
0C26 E0 FF                   CPX #$FF
0C28 60             REET     RTS
0C29                ;
0C29                ;  * PONOF: PRINTER ON/OFF ROUTINE
0C29                ;  * X=1 ACTIVATES PRINTER, X=0 DEACTIVATES PRINTER
0C29                ;
0C29 AD 10 08       PONOF    LDA PRFLG         ;DO NOTHING IF PRINTER FLAG
0C2C F0 1D                   BEQ RTTS          ;IS OFF
0C2E AD 13 08                LDA CSND          ;SEND CHAR?
0C31 F0 09                   BEQ SLDV          ;BRANCH IF NOT
0C33 BD 11 08                LDA PNTL,X        ;LOAD CHAR AND
0C36 4C ED FD                JMP COUT          ;SEND TO PRINTER
0C39 4C 95 FE       VIDP     JMP OUTP          ;ACTIVATE SLOT
0C3C 8A             SLDV     TXA
0C3D F0 FA                   BEQ VIDP          ;BRANCH IF PRINTER OFF
                                                COMMAND
0C3F AD 11 08                LDA PNTL
0C42 AE 12 08                LDX PNTH
0C45 F0 F2                   BEQ VIDP          ;BRANCH IF INTERFACE CARD
0C47 85 36                   STA CSWL          ;ACTIVATE PRINTER ROUTINE
0C49 86 37                   STX CSWL+1
0C4B 60             RTTS     RTS
0C4C                ;
0C4C                ;  * END OF PROGRAM
0C4C                ;
0C4C                         END

***** END OF ASSEMBLY
```

```
]BRUN SORT,D2
BRUN SORT,D2
SYMBOL TABLE SORTED ALPHABETICALLY

 A1L     003C  A2L     003E  A4L     0042  ACNT    0097  AIT     0869
 AMAX    0817  ANP     0909  APASS   0A6F  APCR    0BF6  ASK     0B47
 AST     007E  ATSP    0C04  ATSP1   0967  ATSP2   0966  BEGIN   082E
 CFF1    0C1C  CFFF    0C1A  CH      0024  CHAR1   F9B4  CHAR2   F9BA
 CHEK    0B90  CHKR    0BDF  CHT     0A41  CLKBD   C010  CLSC    FC58
 CNTR    0093  CON     0A90  COUT    FDED  CP1     08E8  CPR     08EA
 CROUT   FD8E  CSND    0813  CSWL    0036  DISA    098F  DNI     08FE
 EAR     0003  EMNO    0A29  END     008E  ENF     0957  ENFL    0972
 EOP     0095  EOT     008A  EOT1    0814  ESC     009B  ETBL    0098
 FF      008C  FMIN    F88C  FORM    002E  FOUN    093E  HEAD    0818
 HULP    0090  IFR     0831  INFO    0B95  IRUNT   095D  ITAB    0BCF
 ITAB1   0BD1  ITBL    0C02  KBD     C000  KLR     098E  LEN     002F
 LM      002C  M1      09CA  M2      09CE  MERGE   0AB2  MFUR    08AC
 MNML    F9C0  MNMR    FA00  MPBY    0A4E  NBLK    0BC1  NCRT    0BB2
 NEF     0940  NEL     0B2E  NFND    0AA4  NFUL    0AD2  NIR     0A8B
 NLA     0896  NLP     08C8  NOBL    09AD  NPR     0A55  NPRQ    0B9A
 NRDY    0893  NZP     0931  OFF     0094  OFF1    0816  OOP     0A33
 OUT     09FB  OUTP    FE95  PAD     0B74  PAI     0B81  PASS    0000
 PC      003A  PC1     0080  PC2     0082  PCA3    0A63  PCA4    0A69
 PCAD    0A60  PCADJ   F953  PCU     0007  PCU1    0084  PCU2    0086
 PINT    0B9F  PNTH    0812  PNTL    0811  PONOF   0C29  POUT    088E
 PR      0ACB  PRAD1   09E6  PRAD2   09EA  PRAD3   0A06  PRAD4   0A0B
 PRAD5   0A11  PRBAK   F948  PRBK    09AA  PRBL2   F94A  PRBYT   FDDA
 PRFLG   0810  PRINT   0B26  PROP    09A2  PRT     0912  PRYX    0A21
 PRYX2   FD96  PZER    0B3D  PZERT   0B4C  RANGE   0A78  REET    0C28
 REK     0AE6  RET     0AD6  RETR    0971  RM      002D  RS      0BCE
 RTS3    0A5F  RTS4    0A6E  RTS5    0AA3  RTSS    0B25  RTTS    0C4B
 RUNT    008C  SAR     0001  SAR1    082C  SAVX    0092  SEAR    0A8C
 SLDV    0C3C  SOT     0088  SPC1    0C0B  SPCN    0C09  SPTA    0A3F
 STCH    0BD8  STOR    08B1  TFUL    0096  TSP     0005  VIDP    0C39
 XASM    08A1  XASS    09E2  YFULL   0B0F

SYMBOL TABLE SORTED BY ADDRESS

 PASS    0000  SAR     0001  EAR     0003  TSP     0005  PCU     0007
 CH      0024  LM      002C  RM      002D  FORM    002E  LEN     002F
 CSWL    0036  PC      003A  A1L     003C  A2L     003E  A4L     0042
 AST     007E  PC1     0080  PC2     0082  PCU1    0084  PCU2    0086
 SOT     0088  EOT     008A  RUNT    008C  FF      008C  END     008E
 HULP    0090  SAVX    0092  CNTR    0093  OFF     0094  EOP     0095
 TFUL    0096  ACNT    0097  ETBL    0098  ESC     009B  PRFLG   0810
 PNTL    0811  PNTH    0812  CSND    0813  EOT1    0814  OFF1    0816
 AMAX    0817  HEAD    0818  SAR1    082C  BEGIN   082E  IFR     0831
 AIT     0869  POUT    088E  NRDY    0893  NLA     0896  XASM    08A1
 MFUR    08AC  STOR    08B1  NLP     08C8  CP1     08E8  CPR     08EA
 DNI     08FE  ANP     0909  PRT     0912  NZP     0931  FOUN    093E
 NEF     0940  ENF     0957  IRUNT   095D  ATSP2   0966  ATSP1   0967
 RETR    0971  ENFL    0972  KLR     098E  DISA    098F  PROP    09A2
 PRBK    09AA  NOBL    09AD  M1      09CA  M2      09CE  XASS    09E2
 PRAD1   09E6  PRAD2   09EA  OUT     09FB  PRAD3   0A06  PRAD4   0A0B
 PRAD5   0A11  PRYX    0A21  EMNO    0A29  OOP     0A33  SPTA    0A3F
 CHT     0A41  MPBY    0A4E  NPR     0A55  RTS3    0A5F  PCAD    0A60
 PCA3    0A63  PCA4    0A69  RTS4    0A6E  APASS   0A6F  RANGE   0A78
 NIR     0A8B  SEAR    0A8C  CON     0A90  RTS5    0AA3  NFND    0AA4
 MERGE   0AB2  PR      0ACB  NFUL    0AD2  RET     0AD6  REK     0AE6
 YFULL   0B0F  RTSS    0B25  PRINT   0B26  NEL     0B2E  PZER    0B3D
 ASK     0B47  PZERT   0B4C  PAD     0B74  PAI     0B81  CHEK    0B90
 INFO    0B95  NPRQ    0B9A  PINT    0B9F  NCRT    0BB2  NBLK    0BC1
 RS      0BCE  ITAB    0BCF  ITAB1   0BD1  STCH    0BD8  CHKR    0BDF
 APCR    0BF6  ITBL    0C02  ATSP    0C04  SPCN    0C09  SPC1    0C0B
 CFFF    0C1A  CFF1    0C1C  REET    0C28  PONOF   0C29  VIDP    0C39
 SLDV    0C3C  RTTS    0C4B  KBD     C000  CLKBD   C010  FMIN    F88C
 PRBAK   F948  PRBL2   F94A  PCADJ   F953  CHAR1   F9B4  CHAR2   F9BA
 MNML    F9C0  MNMR    FA00  CLSC    FC58  CROUT   FD8E  PRYX2   FD96
 PRBYT   FDDA  COUT    FDED  OUTP    FE95
```

A Fast Fractional Math Package for 6502 Microcomputers

by Wes Huntress

Implemented for the Apple II computer, these routines can be used by any 6502 microcomputer to obtain fast fractional arithmetic from assembly language.

Have you ever faced the problem of wanting to use fractions or decimal arithmetic when only Integer BASIC was available? Have you ever wanted to use trigonometric or complex math functions with Integer BASIC? Or have you ever faced the opposite problem with Applesoft — complex math a snap, but if only it was as fast as Integer? If the former was your problem, then you probably upgraded your Apple II from Integer to Applesoft. If you did, or bought an Apple II Plus in the first place, then you may have come across the second problem at one time or another.

Applesoft provides floating point math with 9-digit accuracy including trigonometric functions. These features are superb for most applications requiring complex mathematics, but much slower compared to integer arithmetic. In some speed-critical applications, such as the projection and animation of high-resolution 3-D images, Applesoft is often simply too slow. In the case of 3-D animation, this results in a slow frame projection rate. The only way to improve the speed at present is to access Applesoft math subroutines directly from assembly language and thereby avoid the interpreter overhead.

If you are not a machine-language programmer, you can still increase execution speed of floating point math by using an Applesoft compiler. These compilers have just recently appeared on the software market and will convert your Applesoft program to a machine-language version which will run floating point math two to three times faster. The ultimate solution is to use an arithmetic processor board where the math routines are implemented in hardware. These boards are available, but at a price, and not every Apple owner will have one. Therefore, if you are writing software for a general audience, these peripheral boards are not the solution.

There is an alternate approach to getting a significant increase in math speed for Apple users with standard hardware configurations. The solution is to write an assembly-language math package that contains complex math functions but is built for speed rather than accuracy. One way would be to rewrite the Applesoft floating point math package to use 3-byte or 4-byte numbers instead of the standard Applesoft 5-byte format. The floating point routines in the Integer ROMs use a 4-byte format. Floating point math in any format is still significantly slower than integer math, so that if speed is the utmost consideration, then some form of integer math must be used. Use of integer math to gain speed requires only that we give up the ability in floating point math to represent very large or very small numbers. It is possible to represent fractional or decimal numbers with an integer format. The limitation to accuracy is the number of bytes used to represent a number.

The assembly-language routines presented in this article provide a very fast 3-byte (24-bit) integer math package which is capable of representing fractional numbers. Complex math functions, such as the trigonometric operators, have been implemented. The routines work in the same way as standard 2-byte multiple-precision integer arithmetic except that a third byte is included to represent the fractional part of a number. The first byte of the 3-byte number represents the fractional part, and the next two bytes are the integer part of the number in the familiar byte-swapped format. Examples of 3-byte fractional numbers are:

```
01 01 01 = 257 1/256    = 257.004
FF 40 00 =  64 255/256    64.996
24 03 00 =   3 36/256   =   3.141
```

The accuracy in the fractional part is one part in 256, or in decimal form 0.004. While this is not at all competitive with the accuracy of Applesoft, it may be all that is required in some applications and is the fastest possible fractional arithmetic in software. The accuracy could be improved by adding multiple-precision fractional parts, but this would soon lose efficiency compared to floating point routines. The numbers are signed so that the largest positive number which can be represented is 32767.996, and the largest negative number is −32768.996. The smallest numbers which can be represented are +/− 0.004.

The Fractional Integer AriThmetic (FIAT) package is intended to be tucked between DOS and its buffers. It is invisible to DOS and both BASICs. Figure 1 is a memory map showing where the FIAT code is located. FIAT contains its own variable space in page $98. At three bytes per variable, there is room for 85 variables and constants. This is sufficient for most applications.

To use FIAT from machine language it is easiest to think of it in terms of a pseudo-processor. FIAT has a set of 3-byte registers through which all operations are performed. The "op-codes" are subroutine calls (JSRs) to load these registers, operate on their contents, and move their contents to other registers or to memory. Figure 2 illustrates the programming model for FIAT.

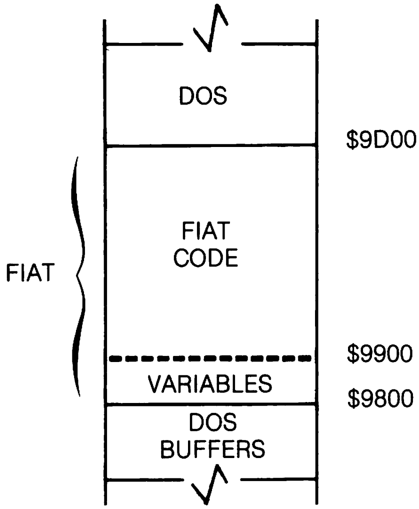


Figure 1: FIAT Memory Map

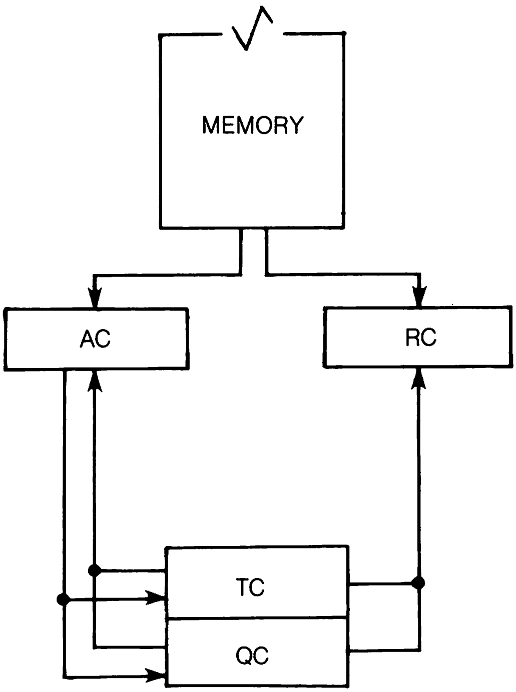


Figure 2: FIAT Programming Model

There are two main registers used in accessing memory and in arithmetic operations. The "AC" register, or accumulator, is the principal working register. All functions operate on the number in the AC register, and the results of all arithmetic operations are left in the AC register. Numbers can be transfered both to and from the AC register and memory. The usual program sequence for using an operator with single operand (a "unary" operator) such as SGN, ABS, INT, SIN, etc., is to first transfer a 3-byte number into the AC register, then call the arithmetic operator, and finally transfer the result from the AC register.

The RC register is used only for those operations requiring two operands ("binary" operators): ADD, SUB, MUL, DIV, and CMP. For these operations, one operand is placed in the AC register and the other in the RC register. Numbers can be moved into, but not out of, the RC register. The order of the operations is: AC SUB(tract) RC, AC DIV(ided by) RC, and AC CMP(ared to) RC. The result of ADD, SUB, MUL and DIV is left in the AC register. The CMP operator conditions the accumulator of the 6502.

The TC and QC registers are provided for storage of intermediate results. Results can be moved into TC and QC from AC, and back from TC or QC into AC or RC. Some care must be exercised in using the TC and QC registers however, since the SQR function uses the TC register, and the TAN and ATN functions use both the TC and QC registers.

There is a fifth register, the SC register, which is an extension of the AC register. The AC register, including the SC register and an extension byte ACX, can be as wide as seven bytes, depending on the operation being performed. This is all transparent to the user. Operands are always loaded in the 3-byte AC register (ACL, ACM, ACH) and all results are found in the AC register.

Listing 1 is a source listing of FIAT outlining the function and usage of each operator. Any special entry or exit conditions are listed below the routine title. Listing 2 gives the entry points for each function and lists the 6502 and FIAT registers used. Listing 3 is a sample listing of an assembly language program which uses FIAT.

Most routines in FIAT have more than one entry point. The principal entry point assumes that the AC register (and RC register if required) has already been loaded. For unary operators, a second entry point is provided which will load a specified variable from memory into AC and then perform the operation. These entry points are labeled with a "#" suffix. For example, the sine operator has a principal entry point labeled SIN and a unary entry point labeled SIN#. The unary entry point in this case requires that the 6502 Y register point to the variable in page $98.

Binary operators have three entry points. The principal entry point assumes that both the AC and RC registers are already loaded. The unary (suffix: "#") entry point loads the RC register with the variable indexed by the 6502 X register. The binary entry point, suffixed by "##", loads both the RC and AC registers. On binary entry, the 6502 X register points to the variable to be loaded into RC, and the 6502 Y register points to the variable to be loaded into AC.

The trig functions SIN, COS, and TAN in FIAT use degrees rather than radians. The unit angle is one degree. Positive angles, negative angles, and angles larger than 360 degrees can be input. Fractional angles will be converted to the nearest degree. The routine FXA can be used to reduce any angle to a positive value between 0 and 359.996 degrees.

The inverse trig functions ASN (arcsin), ACS (arccos), and ATN (arctan) are provided but are limited by the accuracy of the 1-byte fraction. To the accuracy available, the sines of 84, 85, and 86 degrees are all FF 00 00, and the sines of 87, 88, 89, and 90 degrees are all 00 01 00. Therefore ASN (FF 00 00) will return just 86 degrees and ASN (00 01 00) will return 90 degrees. This problem is only serious for angles near 90 degrees for the ASN operator, and for angles near 0 degrees for the ACS operator. The ATN function does not have this problem, but is accurate only to +/− one degree over its whole range and is slower than ASN or ACS. The ASN and ATN operators return an angle between −90 and +90 degrees. The ACS operator returns an angle between 0 and 180 degrees.

Range checking to maximize speed is not provided for any of the operators. The user is responsible for insuring that the input is in the proper format (3-byte signed integer) and that the operation does not result in overflow. The INC, DEC, ADD and SUB operators will "wrap-around" on overflow. For example: 1 ADD 32767 = −32768. A multiply which would result in a number with a value greater than 32767.996 or less than −32768.996 yields nonsense. A DIV by zero or INV zero yields nonsense. The SQR function returns the square root of an unsigned number for negative number input. The trig functions are more forgiving. SIN, COS, and TAN will accept any value and reduce it with FXA. ASN and ACS assume any values greater than +/− 1.0 to be equal to +/− 1.0. The ATN function accepts any value.

The improvement in speed gained by using FIAT instead of Applesoft floating point math is very large for all but the multiply and divide routines. For MUL and DIV the gain is a factor of about 5. For the ADD and SUB routines, the gain is a factor of 50. For INT, SGN, and ABS routines the gain is about 100. For the SQR, SIN, COS, and ASN (*vs.* derived Applesoft arcsin) the gain is about 200. For TAN and ATN the gain is about 40 and 20 respectively.

To make your own copy of FIAT, use your assembler to copy the source listing in listing 1 and assemble it at $800. Get into the monitor and type in the data given in listing 4 for the sine table from $80D to $866. Now, from the monitor, type:

```
*98F3<800.BFFM <CR>
*98F3G <CR>
*BSAVE FIAT,A$98F3,L$3FF <CR>
```

To install FIAT, boot the system and type RUN FIAT-LOADER (listing 4).

Listing 1

```
0800          ;***********************
0800          ;*                     *
0800          ;*  24-BIT FRACTIONAL  *
0800          ;*   SIGNED INTEGER    *
0800          ;*     ARITHMETIC      *
0800          ;*                     *
0800          ;*   BY WES HUNTRESS   *
0800          ;*                     *
0800          ;*  COPYRIGHT (C) 1982 *
0800          ;*    MICRO INK, INC.  *
0800          ;* CHELMSFORD, MA 01824*
0800          ;* ALL RIGHTS RESERVED *
0800          ;*                     *
0800          ;***********************
0800          ;
0800          ;EQUATES
0800          ;
0800          ZP      EPZ 0
0800          QCL     EPZ $03
0800          QCM     EPZ QCL+1
0800          TCL     EPZ $07
0800          ACX     EPZ $F9
0800          ACL     EPZ ACX+1
0800          ACM     EPZ ACL+1
0800          ACH     EPZ ACM+1
0800          SCL     EPZ ACH+1
0800          SCM     EPZ SCL+1
0800          SCH     EPZ SCM+1
0800          RCL     EPZ $EB
0800          RCM     EPZ RCL+1
0800          RCH     EPZ RCM+1
0800          TMP     EPZ RCH+1
0800          CNT     EPZ TMP+1
0800          FLG     EPZ $CE
0800          FLH     EPZ $CF
0800          ;
0800          VAR     EQU $9800
0800          ;
98F3                  ORG $98F3
98F3          ;STUFF INTO DOS, RESET DOS PTRS
98F3          ;RESERVE VARIABLE SPACE
98F3          ;
98F3 A9D3             LDA #$D3
98F5 8D009D           STA $9D00
98F8 A997             LDA #$97
98FA 8D019D           STA $9D01
98FD 4CD4A7           JMP $A7D4
9900          ;
9900          ;RESERVE SIN TABLE SPACE
9900          ;
0867          TRG     DFS 90
995A          ;
995A          ;*********************************
995A          ;*                               *
995A          ;* CLEAR: AC=0                   *
995A          ;*                               *
995A          ;*********************************
995A          ;
995A A000     CLR     LDY #0
995C 84FA             STY ACL
995E 84FB             STY ACM
9960 84FC             STY ACH
9962 60               RTS
9963          ;
9963          ;*********************************
9963          ;*                               *
9963          ;* INCREMENT: AC=AC+1            *
9963          ;*                               *
9963          ;*********************************
9963          ;
9963 205C9C   INC#    JSR TMA
9966 E6FB     INC     INC ACM
9968 D002             BNE ICO
```

```
996A E6FC              INC ACH
996C 60         ICO    RTS
996D            ;
996D            ;*********************************
996D            ;*                               *
996D            ;* DECREMENT: AC=AC-1            *
996D            ;*                               *
996D            ;*********************************
996D            ;
996D 205C9C     DEC#   JSR TMA
9970 38         DEC    SEC
9971 A5FB              LDA ACM
9973 E901              SBC #1
9975 85FB              STA ACM
9977 B002              BCS DCO
9979 C6FC              DEC ACH
997B 60         DCO    RTS
997C            ;
997C            ;*********************************
997C            ;*                               *
997C            ;* INTEGER: AC=INT(AC+.5)        *
997C            ;*                               *
997C            ;*********************************
997C            ;
997C            ;ROUNDS INSTEAD OF TRUNCATES
997C            ;
997C 205C9C     INT#   JSR TMA
997F 24FA       INT    BIT ACL
9981 1003              BPL INO
9983 206699     TNO    JSR INC
9986 A200       INO    LDX #0
9988 86FA              STX ACL
998A 60         TOK    RTS
998B            ;
998B            ;*********************************
998B            ;*                               *
998B            ;* TRUNCATE: AC=INT(AC)          *
998B            ;*                               *
998B            ;*********************************
998B            ;
998B            ;TRUNCATES AS PER BASIC "INT"
998B            ;
998B 205C9C     TNC#   JSR TMA
998E A5FA       TNC    LDA ACL
9990 F0F8              BEQ TOK
9992 24FC              BIT ACH
9994 10F0              BPL INO
9996 30EB              BMI TNO
9998            ;
9998            ;*********************************
9998            ;*                               *
9998            ;* ADD: AC=AC+RC                 *
9998            ;*                               *
9998            ;*********************************
9998            ;
9998 205C9C     ADD##  JSR TMA
999B 206C9C     ADD#   JSR TMR
999E A2FD       ADD    LDX #$FD
99A0 18                CLC
99A1 B5FD       ADL    LDA SCL,X
99A3 75EE              ADC TMP,X
99A5 95FD              STA SCL,X
99A7 E8                INX
99A8 D0F7              BNE ADL
99AA 60                RTS
99AB            ;
99AB            ;*********************************
99AB            ;*                               *
99AB            ;* SUBTRACT: AC=AC-RC            *
99AB            ;*                               *
99AB            ;*********************************
99AB            ;
99AB 205C9C     SUB##  JSR TMA
99AE 206C9C     SUB#   JSR TMR
99B1 A2FD       SUB    LDX #$FD
```

```
99B3 38                 SEC
99B4 B5FD       SBL     LDA SCL,X
99B6 F5EE               SBC TMP,X
99B8 95FD               STA SCL,X
99BA E8                 INX
99BB D0F7               BNE SBL
99BD 60                 RTS
99BE            ;
99BE            ;**********************************
99BE            ;*                                *
99BE            ;* SIGN: A REG = SGN(AC)          *
99BE            ;*                                *
99BE            ;**********************************
99BE            ;
99BE            ; A REG CONDITIONED BY SIGN:
99BE            ;
99BE            ;A=0  FOR AC=0
99BE            ;A=1  FOR AC>0
99BE            ;A=FF FOR AC<0
99BE            ;
99BE 205C9C     SGN#    JSR TMA
99C1 A5FC       SGN     LDA ACH
99C3 3033               BMI CMI
99C5 D008               BNE CPL
99C7 A5FB               LDA ACM
99C9 D004               BNE CPL
99CB A5FA               LDA ACL
99CD F024               BEQ CEQ
99CF A901       CPL     LDA #1
99D1 60                 RTS
99D2            ;
99D2            ;**********************************
99D2            ;*                                *
99D2            ;* COMPARE: A REG = (AC)CMP(RC)   *
99D2            ;*                                *
99D2            ;**********************************
99D2            ;
99D2            ;A REG CONDITIONED BY COMPARE:
99D2            ;
99D2            ;A=0  FOR AC=RC
99D2            ;A=1  FOR AC>RC
99D2            ;A=FF FOR AC<RC
99D2            ;
99D2 205C9C     CMP##   JSR TMA
99D5 206C9C     CMP#    JSR TMR
99D8 A202       CMP     LDX #2
99DA A5FC               LDA ACH
99DC 3006               BMI CMX
99DE A4ED               LDY RCH
99E0 100A               BPL CLQ
99E2 30EB               BMI CPL
99E4 A4ED       CMX     LDY RCH
99E6 3004               BMI CLQ
99E8 100E               BPL CMI
99EA B5FA       CLP     LDA ACL,X
99EC D5EB       CLQ     CMP RCL,X
99EE D006               BNE CNE
99F0 CA                 DEX
99F1 10F7               BPL CLP
99F3 A900       CEQ     LDA #0
99F5 60                 RTS
99F6 B0D7       CNE     BCS CPL
99F8 A9FF       CMI     LDA #$FF
99FA 60                 RTS
99FB            ;
99FB            ;**********************************
99FB            ;*                                *
99FB            ;* ABSOLUTE VALUE: AC=ABS(AC)     *
99FB            ;* CHANGE SIGN: AC=-AC            *
99FB            ;*                                *
99FB            ;**********************************
99FB            ;
99FB 205C9C     CHG#    JSR TMA
99FE 4C089A             JMP CHG
9A01            ;
```

```
9A01 205C9C    ABS#    JSR TMA
9A04           ;
9A04           ;ABSOLUTE VALUE
9A04           ;
9A04 24FC      ABS     BIT ACH
9A06 100E              BPL CHO
9A08           ;
9A08           ;CHANGE SIGN
9A08           ;
9A08 A2FD      CHG     LDX #$FD
9A0A 38                SEC
9A0B B5FD      CHL     LDA SCL,X
9A0D 49FF              EOR #$FF
9A0F 6900              ADC #0
9A11 95FD              STA SCL,X
9A13 E8                INX
9A14 D0F5              BNE CHL
9A16 60        CHO     RTS
9A17           ;
9A17           ;*********************************
9A17           ;*                               *
9A17           ;* MULTIPLY: AC=AC*RC            *
9A17           ;*                               *
9A17           ;*********************************
9A17           ;
9A17 205C9C    MUL##   JSR TMA
9A1A 206C9C    MUL#    JSR TMR
9A1D 20009B    MUL     JSR CKS              ;NEG # ?
9A20 A5FA      MUL1    LDA ACL
9A22 85F9              STA ACX
9A24 A5FB              LDA ACM
9A26 85FA              STA ACL
9A28 A5FC              LDA ACH
9A2A 85FB              STA ACM
9A2C A018              LDY #24
9A2E A900              LDA #0
9A30 85FC              STA ACH
9A32 85FD              STA SCL
9A34 85FE              STA SCM
9A36 A5F9      MSHL    LDA ACX
9A38 4A                LSR
9A39 9013              BCC MROR
9A3B 18                CLC
9A3C A5FC              LDA ACH
9A3E 65EB              ADC RCL
9A40 85FC              STA ACH
9A42 A5FD              LDA SCL
9A44 65EC              ADC RCM
9A46 85FD              STA SCL
9A48 A5FE              LDA SCM
9A4A 65ED              ADC RCH
9A4C 85FE              STA SCM
9A4E 66FE      MROR    ROR SCM
9A50 66FD              ROR SCL
9A52 66FC              ROR ACH
9A54 66FB              ROR ACM
9A56 66FA              ROR ACL
9A58 66F9              ROR ACX
9A5A 88                DEY
9A5B D0D9              BNE MSHL
9A5D A6CE              LDX FLG
9A5F F003              BEQ MLO
9A61 20089A            JSR CHG              ;FIX FOR NEG #
9A64 60        MLO     RTS
9A65           ;
9A65           ;*********************************
9A65           ;*                               *
9A65           ;* DIVIDE: AC=AC/RC              *
9A65           ;*                               *
9A65           ;*********************************
9A65           ;
9A65 205C9C    DIV##   JSR TMA
9A68 206C9C    DIV#    JSR TMR
9A6B 20009B    DIV     JSR CKS              ;NEG # ?
9A6E A5FC      DIV1    LDA ACH
```

```
9A70 85FD            STA SCL
9A72 A5FB            LDA ACM
9A74 85FC            STA ACH
9A76 A5FA            LDA ACL
9A78 85FB            STA ACM
9A7A A018    DIV2    LDY #24
9A7C 84EF            STY CNT
9A7E A900            LDA #0
9A80 85FA            STA ACL
9A82 85FE            STA SCM
9A84 85FF            STA SCH
9A86 06FA    DIVL    ASL ACL
9A88 26FB            ROL ACM
9A8A 26FC            ROL ACH
9A8C 26FD            ROL SCL
9A8E 26FE            ROL SCM
9A90 26FF            ROL SCH
9A92 38              SEC
9A93 A5FD            LDA SCL
9A95 E5EB            SBC RCL
9A97 AA              TAX
9A98 A5FE            LDA SCM
9A9A E5EC            SBC RCM
9A9C A8              TAY
9A9D A5FF            LDA SCH
9A9F E5ED            SBC RCH
9AA1 9008            BCC DIVS
9AA3 86FD            STX SCL
9AA5 84FE            STY SCM
9AA7 85FF            STA SCH
9AA9 E6FA            INC ACL
9AAB C6EF    DIVS    DEC CNT
9AAD D0D7            BNE DIVL
9AAF A6CE            LDX FLG
9AB1 F003            BEQ DVO
9AB3 20089A          JSR CHG                  ;FIX FOR NEG #
9AB6 60      DVO     RTS
9AB7                 ;
9AB7                 ;*********************************
9AB7                 ;*                               *
9AB7                 ;* INVERT: AC=1/AC               *
9AB7                 ;*                               *
9AB7                 ;*********************************
9AB7                 ;
9AB7 205C9C  INV#    JSR TMA
9ABA 20869C  INV     JSR TAR
9ABD A900            LDA #0
9ABF 85FB            STA ACM
9AC1 85FD            STA SCL
9AC3 A901            LDA #1
9AC5 85FC            STA ACH
9AC7 20009B          JSR CKS
9ACA 4C7A9A          JMP DIV2
9ACD                 ;
9ACD                 ;*********************************
9ACD                 ;*                               *
9ACD                 ;* SQUARE ROOT: AC=SQR(AC)       *
9ACD                 ;*                               *
9ACD                 ;*********************************
9ACD                 ;
9ACD                 ;NEWTON-RAPHSON SQUARE ROOT
9ACD                 ;
9ACD                 ;STORE ARGUMENT
9ACD                 ;AND LOOP COUNT
9ACD                 ;
9ACD 205C9C  SQR#    JSR TMA
9AD0 20909C  SQR     JSR TAT
9AD3 A910            LDA #$10
9AD5 85EE            STA TMP
9AD7                 ;
9AD7                 ;INITIAL GUESS = 0
9AD7                 ;
9AD7 A900            LDA #0
9AD9 85CE            STA FLG
9ADB 85EB            STA RCL
```

```
9ADD 85ED              STA RCH
9ADF 85EC              STA RCM
9AE1            ;
9AE1            ;ADD GUESS TO ARG/GUESS
9AE1            ;AND DIVIDE BY TWO
9AE1            ;
9AE1 209E99     SQL    JSR ADD
9AE4 18                CLC
9AE5 66FC              ROR ACH
9AE7 66FB              ROR ACM
9AE9 66FA              ROR ACL
9AEB            ;
9AEB            ;GUESS = OLD GUESS ?
9AEB            ;
9AEB 20D899            JSR CMP
9AEE F00F              BEQ SQO
9AF0 C6EE              DEC TMP
9AF2 F00B              BEQ SQO
9AF4            ;
9AF4            ;STORE NEW GUESS
9AF4            ;
9AF4 20869C            JSR TAR
9AF7            ;
9AF7            ;FETCH ARGUMENT AND
9AF7            ;DIVIDE BY GUESS
9AF7            ;
9AF7 20A49C            JSR TTA
9AFA 206E9A            JSR DIV1
9AFD F0E2              BEQ SQL
9AFF            ;
9AFF 60         SQO    RTS
9B00            ;
9B00            ;**********************************
9B00            ;*                                *
9B00            ;* CHECK SIGN SUBROUTINE          *
9B00            ;* FOR MULTIPLY AND DIVIDE        *
9B00            ;*                                *
9B00            ;**********************************
9B00            ;
9B00 A000       CKS    LDY #0
9B02 24FC              BIT ACH
9B04 1004              BPL CKA
9B06 20089A            JSR CHG
9B09 88                DEY
9B0A 24ED       CKA    BIT RCH
9B0C 100F              BPL CKB
9B0E C8                INY
9B0F A2FD       CGR    LDX #$FD
9B11 38                SEC
9B12 B5EE       CKL    LDA TMP,X
9B14 49FF              EOR #$FF
9B16 6900              ADC #0
9B18 95EE              STA TMP,X
9B1A E8                INX
9B1B D0F5              BNE CKL
9B1D 84CE       CKB    STY FLG
9B1F 60                RTS
9B20            ;
9B20            ;**********************************
9B20            ;*                                *
9B20            ;* FIX ANGLE SUBROUTINE:          *
9B20            ;* INSURES 0 <= AC < 360          *
9B20            ;*                                *
9B20            ;**********************************
9B20            ;
9B20            ;RC=360
9B20            ;
9B20 A900       FXA    LDA #0
9B22 A268              LDX #104
9B24 A001              LDY #1
9B26 85EB              STA RCL
9B28 86EC              STX RCM
9B2A 84ED              STY RCH
9B2C            ;
9B2C            ;ANGLE NEGATIVE ?
```

```
9B2C                ;
9B2C A5FC                   LDA ACH
9B2E 100A                   BPL FXB           ;NO
9B30                ;
9B30                ;FIX NEG ANGLE
9B30                ;ADD 360 UNTIL AC>=0
9B30                ;
9B30 209E99         FXN     JSR ADD
9B33 AA                     TAX
9B34 30FA                   BMI FXN
9B36 60                     RTS
9B37                ;
9B37                ;FIX POS ANGLE
9B37                ;SUB 360 UNTIL AC<360
9B37                ;
9B37 20B199         FXP     JSR SUB
9B3A C901           FXB     CMP #1            ;AC>360
9B3C 9008                   BCC FXO
9B3E D0F7                   BNE FXP           ;YES
9B40 A5FB                   LDA ACM
9B42 C968                   CMP #104
9B44 B0F1                   BCS FXP           ;YES
9B46                ;
9B46                ;ANGLE OK
9B46                ;
9B46 60             FXO     RTS
9B47                ;
9B47                ;*********************************
9B47                ;*                               *
9B47                ;* SINE: AC=SIN(AC)              *
9B47                ;*                               *
9B47                ;*********************************
9B47                ;
9B47 205C9C         SIN#    JSR TMA
9B4A 20209B         SIN     JSR FXA
9B4D 207F99                 JSR INT
9B50 A000                   LDY #0
9B52 84CE                   STY FLG           ;SIGN +
9B54                ;
9B54                ;REDUCE ANGLE TO <= 90
9B54                ;
9B54 38                     SEC
9B55 A5FC                   LDA ACH
9B57 D006                   BNE SNA           ;AC>255
9B59 A5FB                   LDA ACM
9B5B C9B5                   CMP #181
9B5D 9008                   BCC SNB           ;AC<=180
9B5F A968           SNA     LDA #104
9B61 E5FB                   SBC ACM
9B63 85FB                   STA ACM
9B65 C6CE                   DEC FLG           ;SIGN -
9B67 C95B           SNB     CMP #91
9B69 9004                   BCC SNG           ;AC<=90
9B6B A9B4                   LDA #180
9B6D E5FB                   SBC ACM
9B6F                ;
9B6F                ;ANGLE IN A, GET AC=SIN(A)
9B6F                ;
9B6F 205A99         SNG     JSR CLR
9B72 C957                   CMP #87
9B74 9004                   BCC SNT
9B76 E6FB                   INC ACM           ;A>86
9B78 1006                   BPL SNS
9B7A AA             SNT     TAX
9B7B BD0099                 LDA TRG,X
9B7E 85FA                   STA ACL
9B80                ;
9B80                ;NEG VALUE ?

9B80                ;
9B80 24CE           SNS     BIT FLG
9B82 1003                   BPL SNO
9B84 20089A                 JSR CHG
9B87 60             SNO     RTS
9B88                ;
```

```
9B88              ;**********************************
9B88              ;*                                *
9B88              ;* COSINE: AC=COS(AC)             *
9B88              ;*                                *
9B88              ;**********************************
9B88              ;
9B88 205C9C       COS#    JSR TMA
9B8B 20ED9B       COS     JSR CPA            ;90-A
9B8E 4C4A9B               JMP SIN
9B91              ;
9B91              ;**********************************
9B91              ;*                                *
9B91              ;* TANGENT: AC=TAN(AC)            *
9B91              ;*                                *
9B91              ;**********************************
9B91              ;
9B91 205C9C       TAN#    JSR TMA
9B94 20909C       TAN     JSR TAT
9B97 208B9B               JSR COS
9B9A 20C199               JSR SGN
9B9D D004                 BNE TNQ
9B9F A902                 LDA #2
9BA1 85FA                 STA ACL
9BA3 209A9C       TNQ     JSR TAQ
9BA6 20A49C               JSR TTA
9BA9 204A9B               JSR SIN
9BAC 20C29C               JSR TQR
9BAF 206B9A               JSR DIV
9BB2 60                   RTS
9BB3              ;
9BB3              ;**********************************
9BB3              ;*                                *
9BB3              ;* ARCSINE: AC=ASN(AC)            *
9BB3              ;*                                *
9BB3              ;**********************************
9BB3              ;
9BB3 205C9C       ASN#    JSR TMA
9BB6 A000         ASN     LDY #0
9BB8 84CE                 STY FLG
9BBA A6FC         ASN1    LDX ACH
9BBC 1006                 BPL ASC
9BBE 20089A               JSR CHG
9BC1 C6CE                 DEC FLG
9BC3 AA                   TAX
9BC4 D004         ASC     BNE AOV
9BC6 C6FB                 DEC ACM
9BC8 3004                 BMI ASG
9BCA A25A         AOV     LDX #$5A
9BCC 100C                 BPL ASF
9BCE A257         ASG     LDX #$57
9BD0 A5FA                 LDA ACL
9BD2 CA           ASL     DEX
9BD3 F005                 BEQ ASF
9BD5 DD0099               CMP TRG,X
9BD8 90F8                 BCC ASL
9BDA 205A99       ASF     JSR CLR
9BDD 86FB                 STX ACM
9BDF 24CE                 BIT FLG
9BE1 1003                 BPL ASO
9BE3 20089A       TNA     JSR CHG
9BE6 60           ASO     RTS
9BE7              ;
9BE7              ;**********************************
9BE7              ;*                                *
9BE7              ;* ARCCOSINE: AC=ACS(AC)          *
9BE7              ;*                                *
9BE7              ;**********************************
9BE7              ;
9BE7 205C9C       ACS#    JSR TMA
9BEA 20B69B       ACS     JSR ASN            ;AC=ASN
9BED              ;
9BED              ;AC=90-AC
9BED              ;
9BED 38           CPA     SEC
9BEE A900                 LDA #0
```

```
9BF0 E5FA              SBC ACL
9BF2 85FA              STA ACL
9BF4 A95A              LDA #90
9BF6 E5FB              SBC ACM
9BF8 85FB              STA ACM
9BFA A900              LDA #0
9BFC E5FC              SBC ACH
9BFE 85FC              STA ACH
9C00 60                RTS
9C01            ;
9C01            ;**********************************
9C01            ;*                                *
9C01            ;* ARCTANGENT: AC=ATN(AC)         *
9C01            ;*                                *
9C01            ;**********************************
9C01            ;
9C01 205C9C     ATN#   JSR TMA
9C04 20C199     ATN    JSR SGN
9C07 85CF              STA FLH
9C09 85CE              STA FLG
9C0B 1003              BPL ATP
9C0D 20089A            JSR CHG
9C10 A5FC       ATP    LDA ACH
9C12 D0B6              BNE AOV
9C14 A5FB              LDA ACM
9C16 30B2              BMI AOV
9C18 209A9C            JSR TAQ
9C1B 20869C            JSR TAR
9C1E A200              LDX #0
9C20 86CE              STX FLG
9C22 20209A            JSR MUL1
9C25 206699            JSR INC
9C28 20D09A            JSR SQR
9C2B A504              LDA QCM
9C2D D00E              BNE ATM
9C2F 20869C            JSR TAR
9C32 20B89C            JSR TQA
9C35 206E9A            JSR DIV1
9C38 20BA9B            JSR ASN1
9C3B 1006              BPL ATF
9C3D 20BA9A     ATM    JSR INV
9C40 20EA9B            JSR ACS
9C43 24CF       ATF    BIT FLH
9C45 1003              BPL ATO
9C47 20089A            JSR CHG
9C4A 60         ATO    RTS
9C4B            ;
9C4B            ;**********************************
9C4B            ;*                                *
9C4B            ;* LET: VAR1=VAR2                 *
9C4B            ;*                                *
9C4B            ;**********************************
9C4B            ;
9C4B            ;ENTRY: POINTER TO VAR#1 IN Y AND
9C4B            ;POINTER TO VAR#2 IN X
9C4B            ;
9C4B A903       LET##  LDA #3
9C4D 85EF              STA CNT
9C4F BD0098     LTL    LDA VAR,X
9C52 990098            STA VAR,Y
9C55 E8                INX
9C56 C8                INY
9C57 C6EF              DEC CNT
9C59 D0F4              BNE LTL
9C5B 60                RTS
9C5C            ;
9C5C            ;**********************************
9C5C            ;*                                *
9C5C            ;* FETCH AC: AC=VARIABLE          *
9C5C            ;*                                *
9C5C            ;**********************************
9C5C            ;
9C5C            ;TRANSFERS VARIABLE TO AC
9C5C            ;ENTER WITH VAR PTR IN Y
9C5C            ;PROTECTS PTR IN X
```

```
9C5C              ;
9C5C 86EE         TMA     STX TMP
9C5E A2FD                 LDX #$FD
9C60 B90098       MPL     LDA VAR,Y
9C63 95FD                 STA SCL,X
9C65 C8                   INY
9C66 E8                   INX
9C67 D0F7                 BNE MPL
9C69 A6EE                 LDX TMP
9C6B 60                   RTS
9C6C              ;
9C6C              ;**********************************
9C6C              ;*                                *
9C6C              ;*  FETCH RC: RC=VARIABLE         *
9C6C              ;*                                *
9C6C              ;**********************************
9C6C              ;
9C6C              ;TRANSFERS VARIABLE TO RC
9C6C              ;ENTER WITH VAR PTR IN X
9C6C              ;
9C6C 8A           TMR     TXA
9C6D A8                   TAY
9C6E A2FD                 LDX #$FD
9C70 B90098       MRL     LDA VAR,Y
9C73 95EE                 STA TMP,X
9C75 C8                   INY
9C76 E8                   INX
9C77 D0F7                 BNE MRL
9C79 60                   RTS
9C7A              ;
9C7A              ;**********************************
9C7A              ;*                                *
9C7A              ;*  STORE: VARIABLE=AC            *
9C7A              ;*                                *
9C7A              ;**********************************
9C7A              ;
9C7A              ;STORES A RESULT FROM AC
9C7A              ;INTO A VARIABLE LOCATION
9C7A              ;
9C7A              ;ENTER WITH VAR PTR IN Y
9C7A              ;
9C7A A2FD         STR     LDX #$FD
9C7C B5FD         STL     LDA SCL,X
9C7E 990098               STA VAR,Y
9C81 C8                   INY
9C82 E8                   INX
9C83 D0F7                 BNE STL
9C85 60                   RTS
9C86              ;


9C86              ;**********************************
9C86              ;*                                *
9C86              ;*  TRANSFER AC TO RC             *
9C86              ;*                                *
9C86              ;**********************************
9C86              ;
9C86 A202         TAR     LDX #2
9C88 B5FA         TRL     LDA ACL,X
9C8A 95EB                 STA RCL,X
9C8C CA                   DEX
9C8D 10F9                 BPL TRL
9C8F 60                   RTS
9C90              ;
9C90              ;**********************************
9C90              ;*                                *
9C90              ;*  TRANSFER AC TO TC             *
9C90              ;*                                *
9C90              ;**********************************
9C90              ;
9C90 A202         TAT     LDX #2
9C92 B5FA         TAL     LDA ACL,X
9C94 9507                 STA TCL,X
9C96 CA                   DEX
9C97 10F9                 BPL TAL
9C99 60                   RTS
```

```
9C9A           ;
9C9A           ;**********************************
9C9A           ;*                                *
9C9A           ;* TRANSFER AC TO QC              *
9C9A           ;*                                *
9C9A           ;**********************************
9C9A           ;
9C9A A202      TAQ     LDX #2
9C9C B5FA      AQL     LDA ACL,X
9C9E 9503              STA QCL,X
9CA0 CA                DEX
9CA1 10F9              BPL AQL
9CA3 60                RTS
9CA4           ;
9CA4           ;**********************************
9CA4           ;*                                *
9CA4           ;* TRANSFER TC TO AC              *
9CA4           ;*                                *
9CA4           ;**********************************
9CA4           ;
9CA4 A202      TTA     LDX #2
9CA6 B507      TTL     LDA TCL,X
9CA8 95FA              STA ACL,X
9CAA CA                DEX
9CAB 10F9              BPL TTL
9CAD 60                RTS
9CAE           ;
9CAE           ;**********************************
9CAE           ;*                                *
9CAE           ;* TRANSFER TC TO RC              *
9CAE           ;*                                *
9CAE           ;**********************************
9CAE           ;
9CAE A202      TTR     LDX #2
9CB0 B507      TXL     LDA TCL,X
9CB2 95EB              STA RCL,X
9CB4 CA                DEX
9CB5 10F9              BPL TXL
9CB7 60                RTS
9CB8           ;
9CB8           ;**********************************
9CB8           ;*                                *
9CB8           ;* TRANSFER QC TO AC              *
9CB8           ;*                                *
9CB8           ;**********************************
9CB8           ;
9CB8 A202      TQA     LDX #2
9CBA B503      QAL     LDA QCL,X
9CBC 95FA              STA ACL,X
9CBE CA                DEX
9CBF 10F9              BPL QAL
9CC1 60                RTS
9CC2           ;
9CC2           ;**********************************
9CC2           ;*                                *
9CC2           ;* TRANSFER QC TO RC              *
9CC2           ;*                                *
9CC2           ;**********************************
9CC2           ;
9CC2 A202      TQR     LDX #2
9CC4 B503      QRL     LDA QCL,X
9CC6 95EB              STA RCL,X
9CC8 CA                DEX
9CC9 10F9              BPL QRL
9CCB 60                RTS
9CCC           ;
                       END
```

Listing 2

```
;************************
;*                     *
;*  EQUATES FOR 24-BIT  *
;*     MATH PACKAGE     *
;************************
;
;PRINCIPAL ENTRY POINTS
;ASSUME AC (AND RC) LOADED
;
CLR       EQU $995A          ;USES Y,AC
INC       EQU $9966          ;USES AC
DEC       EQU $9970          ;USES A,AC
INT       EQU $997F          ;USES X,AC
TNC       EQU $998E          ;USES A,X,AC
ADD       EQU $999E          ;USES A,X,AC
SUB       EQU $99B1          ;USES A,X,AC
SGN       EQU $99C1          ;USES A
CMP       EQU $99D8          ;USES A,X,Y
ABS       EQU $9A04          ;USES A,X,AC
CHG       EQU $9A08          ;USES A,X,AC
MUL       EQU $9A1D          ;USES A,X,Y,AC,FLG
DIV       EQU $9A6B          ;USES A,X,Y,AC,FLG,CNT
INV       EQU $9ABA          ;USES A,X,Y,AC,RC,FLG,CNT
SQR       EQU $9AD0          ;USES A,X,Y,AC,RC,TC,FLG,CNT,TMP
FXA       EQU $9B20          ;USES A,X,Y,AC,RC
SIN       EQU $9B4A          ;USES A,X,Y,AC,RC,FLG
COS       EQU $9B8B          ;USES A,X,Y,AC,RC,FLG
TAN       EQU $9B94          ;USES A,X,Y,AC,RC,TC,QC,FLG,CNT
ASN       EQU $9BB6          ;USES A,X,Y,AC,FLG
ACS       EQU $9BEA          ;USES A,X,Y,AC,FLG
ATN       EQU $9C04          ;USES A,X,Y,AC,RC,TC,QC,FLG,FLH,CNT,TMP
TMA       EQU $9C5C          ;Y>VAR. USES A,Y,AC,TMP
TMR       EQU $9C6C          ;X>VAR. USES A,X,Y,RC
STR       EQU $9C7A          ;Y>VAR. USES A,X,Y
TAR       EQU $9C86          ;USES A,X,RC
TAT       EQU $9C90          ;USES A,X,TC
TAQ       EQU $9C9A          ;USES A,X,TQ
TTA       EQU $9CA4          ;USES A,X,AC
TTR       EQU $9CAE          ;USES A,X,RC
TQA       EQU $9CB8          ;USES A,X,AC
TQR       EQU $9CC2          ;USES A,X,RC

UNARY OPERATORS: LOADING ENTRY POINTS
LOADS AC WITH VARIABLE INDEXED BY Y-REG

INC# EQU $9963 ;Y>AC
DEC# EQU $996D ;Y>AC
INT# EQU $997C ;Y>AC
TNC# EQU $998B ;Y>AC
SGN# EQU $99BE ;Y>AC
ABS# EQU $9A01 ;Y>AC
CHG# EQU $99FB ;Y>AC
INV# EQU $9AB7 ;Y>AC
SQR# EQU $9ACD ;Y>AC
SIN# EQU $9B47 ;Y>AC
COS# EQU $9B88 ;Y>AC
TAN# EQU $9B91 ;Y>AC
ASN# EQU $9BB3 ;Y>AC
ACS# EQU $9BE7 ;Y>AC
ATN# EQU $9C01 ;Y>AC

;BINARY OPERATORS: FULL LOADING ENTRY POINTS
;LOADS AC WITH VARIABLE INDEXED BY Y-REG,AND
;LOADS RC WITH VARIABLE INDEXED BY X-REG

ADD## EQU $9998 ;Y>AC, X>RC
SUB## EQU $99AB ;Y>AC, X>RC
CMP## EQU $99D2 ;Y>AC, X>RC
MUL## EQU $9A17 ;Y>AC, X>RC
DIV## EQU $9A65 ;Y>AC, X>RC
LET## EQU $9C4B ;Y=X. USES ONLY A,X,Y,CNT
```

```
;BINARY OPERATORS:HALF LOADING ENTRY POINTS
;LOADS RC WITH VARIABLE INDEXED BY X-REG
;ASSUMES AC ALREADY LOADED
;
ADD# EQU $999B ;X>RC
SUB# EQU $99AE ;X>RC
CMP# EQU $99D5 ;X>RC
MUL# EQU $9A1A ;X>RC
DIV# EQU $9A68 ;X>RC

         END
```

Listing 3

```
0800             ;********************************
0800             ;*                              *
0800             ;*   EXAMPLE CODE USING FIAT    *
0800             ;*                              *
0800             ;********************************
0800             ;
0800             ;FIAT EQUATES
0800             ;
0800             INC     EQU $9966
0800             ADD     EQU $999E
0800             CMP     EQU $99D8
0800             DIV     EQU $9A6B
0800             SQR     EQU $9AD0
0800             STR     EQU $9C7A
0800             TAT     EQU $9C90
0800             TAQ     EQU $9C9A
0800             TTR     EQU $9CAE
0800             TQR     EQU $9CC2
0800             MUL#    EQU $9A1A
0800             SIN#    EQU $9B47
0800             SUB##   EQU $99AB
0800             MUL##   EQU $9A17
0800             LET##   EQU $9C4B
0800             ;
0800             ;VARIABLE EQUATES
0800             ;
0800             A       EPZ $00
0800             B       EPZ $03
0800             C       EPZ $06
0800             D       EPZ $09
0800             E       EPZ $0C
0800             F       EPZ $0F
0800             PI      EPZ $12
0800             ;
0800             ;B=(A-B)*C+1
0800             ;
0800 A000                LDY #A
0802 A203                LDX #B
0804 20AB99              JSR SUB##
0807 A206                LDX #C
0809 201A9A              JSR MUL#
080C 206699              JSR INC
080F A003                LDY #B
0811 207A9C              JSR STR
0814             ;
0814             ;F=PI*SQR(A*B+C*D)/SIN(E)
0814             ;
0814 A00C                LDY #E
0816 20479B              JSR SIN#
0819 209A9C              JSR TAQ
081C A000                LDY #A
081E A203                LDX #B
0820 20179A              JSR MUL##
0823 20909C              JSR TAT
0826 A006                LDY #C
0828 A209                LDX #D
```

```
082A 20179A          JSR MUL##
082D 20AE9C          JSR TTR
0830 209E99          JSR ADD
0833 A212            LDX #PI
0835 201A9A          JSR MUL#
0838 20C29C          JSR TOR
083B 206B9A          JSR DIV
083E A00F            LDY #F
0840 207A9C          JSR STR
0843            ;
0843            ;IF A>B THEN C=B
0843            ;
0843 A003            LDY #B
0845 A200            LDX #A
0847 20D899          JSR CMP
084A 1007            BPL NO
084C A006            LDY #C
084E A209            LDX #D
0850 204B9C          JSR LET##
0853 60         NO   RTS
0854            ;
0854            ;
                    END
```

```
10  REM  *************************
20  REM  *                       *
30  REM  *      FIAT LOADER      *
40  REM  *                       *
50  REM  *      WES HUNTRESS     *
60  REM  *                       *
70  REM  *  COPYRIGHT (C) 1982   *
80  REM  *    MICRO INK, INC.    *
90  REM  * CHELMSFORD, MA. 01824 *
92  REM  *                       *
94  REM  *************************
100  POKE  - 25344,211
110  POKE  - 25343,151
120  CALL  - 22572
130  PRINT  CHR$ (4)"BLOAD FIAT"
```

Applesoft Error Messages from Machine Language

by Steve Cochard

The methods and data required to utilize Applesoft error messages in assembly language are presented. Use of these routines should be limited to assembly language routines that are interfaced with Applesoft programs.

I needed to know more about how Applesoft generates its error messages. While writing an assembly language program that interfaced with Applesoft, I found that just the simple "syntax error," which was the only message I knew how to utilize, was not enough.

I started my search for the "errors" by looking at the machine code for the "syntax error" message which is located at $DEC9. It consists of only two commands:

```
LDX #$10
JMP $D412
```

This short routine was intended only to load the X register with the starting address of the word SYNTAX in a table of all error messages. With a little more searching in the $D412 routine, the table was found.

The 240-byte-long error message table is located at $D260. By loading the X register with the appropriate index and then jumping to the $D412 routine, it is possible to utilize any error message from machine language or Applesoft.

Table 1 shows the values to be loaded into the X register to generate any of the available 17 messages. Listings 1 and 2 show very short machine and Applesoft programs to verify that this is true. Listing 3 shows a program that will list the entire table.

Note that this procedure, if utilized in machine language, performs exactly as if the error had occurred in an Applesoft program. The error message is printed, the bell rings, the last executed line number is printed, and the program stops. If an ONERR GOTO statement was already executed, the program will again operate as if the error had occurred in Applesoft. The object line of the ONERR GOTO will be jumped to and executed. Happy Errors!

Table 1: Value of X register and error messages.

0	NEXT WITHOUT FOR	107	BAD SUBSCRIPT
16	SYNTAX	120	REDIM'D ARRAY
22	RETURN WITHOUT GOSUB	133	DIVISION BY ZERO
42	OUT OF DATA	149	ILLEGAL DIRECT
53	ILLEGAL QUANTITY	163	TYPE MISMATCH
69	OVERFLOW	176	STRING TOO LONG
77	OUT OF MEMORY	191	FORMULA TOO COMPLEX
90	UNDEF'D STATEMENT	210	CAN'T CONTINUE
		224	UNDEF'D FUNCTION

Listing 1: Enter from the monitor to interface with program listing 2.

```
300:LDX $0306
303:JMP $D412
```

Listing 2: Applesoft program to print error messages.

```
10 INPUT "WHAT VALUE OF X ? ";X
20 POKE 774,X
30 CALL 768
```

Listing 3: Lists the entire table. Enter it from the monitor and then type in 300G.

```
300:LDX #$00
302:LDA $D260,X
305:EOR #$80
307:BMI $0310
309:ORA #$80
30B:JSR $FDED
30E:LDA #$8D
310:JSR $FDED
313:INX
314:CPX #$FF
316:BNE $0302
318:RTS
```

3
I/O ENHANCEMENTS

I/O Enhancements

You can improve communication with your computer by using any of these handy programs.

Sandy Mossberg's "Trick DOS" will allow you to change DOS commands; as a result you can create abbreviations for the commands, or completely change them.

"Binary File Parameter List" by Clyde Camp not only gives you the ability to see the location of the default address for binary type files, but also displays their lengths. N.R. McBurney's "LACRAB," an effective Applesoft BASIC listing-formatter and cross-reference program, improves the look and readability of your listing. The program utilizes features such as single statement lines and logical indentation.

"Serial Line Editor" by Wes Huntress is an improvement over the monitor ROM line input routine. It provides a better delete and insert character routine (the line appears the way it is stored), move cursor to beginning or end of line command, move cursor to first occurrence of a specified character command, and other features. The author offers methods to interface the Line Editor to any Applesoft program.

Serial Line Editor

by Wes Huntress

This routine is an extended line editor that allows inserting, deleting, and several other features.

The GETLN machine-language routine replaces your Apple's line input routine (resident in monitor ROM). Both Applesoft and Integer BASICs call this routine for line input. The advantage of the alternate routine given here is the editing features it contains. The Apple monitor ESC editing features are very useful for editing BASIC program lines, but are not the best for editing text. The editing features in GETLN are typical of serial text line editing and could form the basis of any line-oriented text processing program. GETLN also allows the input of normally forbidden characters in Applesoft, such as the comma and colon.

All of these advantages are gained at a slight disadvantage in usage. Applesoft programs must be moved up two pages in memory and a few extra program steps are required instead of a simple INPUT statement. GETLN should be used only for string input and string editing. The version given here is for Applesoft. With a few changes it can be made to work for Integer as well.

When called, GETLN prompts for input and places the characters in the keyboard buffer at $200.2FF. All editing is done on the characters placed in the keyboard buffer. On return from GETLN it is necessary to move the characters from the keyboard buffer to the memory space that is to be occupied by the string. For Applesoft, this requires that the location in memory of the string variable's address pointer be known. The method used to accomplish this is the same as given in *CONTACT#6*. A dummy variable is declared as the first variable in the program, i.e. X$ = '' '', which assigns the two-byte variable name to the first two locations in memory at the LOMEM: pointer. The third location is assigned to the string length, and the fourth and fifth locations to the address of the string in memory, low byte first.

The LOMEM: pointer is at $69-70, so that the address of the string X$ can now be found indirectly from the LOMEM: pointer. A separate machine language program, called GI, is provided. It interfaces the GETLN routine with Applesoft programs by placing the address of the keyboard buffer and the buffer string length into the proper location for X$ using the LOMEM: pointer.

The string X$ is now assigned to the string in the keyboard buffer. In order to move it into the upper part of memory where Applesoft strings are normally stored, and to prevent the string from being clobbered the next time GETLN is called, the statement X$=MID$(X$,1) is used. This statement performs a memory move from the present location of X$ (the keyboard buffer) to the next available space in high memory, and is the key to the success of the interface of GETLN with Applesoft programs.

How to Use It

To use GETLN with Applesoft programs, both GI and GETLN must be present in memory. To set up your program and call for input, use the following procedure:

```
5 X$=" ":REM FIRST VARIABLE DECLARATION
.
.
.
.
100 CALL 840:A$=MID$(X$,1):REM KEYBOARD INPUT
```

Line 100 replaces the INPUT A$ statement. CALL 834 is to the keyboard input entry point in the GI interface routine. Three other entry points are provided in the interface routine. The call

```
100 CALL 859:X$=MID$(X$,1):REM DOS INPUT
```

replaces the INPUT A$ statement when READing text files from the disk. A separate routine from the keyboard input routine is required for Applesoft programs since the DOS stores and outputs all text files in negative ASCII. The call

```
100 X$=A$:CALL 806:REM PRINT
```

can be used in place of the PRINT A$ statement to print all control characters in inverse video. Otherwise use the PRINT A$ statement as usual. To recall a string for further editing, use

```
100 X$=A$:CALL 813:A$=MID$(X$,1):REM EDIT
```

The cursor will be placed on the screen at the beginning of the recalled string. Dimensioned strings can be used as well as simple strings. GETLN can also be used alone from assembly language using 800G. It will place the input string in the keyboard buffer in standard ASCII terminated by $8D (CR).

GETLN occupies nearly two pages of memory from $800 to $9AF. Since Applesoft programs normally reside in this space, it is necessary to move your program up in memory to make room for GETLN. This is readily accomplished by two statements:

```
POKE 104,10:POKE 2560,0
```

This line must be executed either from immediate mode or from an EXEC file before loading the Applesoft program. The short interface routine occupies locations $300 to $360.

Editing Features

The following edit commands are implemented in GETLN. Except for the usual Apple ←,→ and RETURN editing keys, all commands are initiated by hitting the ESC key.

→	Move cursor right, copy character
←	Move cursor left
RETURN	Terminate line, clear to end of page
ESC →	Initiate insert mode, ESC or RET to exit
ESC ←	Delete character, recursive
ESC sp bar	Move cursor to beginning (end) of line
ESC char	Move cursor to first occurrence of char
ESC ctrl-shift-M	Delete remainder of line

The first three commands operate just as in the Apple monitor line editor. The monitor ESC functions are replaced with the five ESC functions listed above. Use ESC → to insert characters at any place in the line. Use the usual monitor → and ← keys to position the cursor over the character where you wish to insert. ESC → will push right by one character the entire string beginning from the character under the cursor to the end of the line, leaving a blank under the cursor. As you type in new characters, the old right-hand string is continuously shifted right. The ← and → keys work on the inserted substring as before but will not allow editing left of the first inserted character. In the insert mode, → operates just like the space bar if keyed at the right-hand end of the substring. To terminate the insert mode, press ESC or RETURN. The old right-hand string is moved back one space for reconnection.

The ESC ← command deletes the character under the cursor and pulls left the entire string to the right of the cursor. The function is recursive, so that characters can continue to be deleted by repeated keying of the ← key. The first key pressed other than ← terminates the function.

The ESC space bar command moves the cursor to the end of the line. If the cursor is already at the end of the line, then it is moved to the beginning. This function allows rapid transport of the cursor to the beginning or end of the line.

The ESC char command moves the cursor right in the line to the first occurrence of the character key pressed after the escape key. If the character is not found before the end of the line, then the search branches to the beginning of the line. If the character is not found in the line, then the cursor is not moved.

The ESC ctrl-shift-M command deletes the entire line to the right of the cursor including the character under the cursor. This function allows excess garbage to be cleared from the line for editing readability.

Together these functions give you an intriguing and powerful text line editor. It's much more fun than the Apple monitor line input routine. Try it! You'll like it!

```
0800             ;*********************
0800             ;*                   *
0800             ;* SERIAL LINE EDITOR *
0800             ;*   FOR APPLESOFT   *
0800             ;*                   *
0800             ;*        BY         *
0800             ;*                   *
0800             ;*   WES HUNTRESS    *
0800             ;*  SIERRA MADRE, CA *
0800             ;*                   *
0800             ;* COPYRIGHT (C) 1982 *
0800             ;*   MICRO INK, INC.  *
0800             ;*CHELMSFORD, MA 01824*
0800             ;* ALL RIGHTS RESERVED*
0800             ;*                   *
0800             ;*********************
0800             ;
0800             ;EQUATES: CONSTANTS
0800             ;
0800             BS      EPZ $88
0800             CR      EPZ $8D
0800             CSM     EPZ $9D
0800             CTL     EPZ $20
0800             ESC     EPZ $9B
0800             FIX     EPZ $7F
0800             INV     EPZ $80
0800             NAK     EPZ $95
0800             BEND    EPZ $FE
0800             ZERO    EPZ $00
0800             BLANK   EPZ $A0
0800             ;
0800             ;EQUATES: POINTERS
0800             ;
0800             CHAR#   EPZ $19
0800             EOL     EPZ $1A
0800             STRT    EPZ $1B
0800             TEMP    EPZ $1C
0800             SUBSTR  EPZ $1D
0800             SUBEND  EPZ $1E
0800             MODE    EPZ $1F
0800             ;
0800             ;EQUATES: MONITOR ADDRESSES
0800             ;
0800             BUFFER EQU $0200
0800             KEYIN  EQU $FD0C
0800             PRINT  EQU $FDED
0800             BACKSP EQU $FC10
0800             ADVANC EQU $FBF4
0800             RETURN EQU $FC62
0800             CLREOP EQU $FC42
0800             BELL   EQU $FF3A
0800             ;
0800                    ORG $0800
0800             ;
0800             ;INITIALIZE KEYBOARD BUFFER
0800             ;
0800 A0A0        GETLN  LDY #BLANK        ;LOAD BLANK CHARACTER
0802 8C0002      CLRB   STY BUFFER        ;STORE IT IN KEYBOARD BUFFER
0805 EE0308             INC *-$2          ;FROM $0200
0808 D0F8               BNE CLRB          ;TO $02FF
080A A200               LDX #ZERO         ;SET POINTERS TO ZERO:
080C 8619               STX CHAR#         ;CHARACTER NUMBER IN THE STRING
080E 861A               STX EOL           ;END OF LINE POINTER
0810 861D               STX SUBSTRT       ;SUBSTRING START POINTER
0812 861E               STX SUBEND        ;SUBSTRING END POINTER
0814 861F               STX MODE          ;MAINLINE/SUBSTRING MODE FLAG
0816             ;
0816             ;MAINLINE CHARACTER ENTRY ROUTINE
0816             ;
0816 200CFD      GETCHR JSR KEYIN         ;GET CHAR USING MONITOR ROUTINE
0819 C988        GETCH1 CMP #BS           ;BACKSPACE?
081B F05B               BEQ BKSPCE        ;YES, GOTO BACKSPACE ROUTINE
081D C99B               CMP #ESC          ;ESCAPE KEY?
081F F031               BEQ ESCAPE        ;YES, GOTO ESCAPE VECTOR ROUTINE
```

```
0821 C995          CMP #NAK       ;FORWARD ARROW?
0823 F061          BEQ FORWRD     ;YES, GOTO FORWARD ARROW ROUTINE
0825 C98D          CMP #CR        ;RETURN?
0827 F063          BEQ LINEND     ;YES, GOTO EXIT ROUTINE
0829 A619          LDX CHAR#      ;NONE OF THESE, GET CURRENT CHAR#
082B 297F          AND #FIX       ;FIX NEG ASCII INPUT FOR
                                   APPLESOFT
082D 204508        JSR STRPNT     ;STORE AND PRINT CHAR
0830        ;
0830        ;POINTER UPDATING
0830        ;
0830 E619   FXPTRS INC CHAR#      ;INC POSITION-IN-STRING POINTER
0832 A619          LDX CHAR#      ;GET IT
0834 E41E          CPX SUBEND     ;AT END OF SUBSTRING OR BUFFER?
0836 F076          BEQ WHICH      ;YES, GO FIND OUT WHICH
0838 A41A          LDY EOL        ;GET END OF LINE POINTER
083A C419          CPY CHAR#      ;END OF CURRENT LINE?
083C B004          BCS FXPOUT     ;NO, SKIP EOL POINTER UPDATE
083E E61A          INC EOL        ;INCREMENT END OF LINE POINTER
0840 F05F          BEQ BUFULL     ;256 CHARS! GOTO BUFFER FULL
0842 4C1608 FXPOUT JMP GETCHR     ;DONE. GET ANOTHER CHARACTER
0845        ;
0845        ;STORE AND PRINT ROUTINE
0845        ;
0845 9D0002 STRPNT STA BUFFER,X   ;STORE IN CURRENT BUFFER LOC.
0848 C920          CMP #CTL       ;CONTROL CHARACTER?
084A 9002          BCC PNT        ;NO, SKIP TO PRINT
084C 0980          ORA #INV       ;YES, CONVERT TO INVERSE
084E 20EDFD PNT    JSR PRINT      ;PRINT TO SCREEN
0851 60            RTS
0852        ;
0852        ;ESCAPE KEY VECTOR ROUTINE
0852        ;
0852 A41F   ESCAPE LDY MODE       ;SUBSTRING MODE?
0854 D048          BNE SBEXV      ;YES, GOTO SUBSTRING EXIT VECTOR
0856 200CFD        JSR KEYIN      ;GET ANOTHER CHARACTER
0859 C995          CMP #NAK       ;FORWARD ARROW?
085B F00F          BEQ INSV       ;YES, GOTO INSERT MODE VECTOR
085D C988          CMP #BS        ;BACKSPACE?
085F F011          BEQ DELV       ;YES, GOTO DELETE MODE VECTOR
0861 C9A0          CMP #BLANK     ;SPACE CHAR?
0863 F00A          BEQ ZMMV       ;YES, GOTO CURSOR ZOOM VECTOR
0865 C99D          CMP #CSM       ;CTRL-SHIFT-M?
0867 F00C          BEQ ZAPV       ;YES, GOTO LINE ZAP VECTOR
0869 4C7409        JMP CHRFND     ;NONE OF THESE, GOTO CHAR FIND
086C 4C0509 INSV   JMP INSERT     ;GOTO INSERT ROUTINE
086F 4C5509 ZMMV   JMP ZOOM       ;GOTO CURSOR ZOOM ROUTINE
0872 4CED08 DELV   JMP DELETE     ;GOTO DELETE ROUTINE
0875 4C9A09 ZAPV   JMP ZAP        ;GOTO DELETE-TO-EOL ROUTINE
0878        ;
0878        ;BACKSPACE ROUTINE
0878        ;
0878 A419   BKSPCE LDY CHAR#      ;GET POSITION IN LINE
087A C41D          CPY SUBSTRT    ;AT BEGINNING OF LINE/SUBSTRING?
087C F005          BEQ BSOUT      ;YES, RETURN
087E C619          DEC CHAR#      ;NO, DECREMENT POSITION IN LINE
0880 2010FC        JSR BACKSP     ;BACKSPACE CURSOR
0883 4C1608 BSOUT  JMP GETCHR     ;RETURN
0886        ;
0886        ;FORWARD ARROW ROUTINE
0886        ;
0886 20F4FB FORWRD JSR ADVANC     ;ADVANCE CURSOR
0889 4C3008        JMP FXPTRS     ;RETURN TO INCREMENT CHAR#
088C        ;
088C        ;EXIT ROUTINE
088C        ;
088C A41F   LINEND LDY MODE       ;SUBSTRING MODE?
088E D00E          BNE SBEXV      ;YES, GOTO SUBSTRING EXIT
0890 A619          LDX CHAR#      ;STORE CHARACTER COUNT
0892 861A          STX EOL        ;IN EOL POINTER
0894 9D0002        STA BUFFER,X   ;STORE CR AT END OF STRING
0897 2042FC        JSR CLREOP     ;CLEAR SCREEN TO END OF PAGE
089A 2062FC        JSR RETURN     ;PERFORM CARRIAGE RETURN
089D 60            RTS            ;EXIT TO CALLER
089E 4C3D09 SBEXV  JMP SUBEXT     ;GOTO SUBSTRING EXIT
```

```
08A1              ;
08A1              ;BUFFER FULL ROUTINE
08A1              ;
08A1 C61A         BUFULL DEC EOL         ;DECREMENT EOL POINTER
08A3 C619         BUFUL1 DEC CHAR#       ;DECREMENT CURSOR POSITION
08A5 2010FC              JSR BACKSP      ;BACKSPACE
08A8 203AFF       BELEX  JSR BELL        ;SOUND BELL
08AB 4C1608              JMP GETCHR      ;RETURN
08AE              ;
08AE              ;DETERMINE MAINLINE OR SUBSTRING MODE
08AE              ;
08AE A41F         WHICH  LDY MODE        ;SUBSTRING MODE?
08B0 F0F1                BEQ BUFUL1      ;NO, GOTO BUFFER END ROUTINE
08B2 4C1709              JMP MOVEFD      ;YES, MOVE RIGHT STRING FORWARD
08B5              ;
08B5              ;MOVE STRING BACK ROUTINE
08B5              ;
08B5 A619         MOVEBK LDX CHAR#       ;GET DESTINATION START
08B7 A41B                LDY STRT        ;GET STRING START
08B9 A51A                LDA EOL         ;GET STRING END
08BB 38                  SEC
08BC E51B                SBC STRT        ;SUBTRACT STRING START
08BE 18                  CLC
08BF 6519                ADC CHAR#       ;ADD PRESENT CURSOR POSITION
08C1 851C                STA TEMP        ;STORE NEW EOL POINTER
08C3 B90002       MVBLP  LDA BUFFER,Y    ;GET STRING CHARACTER
08C6 204508              JSR STRPNT      ;STORE AND PRINT CHARACTER
08C9 C8                  INY             ;INCREMENT THE
08CA E8                  INX             ;POSITION POINTERS
08CB C41A                CPY EOL         ;END OF STRING?
08CD 90F4                BCC MVBLP       ;NO, GET ANOTHER CHARACTER
08CF 2042FC              JSR CLREOP      ;YES, CLEAR TO END OF PAGE
08D2 8A                  TXA             ;STORE CURSOR POSITION
08D3 A8                  TAY             ;IN Y REGISTER
08D4 A9A0                LDA #BLANK      ;GET SPACE CHARACTER
08D6 9D0002       CLRLP  STA BUFFER,X    ;STORE IN BUFFER BEYOND NEW EOL
08D9 E8                  INX             ;INCREMENT POSITION
08DA E41A                CPX EOL         ;AT OLD END OF LINE?
08DC 90F8                BCC CLRLP       ;NO, DO IT AGAIN
08DE A61C                LDX TEMP        ;YES, GET NEW EOL
08E0 861A                STX EOL         ;STORE IT
08E2 98                  TYA             ;GET CURSOR POSITION
08E3 AA                  TAX             ;BACK INTO X REGISTER
08E4              ;
08E4              ;RESTORE CURSOR ROUTINE
08E4              ;
08E4 2010FC       RESTOR JSR BACKSP      ;BACKSPACE
08E7 CA                  DEX             ;DECREMENT CURSOR POSITION
08E8 E419                CPX CHAR#       ;AT PRESENT CHARACTER POSITION?
08EA D0F8                BNE RESTOR      ;NO, DO IT AGAIN
08EC 60                  RTS             ;YES, RETURN
08ED              ;
08ED              ;DELETE ROUTINE
08ED              ;
08ED A619         DELETE LDX CHAR#       ;GET PRESENT CHARACTER POSITION
08EF E8                  INX             ;INCREMENT TO NEXT CHARACTER
08F0 861B                STX STRT        ;STORE STRING START POSITION
08F2 A41A         DELELP LDY EOL         ;GET END OF LINE POINTER
08F4 C419                CPY CHAR#       ;SAME AS NEXT CHARACTER POSITION?
08F6 F00A                BEQ DELOUT      ;YES, NOTHING TO DELETE!
08F8 20B508              JSR MOVEBK      ;NO, MOVE STRING BACK ONE SPACE
08FB 200CFD              JSR KEYIN       ;GET ANOTHER CHARACTER
08FE C988                CMP #BS         ;ANOTHER BACKSPACE CHARACTER?
0900 F0F0                BEQ DELELP      ;YES, DELETE ANOTHER CHARACTER
0902 4C1908       DELOUT JMP GETCH1      ;NO, BACK TO MAINLINE
0905              ;
0905              ;INSERT ROUTINE INITIALIZE
0905              ;
0905 A61A         INSERT LDX EOL         ;GET END OF LINE POINTER
0907 E0FE                CPX #BEND       ;END OF ALLOWABLE INSERTIONS?
0909 B09D                BCS BELEX       ;YES, STOP INPUT
090B A619                LDX CHAR#       ;NO, GET POSITION IN LINE
090D E41A                CPX EOL         ;AT END OF LINE?
090F F029                BEQ INOUT       ;YES, NO NEED TO INSERT!
0911 861D                STX SUBSTR      ;NO, STORE SUBSTRING START
```

```
0913 861E           STX SUBEND       ;STORE PRESENT SUBSTRING END
0915 851F           STA MODE         ;SET SUBSTRING MODE FLAG
0917         ;
0917         ;MOVE STRING FORWARD ROUTINE
0917         ;
0917 20F4FB  MOVEFD JSR ADVANC       ;ADVANCE CURSOR
091A BD0002         LDA BUFFER,X     ;GET FIRST STRING CHARACTER
091D E61A           INC EOL          ;INCREMENT EOL POINTER
091F F02E           BEQ SBOUT        ;BUFFER END! STOP INPUT
0921 E8      MVFLP  INX              ;POINT TO SECOND CHARACTER
0922 BC0002         LDY BUFFER,X     ;GET SECOND CHARACTER
0925 204508         JSR STRPNT       ;STORE AND PRINT FIRST CHAR
0928 98             TYA              ;TRANFER SECOND CHAR TO ACC.
0929 E41A           CPX EOL          ;END OF LINE?
092B D0F4           BNE MVFLP        ;NO, DO IT AGAIN
092D E8             INX              ;YES
092E 20E408         JSR RESTOR       ;RESTORE CURSOR
0931 98             TYA              ;GET SPACE CHAR INTO ACC.
0932 204508         JSR STRPNT       ;STORE & PRINT AT INSERT POSITION
0935 2010FC         JSR BACKSP       ;RETURN CURSOR TO INSERT POSITION
0938 E61E           INC SUBEND       ;INCREMENT SUBSTRING END POINTER
093A 4C1608  INOUT  JMP GETCHR       ;GET ANOTHER CHAR
093D         ;
093D         ;SUBSTRING EXIT ROUTINE
093D         ;
093D A61E    SUBEXT LDX SUBEND       ;GET SUBSTRING END POSITION
093F 861B           STX STRT         ;STORE IN STRING START POINTER
0941 20B508         JSR MOVEBK       ;MOVE RIGHT STRING BACK
0944 A200           LDX #ZERO        ;RESET THE
0946 861D           STX SUBSTR       ;SUBSTRING START,
0948 861E           STX SUBEND       ;SUBSTRING END POINTERS
094A 861F           STX MODE         ;AND MODE FLAG
094C 4C1608         JMP GETCHR       ;BACK TO MAINLINE
094F 2010FC  SBOUT  JSR BACKSP       ;BACKSPACE
0952 4CA108         JMP BUFULL       ;GOTO BUFFER FULL
0955         ;
0955         ;CURSOR ZOOM ROUTINE
0955         ;
0955 A51A    ZOOM   LDA EOL          ;GET EOL POINTER
0957 F00E           BEQ ZMOUT        ;NULL LINE! RETURN
0959 AA             TAX              ;STORE EOL IN X REGISTER
095A E519           SBC CHAR#        ;CURSOR AT END OF LINE?
095C F00C           BEQ ZBEG         ;YES, ZOOM TO LINE START
095E 8619           STX CHAR#        ;STORE CURSOR POSITION (EOL)
0960 AA             TAX              ;GET ADVANCE COUNT IN X REGISTER
0961 20F4FB  ZOOMLP JSR ADVANC       ;ADVANCE CURSOR
0964 CA             DEX              ;DECREMENT ADVANCE COUNT
0965 D0FA           BNE ZOOMLP       ;ADVANCE AGAIN IF NOT AT EOL
0967 4C1608  ZMOUT  JMP GETCHR       ;BACK TO MAINLINE
096A 2010FC  ZBEG   JSR BACKSP       ;BACKSPACE
096D CA             DEX              ;DECREMENT POSITION IN LINE
096E D0FA           BNE ZBEG         ;DO IT AGAIN IF NOT AT LINE START
0970 8619           STX CHAR#        ;STORE CURSOR POSITION
0972 F0F3           BEQ ZMOUT        ;BACK TO MAINLINE
0974         ;
0974         ;CHARACTER SEARCH ROUTINE
0974         ;
0974 297F    CHRFND AND #FIX         ;CONVERT NEG ASCII INPUT
0976 851B           STA STRT         ;STORE KEY CHARACTER
0978 A619           LDX CHAR#        ;GET PRESENT CURSOR POSITION
097A E8      CHRFLP INX              ;INCREMENT CURSOR POINTER
097B 20F4FB         JSR ADVANC       ;ADVANCE CURSOR
097E E419    CHRF1  CPX CHAR#        ;AT OLD CURSOR POSITION?
0980 F00D           BEQ CHFOUT       ;YES, CHARACTER NOT FOUND
0982 E41A           CPX EOL          ;END OF LINE?
0984 B00C           BCS SBEG         ;YES, START AGAIN AT LINE START
0986 BD0002         LDA BUFFER,X     ;GET CHARACTER AT THIS POSITION
0989 C51B           CMP STRT         ;SAME AS KEY?
098B D0ED           BNE CHRFLP       ;NO, TRY AGAIN
098D 8619           STX CHAR#        ;YES, STORE CURSOR POSITION
098F 4C1608  CHFOUT JMP GETCHR       ;BACK TO MAINLINE
0992 2010FC  SBEG   JSR BACKSP       ;BACKSPACE
0995 CA             DEX              ;BEGINNING OF LINE?
0996 D0FA           BNE SBEG         ;NO, BACKSPACE AGAIN
0998 F0E4           BEQ CHRF1        ;YES, CONTINUE SEARCH
```

```
099A              ;
099A              ;ZAP (DELETE TO END OF LINE) ROUTINE
099A              ;
099A A619         ZAP     LDX CHAR#            ;GET CURSOR POSITION
099C A9A0                 LDA #BLANK           ;LOAD ACC. WITH SPACE CHAR
099E 204508       ZAPLP   JSR STRPNT           ;STORE AND PRINT IT
09A1 E8                   INX                  ;NEXT POSITION
09A2 E41A                 CPX EOL              ;END OF LINE?
09A4 90F8                 BCC ZAPLP            ;NO, DO IT AGAIN
09A6 20E408               JSR RESTOR           ;YES, RESTORE CURSOR
09A9 4C1608               JMP GETCHR           ;BACK TO MAINLINE
09AC              ;
09AC              ;DISK INPUT ROUTINE
09AC              ;
09AC A2FF         DISKIN  LDX #ZERO-$1         ;INITIATE THE
09AE E8           DISKL1  INX                  ;CHAR# POINTER
09AF 200CFD               JSR KEYIN            ;GET A CHARACTER
09B2 9D0002               STA BUFFER,X         ;STORE IN BUFFER
09B5 C98D                 CMP #CR              ;CARRIAGE RETURN?
09B7 D0F5                 BNE DISKL1           ;NO, GET ANOTHER CHARACTER
09B9 861A                 STX EOL              ;YES, STORE CHARACTER COUNT
09BB E8                   INX                  ;INIT FOR ASCII CONVERSION
09BC BDFF01       DISKL2  LDA BUFFER-$1,X      ;GET BUFFER CHARACTER
09BF 297F                 AND #FIX             ;CONVERT FOR APPLESOFT
09C1 9DFF01               STA BUFFER-$1,X      ;PUT IT BACK
09C4 CA                   DEX                  ;COUNT BACK TO ZERO
09C5 D0F5                 BNE DISKL2           ;LOOP IF NOT FINISHED
09C7 A61A                 LDX EOL              ;CHAR COUNT IN X REG.
09C9 60                   RTS                  ;EXIT TO CALLER
                          END
```

```
0800              ;**********************
0800              ;*                    *
0800              ;*   INTERFACE CODE   *
0800              ;*     FP - GETLN     *
0800              ;*                    *
0800              ;*         BY         *
0800              ;*                    *
0800              ;*    WES HUNTRESS    *
0800              ;*  SIERRA MADRE, CA  *
0800              ;*                    *
0800              ;* COPYRIGHT (C) 1982 *
0800              ;*   MICRO INK, INC.  *
0800              ;*CHELMSFORD, MA 01824*
0800              ;* ALL RIGHTS RESERVED*
0800              ;*                    *
0800              ;**********************
0800              ;
0800              ;EQUATES: CONSTANTS & ZERO PAGE
0800              ;
0800              CURS   EPZ $19
0800              ZERO   EPZ $00
0800              BLANK  EPZ $A0
0800              LENLOC EPZ $02
0800              STADRL EPZ $08
0800              STADRH EPZ $09
0800              STRLEN EPZ $1A
0800              VARPTR EPZ $69
0800              ;
0800              ;EQUATES: BUFFER & ADDRESSES
0800              ;
0800              BUFFER EQU $0200
0800              GETLN  EQU $0800
0800              EENTRY EQU $0810
0800              STRPNT EQU $0845
0800              DISKIN EQU $09AC
0800              BACKSP EQU $FC10
0800              RETURN EQU $FC62
0800              ;
0300                     ORG $0300
0300                     OBJ $0800
0300              ;
0300              ;PRINT X$ SUBROUTINE
0300              ;
0300 A002         PSCRN  LDY #LENLOC
0302 B169                LDA (VARPTR),Y      ;GET X$ STRING LENGTH
0304 851A                STA STRLEN          ;STORE STRING LENGTH PTR
0306 A900                LDA #$00
0308 C51A                CMP STRLEN          ;LEN=0 MEANS JUST A CARRIAGE RETURN

030A F019                BEQ PSCRNX          ;SKIP IF JUST A CARRIAGE RETURN
030C C8                  INY
030D B169                LDA (VARPTR),Y      ;GET X$ ADDR LOW BYTE
030F 8508                STA STADRL          ;STORE IN X$ ADDR PTR LOW
0311 C8                  INY
0312 B169                LDA (VARPTR),Y      ;GET X$ ADDR HI BYTE
0314 8509                STA STADRH          ;STORE IN X$ ADDR PTR HI
0316 A000                LDY #ZERO           ;INITIATE THE
0318 A200                LDX #ZERO           ;  COUNTERS
031A B108         PNTLP  LDA (STADRL),Y      ;GET MID$(X$,Y,1)
031C 204508              JSR STRPNT          ;STORE & PRINT
031F E8                  INX                 ;INCREMENT
0320 C8                  INY                 ;  COUNTERS
0321 C41A                CPY STRLEN          ;END OF STRING?
0323 90F5                BCC PNTLP           ;NO, GET ANOTHER CHAR
0325 60           PSCRNX RTS                 ;EXIT TO CALLER
0326              ;
0326              ;PRINT XS TO SCREEN
0326              ;
0326 200003       PRINT  JSR PSCRN           ;PRINT X$
0329 2062FC              JSR RETURN          ;DO A CARRIAGE RETURN
032C 60                  RTS                 ;EXIT TO CALLER
032D              ;
032D              ;EDIT X$
032D              ;
032D 200003       EDIT   JSR PSCRN           ;PRINT X$
```

```
0330 A9A0              LDA #BLANK          ;PUT SPACE CHAR
0332 9D0002     EDLP1  STA BUFFER,X        ;  INTO REMAINING
0335 E8                INX                 ;  BUFFER SPACE
0336 D0FA              BNE EDLP1
0338 2010FC     EDLP2  JSR BACKSP          ;RESTORE CURSOR
033B 88                DEY                 ;  TO LINE START
033C D0FA              BNE EDLP2
033E A200              LDX #ZERO           ;STORE CURSOR
0340 8619              STX CURS            ;  POSITION
0342 201008            JSR EENTRY          ;GETLN EDIT ENTRY
0345 4C4B03            JMP TOX$            ;PUT IN X$
0348            ;
0348            ;X$ KEYBOARD INPUT
0348            ;
0348 200008     KYBIN  JSR GETLN           ;GET A LINE
034B A002       TOX$   LDY #LENLOC         ;TRANSFER STRING
034D 8A                TXA                 ;  LENGTH FROM ACC.
034E 9169              STA (VARPTR),Y      ;  TO X$
0350 C8                INY
0351 A900              LDA #ZERO           ;STORE
0353 9169              STA (VARPTR),Y      ;  KEYBOARD
0355 C8                INY                 ;  BUFFER
0356 A902              LDA #LENLOC         ;  ADDRESS
0358 9169              STA (VARPTR),Y      ;  INTO X$
035A 60                RTS                 ;EXIT TO CALLER
035B            ;
035B            ;X$ DOS INPUT
035B            ;
035B 20AC09     DOSIN  JSR DISKIN          ;GETLN DOS INPUT ENTRY
035E 4C4B03            JMP TOX$            ;PUT INPUT IN X$
                       END
```

Trick DOS

by Sanford M. Mossberg

Here are a few techniques to help you get more power from Apple DOS.

On booting a disk, the DOS command table (DCT) comes to reside at RAM locations $A884-$A908 (decimal 43140-43272). The last letter of each of the 28 DOS commands is represented by a negative ASCII character which signals the end of the command. Other letters or numerals are written in positive ASCII code. A zero marks the end of the DCT. Armed with these simple facts, we can trick DOS 3.2 or 3.3 into obeying our whims and desires.

Listing 1 provides code for TRICK DOS. Following initialization (lines 2000-2060) and optional instructions (lines 2500-2670), a menu is presented (lines 600-710), each item of which is analyzed:

1. *Display Current DOS Command Table:* The heart of the entire program is found in the subroutine at lines 100-180. The starting location (START) of the table never changes. Lines 120-130 search successive memory locations in the DCT until a zero byte is found. The end address of the table, not including the zero byte, is assigned to the variable FIN. Line 140 initializes the array DOS$(*,*), the contents of which are noted in line 102. Lines 150-180 PEEK DCT locations, fill the two-dimensional matrix and create a string (DOS$) which contains every character in the DCT. Subsequently, the array variables will be used to format screen display (lines 860-880 and 1060-1070), and the string variable will be manipulated to alter the command table by POKEing data into RAM. The displayed DCT may be listed to a printer (see figure 1).

2. *Change DOS Command Table:* The program block starting at line 1000 first outputs current commands by utilizing the routine described earlier. The command to be changed (OC$) is requested in line 1080. Since keyboard input is in positive ASCII code, the high bit of the final letter is turned on (line 1090). The validity of the command is checked in line 1100 and variable PT marks the position of the command in the array. An invalid command triggers an error message

(line 1110) and returns the user to the prior input request. The replacement command (NC$) is solicited in line 1130 and negative ASCII conversion occurs in line 1140. The subroutine at lines 400-500 rearranges the DCT. Commands preceding and following the changed command are contained in T1$ and T3$, respectively; the new command is placed in T2$. In line 460, DOS$ is recreated by concatenation of the above-noted strings. Lines 470-500 POKE the new command table into memory. An incidental, but important, feature of this entire section, is the effective error trapping (lines 1080, 1110, 1120, 1130, 1170, 1180, 1210 and 1240) which prevents potential crashing of the program and assures professionally formatted screen display.

Figure 1: Current DOS Commands and Addresses

DEC	HEX		DEC	HEX	
43140	A884	INIT	43206	A8C6	APPEND
43144	A888	LOAD	43212	A8CC	RENAME
43148	A88C	SAVE	43218	A8D2	CATALOG
43152	A890	RUN	43225	A8D9	MON
43155	A893	CHAIN	43228	A8DC	NOMON
43160	A898	DELETE	43233	A8E1	PR#
43166	A89E	LOCK	43236	A8E4	IN#
43170	A8A2	UNLOCK	43239	A8E7	MAXFILES
43176	A8A8	CLOSE	43247	A8EF	FP
43181	A8AD	READ	43249	A8F1	INT
43185	A8B1	EXEC	43252	A8F4	BSAVE
43189	A8B5	WRITE	43257	A8F9	BLOAD
43194	A8BA	POSITION	43262	A8FE	BRUN
43202	A8C2	OPEN	43266	A902	VERIFY

3. *Restore Normal DOS Command Table* and

4. *Try these commands:* Data statements in lines 2100-2110 contain ASCII code for the normal DCT. Line 1330 reads the data into the variable NDOS$. A sample table which I have found useful is coded in lines 2120-2130. Line 1340 produces MYDOS$. Lines 1380-1390 replace the resident DCT with either of these strings, thus restructuring the entire command table rapidly.

5. *Exit Program:* At program termination all text and graphics modes should be normalized. Line 1510 accomplishes this by successively turning off hi-res, turning on text page one, clearing the keyboard strobe and setting a full text window. Although TRICK DOS does not require these steps, the habit is a good one to cultivate. After the program ends, the new command table will remain viable in RAM until rebooting occurs or power is discontinued. If you prefer, the new DCT can be preserved permanently by initializing a disk.

Knowing that DOS intercepts and reviews all commands before the Applesoft interpreter can process the command, several admonitions are appropriate. Each newly created DOS command should have a character set that does not duplicate the first letters of any Applesoft BASIC command. To better understand this pitfall, imagine that we have changed "LOAD" to "L" and "RENAME" to "RE". Now, if we type "LIST" or "LEFT$", DOS understands this to mean LOAD (L=LOAD) the file "IST" or "EFT$", and the "FILE NOT FOUND" error message is returned. Typing "REM" would produce the same error message as DOS attempted to RENAME (RE=RENAME) the nonexistent file "M." So far this is annoying but not harmful.

Consider the results from changing "INIT" to "I." Any Applesoft command beginning with an "I" would promptly start initializing the disk. This would be catastrophic and must be avoided! For the reasons cited above, I advise you to peruse a list of Applesoft BASIC commands before modifying a DOS command. Changing "LOAD" to "LD", "RENAME" to "RNM" and "INIT" to "I*" would have avoided the chaos. Choice #4 from the menu will create a table of "safe" commands that I have found to be functional.

When you begin using a newly created DCT, mistakes will be inevitable and error messages will proliferate. The DCT commands "LOAD" and "SAVE" are special in that they also exist as Applesoft commands to a cassette recorder. If either is used erroneously, the system will hang. Only by pressing "RESET" can you recover. If you do not have autostart ROM, altering these two commands may be more of a nuisance than an aid.

Experiment freely and enjoy your newfound power over DOS.

```
1  REM  ************************
2  REM  *                      *
3  REM  *       TRICK DOS      *
4  REM  *     SANDY MOSSBERG   *
5  REM  *                      *
6  REM  *   COPYRIGHT (C) 1982 *
7  REM  *     MICRO INK, INC.  *
8  REM  *  CHELMSFORD, MA 01824*
9  REM  *   ALL RIGHT RESERVED *
10  REM *                      *
11  REM ************************
20  TEXT : CALL  - 936: POKE  - 16298,0: POKE  - 16300,0: POKE  - 16368,0

30  GOSUB 2010: GOSUB 3010: GOSUB 2510: GOTO 610
100  REM

        PEEK COMMAND TABLE
         AND CREATE ARRAY

102  REM ARRAY DOS$(R1-28,C1-2)
        C1=COMMAND
        C2=START ADDR

104  REM DOS$=DOS COMMAND TABLE

106  REM DOS=ADDR COMMAND TABLE

110 TM = START
120  IF  PEEK (TM) = 0 THEN FIN = TM - 1: GOTO 140: REM  FIND END OF TABL
     E
130 TM = TM + 1: GOTO 120
140 I = 1: FOR J = 1 TO 29: FOR K = 1 TO 2:DOS$(J,K) = "": NEXT K,J:DOS$(
     1,2) =  STR$ (START):DOS$ = "": REM INITIALIZE
150  FOR DOS = START TO FIN
160  IF  ASC ( CHR$ ( PEEK (DOS))) > 127 THEN DOS$(I,1) = DOS$(I,1) +  CHR$
     ( PEEK (DOS)):DOS$ = DOS$ +  CHR$ ( PEEK (DOS)):DOS$((I + 1),2) =  STR$
     (DOS + 1):I = I + 1: GOTO 180: REM IF HI BYTE INCR I
170 DOS$(I,1) = DOS$(I,1) +  CHR$ ( PEEK (DOS)):DOS$ = DOS$ +  CHR$ ( PEEK
     (DOS))
180  NEXT DOS: RETURN
300  REM

        DEC --> HEX

310 HD% = DOS / 256:NBR = HD%: GOSUB 340:HB$ = HEX$
320 LD% =  FN MOD(DOS):NBR = LD%: GOSUB 340:LB$ = HEX$
330 HEX$ = HB$ + LB$: RETURN
340 H% = NBR / 16 + 1:L% = NBR / 16:L = L% * 16:L% = NBR - L + 1
350 HEX$ =  MID$ (H$,H%,1) +  MID$ (H$,L%,1): RETURN
400  REM

         REORGANIZE
        COMMAND TABLE

410  IF PT = 1 THEN T1$ = "": GOTO 430
420 T1$ =  LEFT$ (DOS$, VAL (DOS$(PT,2)) - START)
430  FOR I = 1 TO  LEN (NC$):T2$ = T2$ +  MID$ (NC$,I,1): NEXT
440  IF PT = 28 THEN T3$ = "": GOTO 460
450 T3$ =  RIGHT$ (DOS$,FIN + 1 -  VAL (DOS$((PT + 1),2)))
460 DOS$ = T1$ + T2$ + T3$:T2$ = ""
470 DOS = START
480  FOR I = 1 TO  LEN (DOS$): POKE DOS, ASC ( MID$ (DOS$,I,1)):DOS = DOS
      + 1: NEXT
490 FIN = FIN +  LEN (NC$) -  LEN (OC$)
500  POKE FIN + 1,0: RETURN
600  REM

         MENU

610  HOME :TT$ = "=============": GOSUB 3110
620 TT$ = "TRICK DOS MENU": GOSUB 3110
630 TT$ = "=============": GOSUB 3110
640  VTAB 6: PRINT "1.DISPLAY CURRENT DOS COMMAND TABLE.": PRINT
650  PRINT "2.CHANGE DOS COMMAND TABLE.": PRINT
660  PRINT "3.RESTORE NORMAL DOS COMMAND TABLE.": PRINT
```

```
670  PRINT "4.TRY SANDY'S COMMANDS.": PRINT
680  PRINT "5.EXIT PROGRAM.": PRINT : PRINT
690  VTAB 17: CALL  - 958: PRINT "   WHICH CHOICE? ";: GET I$: PRINT I$:
     CH =  VAL (I$)
700  IF CH < 1 OR CH > 5 OR I$ = "" THEN 690
710  ON CH GOTO 800,1000,1300,1300,1500
800  REM

        DISPLAY CURRENT TABLE

810  HOME :TT$ = "================================": GOSUB 3110
820 TT$ = "CURRENT DOS COMMANDS & ADDRESSES": GOSUB 3110
830 TT$ = "================================": GOSUB 3110
840  IF  NOT FF THEN  VTAB 8: INVERSE :TT$ = " READING DOS COMMAND TABLE
     ": GOSUB 3110: NORMAL
850  GOSUB 110: VTAB 4: CALL  - 958
860  PRINT : HTAB 2: INVERSE : PRINT "DEC";: HTAB 8: PRINT "HEX";: HTAB 2
     2: PRINT "DEC";: HTAB 28: PRINT "HEX": NORMAL : PRINT
870  FOR I = 1 TO 14
880  PRINT DOS$(I,2)" ";:DOS =  VAL (DOS$(I,2)): GOSUB 310: PRINT HEX$" "
     DOS$(I,1);: HTAB 21: PRINT DOS$((I + 14),2)" ";:DOS =  VAL (DOS$((I +
     14),2)): GOSUB 310: PRINT HEX$" "DOS$((I + 14),1): NEXT
890  IF FF THEN  FOR I = 1 TO 5: PRINT : NEXT : RETURN
900  VTAB 22: PRINT "LIST TABLE TO PRINTER (Y/N) ? ";: GET I$
910  IF I$ = "Y" THEN FF = 1: HTAB 1: CALL  - 998: CALL  - 958: PRINT B$:
      INVERSE : PRINT " TURN PRINTER ON AND PRESS ANY KEY ": PRINT : HTAB
     10: PRINT " EXPECT A PAUSE ";: GET I$: PRINT : NORMAL : PRINT D$;DOS
     $(20,1);1: GOSUB 810:FF = 0: PRINT D$;DOS$(20,1);0: GOTO 610
920  IF I$ = "N" THEN 610
930  HTAB 1: GOTO 900
1000  REM

         CHANGE TABLE

1010  HOME :TT$ = "===============": GOSUB 3110
1020 TT$ = "CHANGE COMMANDS": GOSUB 3110
1030 TT$ = "===============": GOSUB 3110
1040  VTAB 4: CALL  - 958: VTAB 8: INVERSE :TT$ = " READING DOS COMMAND T
     ABLE ": GOSUB 3110: NORMAL
1050  GOSUB 110: VTAB 5: CALL  - 958
1060  FOR I = 1 TO 7
1070  PRINT DOS$(I,1);: HTAB 10: PRINT DOS$((I + 7),1);: HTAB 20: PRINT D
     OS$((I + 14),1);: HTAB 30: PRINT DOS$((I + 21),1): NEXT
1080  VTAB 14: CALL  - 958: INPUT "TYPE COMMAND TO BE CHANGED: ";OC$: IF
     OC$ = "" THEN 1180
1090 OC$ =  MID$ (OC$,1, LEN (OC$) - 1) +  CHR$ ( ASC ( RIGHT$ (OC$,1)) +
     128): REM TURN HI BIT ON IN LAST LETTER OF    COMMAND
1100  FOR I = 1 TO 28: IF OC$ = DOS$(I,1) THEN PT = I: GOTO 1130: REM PT=
     POINTER TO      POSITION OF COMMAND IN ARRAY
1110  IF I = 28 THEN  PRINT B$: VTAB 16: INVERSE : PRINT " NOT A VALID CU
     RRENT COMMAND ": NORMAL : FOR J = 1 TO 3000: NEXT : GOTO 1080
1120  NEXT I
1130  VTAB 16: CALL  - 958: INPUT "TYPE NEW COMMAND: ";NC$: IF NC$ = "" THEN
     1130
1140 NC$ =  MID$ (NC$,1, LEN (NC$) - 1) +  CHR$ ( ASC ( RIGHT$ (NC$,1)) +
     128): REM TURN HI BIT ON IN LAST LETTER OF    COMMAND
1150  PRINT B$: VTAB 18: HTAB 3: PRINT "CONFIRM (Y/N) ? ";: GET I$: PRINT
     I$
1160  IF I$ = "Y" THEN  VTAB 20: INVERSE : PRINT " WRITING COMMAND TABLE
     ": GOSUB 410: VTAB 18: HTAB 1: CALL  - 958: PRINT " CHANGE COMPLETED
      ": NORMAL : GOTO 1220
1170  IF I$ <  > "N" THEN  VTAB 18: CALL  - 958: GOTO 1150
1180  VTAB 18: CALL  - 958: PRINT : PRINT "RETURN TO MENU OR TRY AGAIN (M
     /A) ? ";: GET I$: PRINT I$
1190  IF I$ = "A" THEN  GOTO 1080
1200  IF I$ = "M" THEN 610
1210  GOTO 1180
1220  VTAB 20: CALL  - 958: PRINT "ANOTHER CHANGE (Y/N) ? ";: GET I$: PRINT
     I$: IF I$ = "Y" THEN 1040
1230  IF I$ = "N" THEN 610
1240  GOTO 1220
1300  REM

         RESTORE NORMAL TABLE OR
          INSTALL SANDY'S TABLE
```

```
1310  VTAB 20: INVERSE : PRINT " WRITING COMMAND TABLE ";
1320 NDOS$ = "":MYDOS$ = ""
1330  FOR I = 1 TO 132: READ D:NDOS$ = NDOS$ +  CHR$ (D): NEXT
1340  FOR I = 1 TO 67: READ D:MYDOS$ = MYDOS$ +  CHR$ (D): NEXT : RESTORE

1350 DOS = START
1360  IF CH = 3 THEN TM$ = NDOS$:TT$ = " NORMAL DOS COMMAND TABLE REESTAB
     LISHED ":FIN = START +  LEN (NDOS$) - 1
1370  IF CH = 4 THEN TM$ = MYDOS$:TT$ = " SANDY'S COMMAND TABLE INSTALLED
     ":FIN = START +  LEN (MYDOS$) - 1
1380  FOR I = 1 TO  LEN (TM$): POKE DOS, ASC ( MID$ (TM$,I,1)):DOS = DOS +
     1: NEXT
1390  POKE FIN + 1,0
1400  HTAB 1: PRINT TT$: NORMAL : GOSUB 3210: HTAB 1: GOTO 690
1500  REM

         END PROGRAM

1510  POKE  - 16298,0: POKE  - 16300,0: POKE  - 16368,0: TEXT : HOME
1520  VTAB 10: INVERSE :TT$ = " END OF TRICK DOS PROGRAM ": GOSUB 3110: NORMA

1530  VTAB 15: PRINT " INITIALIZING A DISK BEFORE REBOOTING": PRINT "WILL
     PRESERVE THE CURRENT DOS COMMANDS"
1540  VTAB 22: END
2000  REM

         INITIALIZE

2010  DIM DOS$(30,2)
2020 D$ =  CHR$ (4):B$ =  CHR$ (7):SS$ = "                     ": REM 21
     SPACES
2030 H$ = "0123456789ABCDEF"
2040  DEF  FN MOD(X) = X -  INT (X / 256) * 256: REM SIMULATE MOD FUNCTIO
     N
2050 START = 43140: REM START OF TABLE
2060  RETURN
2100  DATA 73,78,73,212,76,79,65,196,83,65,86,197,82,85,206,67,72,65,73,2
     06,68,69,76,69,84,197,76,79,67,203,85,78,76,79,67,203,67,76,79,83,19
     7,82,69,65,196,69,88,69,195,87,82,73,84,197,80,79,83,73,84,73,79,206
     ,79,80,69,206,65,80,80,69,78,196
2110  DATA 82,69,78,65,77,197,67,65,84,65,76,79,199,77,79,206,78,79,77,79
     ,206,80,82,163,73,78,163,77,65,88,70,73,76,69,211,70,208,73,78,212,6
     6,83,65,86,197,66,76,79,65,196,66,82,85,206,86,69,82,73,70,217: REM
     NORMAL TABLE
2120  DATA 73,170,76,196,83,214,82,85,206,67,72,206,68,204,76,203,85,76,2
     03,67,211,82,196,69,88,195,87,210,80,83,206,79,208,65,208,82,69,206,
     67,65,212,77,206,78,77,206,80,163,73,163,77,65,216,70,208,73,78,212,
     66,211,66,204,66,210,86,69,210
2130  DATA 77,206,78,77,206,80,163,73,163,77,65,216,70,208,73,78,212,66,2
     11,66,204,66,210,86,69,210: REM             SANDY'S TABLE
2500  REM

         INSTRUCTIONS

2510  HOME :TT$ = "============": GOSUB 3110
2520 TT$ = "INSTRUCTIONS": GOSUB 3110
2530 TT$ = "============": GOSUB 3110
2540  VTAB 7: CALL  - 958: PRINT "DO YOU WANT INSTRUCTIONS (Y/N) ? ";: GET
     I$: PRINT I$: IF I$ = "N" THEN  RETURN
2550  IF I$ <  > "Y" THEN 2540
2560  POKE 34,4: VTAB 5: CALL  - 958
2570  PRINT "1.THE DOS COMMAND TABLE RESIDES AT RAM": PRINT "  LOCATIONS
     $A884 TO $A908 (DEC 43140": PRINT "  TO 43272).": PRINT
2580  PRINT "2.EACH COMMAND IS REPRESENTED BY ASCII": PRINT "  CHARACTER
     CODES. ONLY THE LAST LETTER": PRINT "  OF A COMMAND HAS THE HIGH BIT
      ON SO": PRINT "  THAT DOS CAN RECOGNIZE THE END OF THE"
2590  PRINT "  COMMAND. NOTE THE EXAMPLES BELOW:": PRINT : PRINT "       L
     OAD = 4C 4F 41 C4": PRINT "       INIT = 49 4E 49 D4": PRINT "       R
     UN  = 52 55 CE": PRINT : PRINT
2600  PRINT "3.ZERO MARKS THE END OF THE TABLE."
2610  GOSUB 3210: HOME
2620  PRINT "4.THIS PROGRAM WILL ENABLE YOU TO ALTER": PRINT "  THE COMMA
     ND TABLE. YOU MAY DESIRE TO": PRINT "  CHANGE 'CATALOG' TO ";: INVERSE
     : PRINT "CAT";: NORMAL : PRINT " OR 'SAVE' TO ": PRINT "  ";: INVERSE
     : PRINT "SV";: NORMAL
```

```
2630  PRINT ". BE SURE THAT YOUR NEW DOS COMMAND": PRINT "  DOES NOT DUPL
     ICATE THE FIRST PART OF": PRINT "  AN APPLESOFT BASIC COMMAND, OTHER
     WISE": PRINT "  UNUSUAL EVENTS MAY OCCUR. EXPERIMENT!"
2640  PRINT "  TIREDNESS OR SILLINESS MAY RESULT IN": PRINT "  WEIRD SYMB
     OLS!!!": PRINT
2650  PRINT "5.THESE MODIFICATIONS WILL TRIGGER A": PRINT "  SYNTAX ERROR
      IF A DIRECT OR DEFERRED": PRINT "  COMMAND UTILIZES 'NORMAL' TERMIN
     OLOGY."
2660  PRINT "6.";: INVERSE : PRINT "TRICK DOS";: NORMAL : PRINT " IS MENU
      -DRIVEN AND SELF-": PRINT "  PROMPTING. HAVE FUN!!!"
2670  POKE 34,0: GOSUB 3210: RETURN
3000  REM

        TITLE PAGE

3005  REM SF APPLE CORE FORMAT

3010  INVERSE : VTAB 4
3020 TT$ = SS$: GOSUB 3110: GOSUB 3110
3030 TT$ = "        TRICK DOS       ": GOSUB 3110
3040 TT$ = SS$: GOSUB 3110: GOSUB 3110
3050 TT$ = "  BY SANDY MOSSBERG  ": GOSUB 3110
3060 TT$ = SS$: GOSUB 3110: GOSUB 3110: NORMAL
3070  VTAB 16:TT$ = "CUSTOMIZE YOUR SET OF DOS COMMANDS!": GOSUB 3110
3080  GOSUB 3210: RETURN
3100  REM

        PRINT CENTER

3110 WIDTH = 20 - ( LEN (TT$) / 2): IF WIDTH <  = 0 THEN  PRINT TT$: RETURN

3120  HTAB WIDTH: PRINT TT$: RETURN
3200  REM

        CONTINUE/END

3210  VTAB 23: HTAB 12: PRINT "[ESC] TO END"
3220  VTAB 24: PRINT  TAB( 8);"[SPACE] TO CONTINUE ";
3230  PRINT "[ ]";: HTAB 29: GET ZZ$: IF ZZ$ =  CHR$ (27) OR ZZ$ =  CHR$
     (3) THEN  TEXT : HOME : GOTO 1510
3240  IF ZZ$ =  CHR$ (32) THEN  RETURN
3250  CALL  - 868: CALL  - 1008: GOTO 3230: REM
```

LACRAB

by N.R. McBurney

This utility produces a logically formatted and aesthetically pleasing listing of Applesoft programs, as well as a cross-reference table of their variables. These two functions not only yield a more professional looking documentation, but also make the task of program debugging and maintenance significantly easier.

Introduction

The following is an example of the screen output produced by the LIST command:

```
2400  IF BYTE = C1 THEN RE = 1:KM = KM + 5: REM COMMENT
2410  FOR I = 1 TO 255:BYTE =  PEEK (LOC):LOC = LOC + 1: IF BYTE = 0 THEN
    RE = 0:LOC = LOC + 2: GOTO 2340
2420  IF RE THEN KM = KM + 1
2430  NEXT
```

It isn't very easy to read. In fact, it is rather confusing. Take a second to examine the program listing at the end of this article, specifically at the listing for lines 2400-2430. I hope that you'll agree that the format of this second listing is considerably easier to read and more informative than the above example.

LACRAB stands for List And Cross Reference Applesoft BASIC. It has capabilities that make program debugging and documentation significantly easier. First, LACRAB prints only one statement per line and indents lines to suggest subordinate relationships. This feature alone greatly improves program readability. Second, LACRAB puts REM statements in boxes so that they stand out clearly. In-line REM statements (i.e., REM statements tacked on to another statement with a colon) are tabbed out to separate them from executable code and make them easy to see. Third, user-provided titling is accommodated along with automatic pagination for professional-quality documentation. Fourth, LACRAB generates a cross-reference table that identifies each line in which a variable appears. That table also flags undefined variables, equivalent variables, and variables that appear on only one line. Finally, the program length, in bytes, is printed out along with an approximation of the amount of RAM occupied by REM statements.

To be able to perform the above tasks on a program in RAM, we need to know how the program is represented in RAM and where it begins and ends. A BASIC statement in RAM starts with two bytes that point to the next BASIC statement. This is followd by two bytes containing the line number in numeric integer format, followed by the BASIC statement proper. Finally, a zero byte indicates the end of the BASIC statement.

Within the BASIC statement, bytes with values less than 128 represent ASCII characters. Bytes with values greater than 127 represent tokens that the Applesoft interpreter has substituted for BASIC keywords (e.g., 186 for PRINT). These token values are described in Appendix F of the Applesoft BASIC manual. Appendix L of that same manual tells us that the address of the start of the program is contained in decimal locations 103-104 and the end of the program in locations 175-176. Armed with this knowledge, one can write a program that examines the necessary memory locations byte by byte, builds up each line as a string, and outputs it to a printer. LACRAB is an elaboration of this basic scheme.

Program Operation

To run LACRAB, simply load the program to be listed and type EXEC LIST. The screen will clear and request heading information as below:

```
PROGRAM NAME?LISTER TEST CASE #1
DATE/TIME?AUGUST 11, 1980 8:50 PM
```

Once that information is provided the menu shown below will appear:

Figure 1: LACRAB Menu

```
                SYSTEM MENU
                    FOR
                PROGRAM TO:

   LIST AND CROSS REFERENCE APPLESOFT BASIC

 1) LISTING ONLY

 2) CROSS REFERENCE ONLY

 3) LISTING AND XREF

  WHICH OPTION?
```

After you've selected one of the above print options, your program will be listed. LACRAB assumes that the printer interface board is in slot one. If a cross-reference was requested, a display similar to the one below will appear when the listing is complete:

Figure 2: LACRAB Cross-Reference Monitor Display

```
                    LACRAB
     SYMBOL TABLE GENERATION MONITOR

LINE        CURRENT          OPERATING
NUMBER      SYMBOL           STATISTICS

1170        BLK
1170        LOC              CURR. LINE          1230
1170        LOC              LINES PROC.           23
1170        LOC              PROG. BYTES        10522
1180        LNE$             CURRENT BYTE         560
1180        LNE              % COMPLETE            5%
1180        B5               SYM TABLE LEN         11
1190        BYTE             LAST         SYMBOL:
1190        LOC                                    C2
1190        LOC
1190        LOC
1210        BYTE
1210        C1
1220        COMMENTS
1230        BYTE
1230        C2
```

Frankly, there isn't any logical requirement for the above display. I provide it because the cross-reference portion of LACRAB can be time-consuming (approximately 12 minutes to cross-reference LACRAB) and it frustrates me to stare at a blank screen. Once the cross-reference is complete and has printed out, LACRAB terminates with the following display:

```
37 LINES PRINTED.
LISTING COMPLETE....

]
```

At that point the program you just listed will be available to you.

How it Works

The first executable statement in LACRAB (line #3440) transfers control to the initialization routine (lines 3440-3840). This routine and the menu display section (lines 3850-4260) are located at the end of the program to make the remainder of the program run faster. (As a general rule, infrequently executed code should always be placed at the end of a program.)

The variables CO$, LINE$ and DF% (dimensioned in line 3450) are used during the cross-reference to store variable names (CO$), line numbers where the variable is referenced (LINE$), and a flag to indicate that a variable has been defined (DF%). Each of these variables is dimensioned to 200 and hence limits the number of variables that LACRAB can cross-reference to 200 — a limit that I've yet to approach.

The variable CO$ does double duty. In addition to holding variable names, it is used as temporary storage for consecutive REM statements while LACRAB is listing. Again, this limits LACRAB's capacity to 200 consecutive REM statements per program. I don't believe I've ever seen a BASIC program with 200 consecutive REM statements and don't believe this imposes much of a limitation.

In line 3620, the page width is assigned the value of 76 print positions. This value can be changed to adapt LACRAB to your particular printer configuration. The variable S6, defined in line 3790, sets the page length at 66 lines. The function PAGE, defined in line 3800, is simply a modulo function used in the output section to determine when to print page headings. The variable KOMMENT, set in line 3810, establishes the print position for 'in-line' REM statements. Again, at least in theory, you should be able to set this to any value compatible with your printer's capabilities and your own sense of esthetics.

After initialization and selection of output options, control is transferred to the program listing section (lines 1120-2040). Line numbers 1170 and 1180 pick up the line number of the next statement to be listed. At line 1190 LACRAB starts examining the program statement byte by byte. Lines 1200-1270 check for tokens that will require special formating: REM (C1), colon (C2), THEN (C3), FOR (C4), and NEXT (C5). If the byte has none of these values, it is translated either into a character or a BASIC keyword and appended to the next line to be printed. This process occurs in lines 1290-1360.

If the byte is a REM token (i.e., byte = 178) that immediately follows a line number, control is transferred to lines 1370-1420. Here the REM statement is decoded and stored in the CO$ array. The variable COMMENTS, used to keep track of how many consecutive REM statements have been processed, is incremented by one. LACRAB will continue to 'save' REM statements until the first non-REM statement is detected (line #1220). When that occurs, control is transferred to the routine in lines 1850-2040 where the comments are boxed and then output. Note that when LACRAB outputs REM statements the REM keyword is not printed. In the author's opinion, the output format of LACRAB makes it perfectly obvious which statements are and are not comment statements.

I have elected to take a contrary approach with implied GOTO statements. When LACRAB encounters a BASIC statement of the form

```
IF condition THEN line number
```

it prints out:

```
IF condition THEN GO TO line number
```

Note the space between GO and TO. LACRAB prints 'GO TO' instead of 'GOTO' to indicate that the 'GOTO' does not actually exist in the statement.

THEN tokens are processed in lines 1530-1580 and colons (:) are processed in lines 1450-1520. If the next byte following the colon is a REM token, lines 1460-1500 tabs the REM statement out to the print position specified by KOMMENT (currently 41). Since there may be some confusion if the REM keyword is omitted from in-line REM statements, LACRAB replaces the REM with a '!'.

The GOSUB 3240's sprinkled throughout the listing section of LACRAB transfer control to the line output routine (lines 3240-3310). If you make any changes to LACRAB (perhaps you're as opinionated as the author as to what constitutes esthetically pleasing program listings!), you should be careful to use this routine for output. The routine handles pagination, page numbering, and the 'folding' of lines where appropriate. It is this section of LACRAB that you would want to modify to make use, for example, of a printer's form feed feature or perhaps print out titles in an expanded print font. All LACRAB printer output should be handled by this routine.

The cross-reference portion of LACRAB begins at line 2060. Lines 2060-2260 display the headings for the screen display shown in figure 2.

Lines 2270-2311 involve a bit of trickery. What this code does, in effect, is to delete lines 1000-2310. The listing portion of LACRAB is no longer needed once the cross-reference is started. This results in faster execution of LACRAB's cross-reference procedure. This piece of bit-shuffling wizardry is accomplished by finding the address of where we currently are in the program (line #2290), skipping two lines (line #2310), and then resetting the start of program pointer to this new address (line #2311).

The main cross-reference loop begins at line 2340. At line 2350 the line number (LNE) is decoded. The rest of line 2350 and line 2360 update the cross-reference display shown in figure 2.

Lines 2370 through 2780 are a routine that decodes each variable as it is encountered. As each variable is decoded, that symbol and its associated line number are displayed at the bottom of the left-hand side of the display. At line 2710, the line number where the symbol is referenced is stored in the corresponding string array LINES$. This is accomplished by appending the line number (stored as two bytes in the string). The line number of the reference to the first variable (CO$(1)) is stored in character positions 1 and 2 of LINE$(1) in integer word format. The line number of the second reference to the same variable is stored in character positions 3 and 4, and so on.

There are several ways I could have handled the storage of line references. One can dimension matrices to handle the maximum number of references anticipated; one can write his own dynamic memory scheme; or one can take the easy way out and use strings, letting Apple worry about memory management. I opted for the latter solution.

Since a string can be, at most, 255 characters in length, no more than 127 references to a single variable are possible. More references will generate an error message at line 2750. In practice, I have never found this limit restrictive.

Once all of the program variables and their references have been decoded and stored in memory, they are sorted (in lines 2790-2830). When the sort begins, the flashing message "SORTING" is displayed on the screen. During the sort, every time an interchange occurs (line #2830), the Apple's speaker clicks. As before, I just like to be assured that something is occurring.

After the sort is complete, LACRAB starts printing the cross-reference table (lines 2880-3150). As it prints out each variable and its associated line references, it may prepend one of three symbols to the variable. If during the building of the cross-reference table LACRAB cannot find a variable definition, that variable is prepended with " —>>" during printout. If the variable only occurs in one line, it is prepended with an asterisk (*) at line 2920. While this may not always indicate a problem, it generally points to a misspelled variable name. Finally, if a variable is equivalent to a previous variable, "**" is prepended to the variable name. Because Applesoft BASIC only recognizes the first two characters of a variable, SIGMA and SIGN would be flagged as equivalent by LACRAB.

At the end of the cross-reference, an explanation of the symbols described above is printed (lines 3160-3200) and lines 3210 and 3240 print out the program length and the amount of RAM taken up by REM statements. LACRAB's last activity is to reset the end-of-program and start-of-program pointers (lines 3380-3400) and return control to the user.

Bugs — Real and Imagined

I know of two bugs in LACRAB. First, if one uses numbers in exponential format (e.g., I = 1.0E16), LACRAB will pick up the exponential portion as a variable during the cross-reference. 'E16' in the previous example would be identified as a variable. The second bug occurs when a statement is attached to a 'DATA' statement with a colon (e.g., 10 DATA 25:I = 10). During the cross-reference, LACRAB simply skips to the end of the line when it detects either a 'DATA' or a 'REM' statement. Hence, in the above example, LACRAB would be unaware of the reference to 'I' in statement number 10. Since I never combine 'DATA' with other type of statements, and rarely use exponential notation, I've never incorporated the necessary code to resolve those deficiencies.

Conclusion

LACRAB was written on an Apple II Plus (floating point BASIC-in-ROM) with 48K RAM. With two minor changes LACRAB should work with RAM Applesoft BASIC. The first location to be examined by LACRAB should be changed in line 3760 from 2051 to 12291 (i.e., 3760 LOC=12291). Line 3400 POKEs the hex value $801 into locations 103-104. The value needs to be changed to $3001 (i.e., 3400 POKE 103,1:POKE 104,48). Since I don't have RAM BASIC I've not tested these changes.

LACRAB takes up approximately 10.2K of RAM. Running it through a good optimizer such as Sensible Software's AOPT program will reduce that by about 35% to 6.6K, although it will not appreciably speed up processing.

```
                                                              LACRAB
                                                            04/05/82
                                                            PAGE - 1
===========================================================================

                                  +---------------------------+
 1000                             !  ************************ !
 1010                             !  *                      * !
 1015                             !  *       APPLESOFT      * !
 1020                             !  * BASIC PROGRAM LISTER * !
 1030                             !  *    N. R. MCBURNEY    * !
 1040                             !  *                      * !
 1050                             !  *  COPYRIGHT (C) 1982  * !
 1060                             !  *    MICRO INK, INC.   * !
 1070                             !  * CHELMSFORD, MA 01824 * !
 1080                             !  *  ALL RIGHTS RESERVED * !
 1090                             !  *                      * !
 1095                             !  ************************ !
                                  +---------------------------+
 1110 GOSUB 3440:                               ! CALL INITIALIZATION ROUTINE
                                        +--------------+
 1120                                   ! MAIN PROGRAM !
                                        +--------------+
 1140 IF NOT LST THEN
         GO TO 3320
 1150 PR# 1
 1160 IF LOC >  = EOP THEN
         GO TO 3320
 1170 LNE = PEEK(LOC) + BLK * PEEK(LOC + 1):
      LOC = LOC + 2
 1180 LNE$ = RIGHT$("     " + STR$(LNE),B5):! CONVERT LINE NUMBER TO STRING
 1190 BYTE = PEEK(LOC):
      LOC = LOC + 1
                                    +----------------------+
 1200                               ! CHECK FOR KEY TOKENS !
                                    +----------------------+
 1210 IF BYTE = C1 THEN
         GO TO 1380
 1220 IF COMMENTS THEN
         GO TO 1850
 1230 IF BYTE = C2 THEN
         GO TO 1450
 1240 IF BYTE = C3 THEN
         GO TO 1540
 1250 IF BYTE = C4 THEN
         GO TO 1600
 1260 IF BYTE = C5 THEN
         GO TO 1750
 1270 IF BYTE < C6 THEN
         GO TO 1310
                                      +-------------------+
 1280                                 ! BUILD UP THE LINE !
                                      +-------------------+
 1290 TXT$ = TXT$ + TKN$(BYTE - A8):
      IF BYTE < 210 THEN
         TXT$ = TXT$ + " "
 1300 GOTO 1190
 1310 IF BYTE = 0 THEN
         GOSUB 3240:
         QUOTE = 0:

                                                              LACRAB
                                                            04/05/82
                                                            PAGE - 2
===========================================================================

         LOC = LOC + 2:
         GOTO 1160
 1320 TXT$ = TXT$ + CHR$(BYTE)
 1330 IF BYTE <  > 34 THEN
         GO TO 1190
 1340 IF QUOTE = 0 THEN
         QUOTE = 1:
         GOTO 1190
 1350 QUOTE = 0
```

```
1360 GOTO 1190
                                        +------------------+
1370                                    ! PROCESS COMMENTS !
                                        +------------------+
1380 COMMENTS = COMMENTS + 1:
     CO$(COMMENTS) = LNE$ + "! "
1400 BYTE = PEEK(LOC):
     LOC = LOC + 1:
     IF BYTE = 0 THEN
        LOC = LOC + 2:
        GOTO 1160
1420 CO$(COMMENTS) = CO$(COMMENTS) + CHR$(BYTE):
     GOTO 1400
                                          +---------------+
1440                                      ! PROCESS COLON !
                                          +---------------+
1450 TXT$ = TXT$ + ":"
1460 IF PEEK(LOC) <  > 178 THEN
        GO TO 1510:                          ! CHECK FOR 'REM'
1470 J = LEN(TXT$) + SPACE + B5:
     RM = 1:
     IF J > KOMMENT THEN
        GO TO 1500
1480 FOR I = J TO KOMMENT
1490    TXT$ = TXT$ + " ":
     NEXT :
     QUOTE = 0
1500 TXT$ = TXT$ + "! ":
     LOC = LOC + 1:
     GOTO 1190
1510 IF NOT QUOTE THEN
        GOSUB 3240
1520 GOTO 1190
                                         +----------------+
1530                                     ! PROCESS 'THEN' !
                                         +----------------+
1540 TXT$ = TXT$ + " THEN":
     THN = THN + 3:
     GOSUB 3240:
     SPACE = SPACE + 3:
     IF PEEK(LOC + 1) < A3 OR PEEK(LOC) > A4 THEN
        GO TO 1190
1570 TXT$ = "GO TO ":                        ! ADD IMPLIED 'GO TO'
1580 GOTO 1190
                                          +---------------+
1590                                      ! PROCESS 'FOR' !

                                                                       LACRAB
                                                                     04/05/82
                                                                     PAGE - 3
===========================================================================

                                          +---------------+
1600 TXT$ = TXT$ + "FOR "
1610 BYTE = PEEK(LOC):
     LOC = LOC + 1
1620 IF BYTE = 0 THEN
        GOSUB 3240:
        LOC = LOC + 2:
        GOTO 1710
1630 IF BYTE <  > C2 THEN
        GO TO 1660
1640 IF PEEK(LOC) = C1 THEN
        NFR = 1:
        GOTO 1230
1650 TXT$ = TXT$ + ":":
     GOSUB 3240:
     GOTO 1710
1660 IF BYTE < A8 THEN
        GO TO 1690
1670 TXT$ = TXT$ + TKN$(BYTE - A8):
     IF BYTE < 210 THEN
        TXT$ = TXT$ + " "
1680 GOTO 1610
1690 TXT$ = TXT$ + CHR$(BYTE)
```

```
1700 GOTO 1610
1710 SPACE = SPACE + 3
1720 IF BYTE = 0 THEN
        GO TO 1160
1730 GOTO 1190
                                    +----------------+
1740                                ! PROCESS 'NEXT' !
                                    +----------------+
1750 SPACE = SPACE - 3:
     TXT$ = TXT$ + "NEXT "
1770 BYTE = PEEK(LOC):
     LOC = LOC + 1:
     IF BYTE = 0 THEN
        GOSUB 3240:
        LOC = LOC + 2:
        GOTO 1160
1790 IF BYTE = C2 THEN
        TXT$ = TXT$ + ":":
        GOSUB 3240:
        GOTO 1190
1800 IF BYTE = A7 THEN
        TXT$ = TXT$ + ",":
        SPACE = SPACE - 3:
        GOTO 1770
1810 IF BYTE > A8 THEN
        TXT$ = TXT$ + " " + TKN$(BYTE - A8):
        GOTO 1770
1820 TXT$ = TXT$ + CHR$(BYTE):
     GOTO 1770
                                   +------------------+
1840                               ! PROCESS COMMENTS !
                                   +------------------+

                                                                   LACRAB
                                                                 04/05/82
                                                                 PAGE - 4
=========================================================================

1850 SSPACE = SPACE:                       ! SAVE CURRENT SPACING
1860 LN = 0:
     SVE$ = LNE$
1870 FOR I = 1 TO COMMENTS:                ! FIND LENGTH OF LONGEST COMMENT
1880    IF LEN(CO$(I)) > LN THEN
           LN = LEN(CO$(I))
1890 NEXT
1900 SPACE = (WIDTH - LN) / 2:
     IF SPACE < 1 THEN
        SPACE = 1:                         ! CENTER COMMENTS
1910 GOSUB 2000:                           ! BOX THIS SET OF COMMENTS
1920 FOR I = 1 TO COMMENTS
1930    LNE$ = LEFT$(CO$(I),B5):
        TXT$ = MID$(CO$(I),6)
1940    FOR J = LEN(CO$(I)) TO LN
1950       TXT$ = TXT$ + " ":
        NEXT :
        TXT$ = TXT$ + "!"
1960    GOSUB 3240:
     NEXT
1970 GOSUB 2000
1980 COMMENTS = 0:
     LNE$ = SVE$:
     SPACE = SSPACE:
     IF NOT CX THEN
        GO TO 1230
1990 GOTO 3340
2000 LNE$ = "     "
2010 TXT$ = "+"
2020 FOR I = 1 TO LN - B5:
        TXT$ = TXT$ + "-":
     NEXT
2030 TXT$ = TXT$ + "+":
     GOSUB 3240
2040 RETURN
                                +-------------------------+
2050                            ! GENERATE STATUS DISPLAY !
                                +-------------------------+
```

```
2060 XREF = 0:
     PRINT CHR$(4)
2070 HOME :                              ! BEGIN STATUS DISPLAY
2080 PRINT TAB( 14);"*** LACRAB ***"
2090 PRINT TAB( 5);"SYMBOL TABLE GENERATION MONITOR"
2100 A$ = "---------------------------------------":
     PRINT A$
2110 PRINT "LINE";TAB( 9);"CURRENT";TAB( 23);"OPERATING"
2120 PRINT "NUMBER  SYMBOL";TAB( 23);"STATISTICS"
2130 PRINT LEFT$(A$,7);" ";LEFT$(A$,13);" ";LEFT$(A$,18)
2140 POKE 32,22:
     HTAB 23
2150 PRINT "CURR. LINE"
2160 PRINT "LINES PROC."
2170 L = EOP - 2049:
     SIZE = L:                           ! SAVE PROGRAM SIZE
2180 PRINT "PROG. BYTES ";RIGHT$("     " + STR$(L),5);
```



```
==============================================================================

2190 L = L * 0.0099:
     POKE 32,0:
     VTAB 12:
     HTAB 23
2200 PRINT "CURRENT BYTE"
2210 POKE 32,22:
     HTAB 23
2220 PRINT "% COMPLETE"
2230 PRINT "SYM TABLE LEN"
2240 PRINT "LAST SYMBOL:"
2250 POKE 34,7:
     POKE 32,0
2260 LOC = 8 * BLK + 3:
     GOTO 2290
                    +-------------------------------------------+
2270                ! SET START OF PROGRAM ADDRESS TO START     !
2280                ! OF CROSS REFERENCE (IE. SPEED UP PROGRAM) !
                    +-------------------------------------------+
2290 START = PEEK(121) + BLK * PEEK(122):
     FOR I = 1 TO 3
2310    IF PEEK(START) <  > 0 THEN
           START = START + 1:
           GOTO 2310
2311    START = START + 1:
     NEXT :
     POKE 103,PEEK(START):
     POKE 104,PEEK(START + 1)
                                 +-----------------+
2330                             ! CROSS REFERENCE !
                                 +-----------------+
2340 IF LOC >  = EOP THEN
        GO TO 2790
2350 LNE = PEEK(LOC) + PEEK(LOC + 1) * BLK:
     SYMBOL = 0:
     NN = 0:
     RD = 0:
     VTAB 9:
     HTAB 35:
     PRINT RIGHT$("     " + STR$(LNE),B5);:
     KK = KK + 1:
     VTAB 10:
     HTAB 35:
     PRINT RIGHT$("     " + STR$(KK),B5);
2360 VTAB 12:
     HTAB 35:
     PRINT RIGHT$("     " + STR$(LOC - B6),B5);:
     VTAB 13:
     HTAB 35:
     PRINT RIGHT$("     " + STR$(INT((LOC - B6) / L)) + "%",B5);:
     LOC = LOC + 2
2370 BYTE = PEEK(LOC):
     LOC = LOC + 1
```

```
2380 IF BYTE = B1 OR BYTE = B2 OR BYTE = B3 THEN
       RD = 1:                         ! READ, GET OR INPUT
2390 IF BYTE <  > B4 AND BYTE <  > C1 THEN

                                                                   LACRAB
                                                                 04/05/82
                                                                 PAGE - 6
==========================================================================

       GO TO 2450:                     ! CHECK FOR 'DATA' & 'REM' TOKENS
2400 IF BYTE = C1 THEN
       RE = 1:
       KM = KM + 5:                    ! COMMENT
2410 FOR I = 1 TO 255:
       BYTE = PEEK(LOC):
       LOC = LOC + 1:
       IF BYTE = 0 THEN
          RE = 0:
          LOC = LOC + 2:
          GOTO 2340
2420   IF RE THEN
          KM = KM + 1
2430 NEXT
2450 IF BYTE <  > C9 THEN
       GO TO 2500:                     ! CHECK FOR QOUTED LITERAL
2460 FOR I = 1 TO BLK:
       BYTE = PEEK(LOC):
       LOC = LOC + 1
2470   IF BYTE = C9 THEN
          GO TO 2550
2480   IF BYTE = 0 THEN
          GO TO 2500
2490 NEXT
2500 IF BYTE = 0 AND NOT WRD THEN
       LOC = LOC + 2:
       GOTO 2340
2510 IF (BYTE < A1 OR BYTE > A2) AND NOT (WRD AND BYTE >  = A3 AND BYTE <
     = A4) THEN
       GO TO 2530:                     ! NON-VARIABLE CHARACTER
2520 SYMBOL$ = SYMBOL$ + CHR$(BYTE):
     :
     WRD = 1:
     GOTO 2370
2530 IF NOT WRD THEN
       LP = 1:
       GOTO 2640
2540 IF BYTE = A5 OR BYTE = A6 THEN
       GO TO 2520:                     ! '$' OR '%'
2550 WRD = 0:
     POKE 33,22:
     IF SYMBOL$ <  > "" THEN
       VTAB 24:
       HTAB 1:
       PRINT LNE;TAB( 9);SYMBOL$:
       IF NOT TEST THEN
          SYMBOL = 1
2560 POKE 33,39:
     IF KNT = 0 THEN
       GO TO 2600
2570 FOR I = 1 TO KNT
2580   IF SYMBOL$ = CO$(I) THEN
          GO TO 2630:                  ! NOT A NEW SYMBOL
2590 NEXT
2600 IF SYMBOL$ = "" THEN
                                                                   LACRAB
                                                                 04/05/82
                                                                 PAGE - 7
==========================================================================

       GO TO 2770
2610 KNT = KNT + 1:
     I = KNT:
     IF KNT > 200 THEN
       PRINT "TOO MANY SYMBOLS FOR CROSS REFERENCE";CHR$(7):
       STOP
```

```
2620 VTAB 14:
     HTAB 36:
     PRINT RIGHT$("     " + STR$(KNT),4);:
     VTAB 16:
     HTAB 25:
     HTAB 25:
     VTAB 16:
     PRINT RIGHT$("               " + SYMBOL$,15):
     CO$(KNT) = SYMBOL$
2630 SYMBOL$ = "":
     IF SYMBOL AND NOT NN THEN
        NN = I
2640 IF SYMBOL AND (BYTE = C8 OR RD) THEN
        DF%(NN) = 1:
        SYMBOL = 0
2650 IF BYTE = C7 THEN
        TEST = 1:                         ! BEGIN 'IF'
2660 IF BYTE = C2 THEN
        SYMBOL = 0:
        NN = 0:                           ! 'COLON'
2670 IF BYTE = C3 THEN
        TEST = 0:
        NN = 0:
        SYMBOL = 0:                       ! END 'IF'
2680 IF LP THEN
        LP = 0:
        GOTO 2370
2690 IF LEN(CO$(I)) > MAX THEN
        MAX = LEN(CO$(I))
2700 IF LEN(LINE$(I)) >  = 254 THEN
        GO TO 27450
2710 LINE$(I) = LINE$(I) + CHR$(LNE / BLK) + CHR$(LNE - INT(LNE / BLK) * BL
     K)
2730 GOTO 2770
2750 PRINT "TOO MANY REFERENCES TO ";CO$(I);".":
     PRINT "REFERENCES AFTER LINE #";LNE;" IGNORED.":
     GOTO 2770
2770 IF BYTE = 0 THEN
        LOC = LOC + 2:
        GOTO 2340
2780 GOTO 2370
                                  +----------------------+
2790                              ! SORT CROSS REFERENCE !
                                  +----------------------+
2800 FLASH :
     VTAB 20:
     HTAB 25:
     PRINT "SORTING":
     NORMAL
```



```
2810 FOR L = KNT TO 2 STEP  - 1:
        K = 1:
        C$ = LEFT$(CO$(1),2) + RIGHT$(CO$(1),1)
2820    FOR J = 2 TO L:
           B$ = LEFT$(CO$(J),2) + RIGHT$(CO$(J),1):
           IF B$ > C$ THEN
              K = J:
              C$ = B$
2830    NEXT :
        A$ = CO$(L):
        CO$(L) = CO$(K):
        CO$(K) = A$:
        A$ = LINE$(L):
        LINE$(L) = LINE$(K):
        LINE$(K) = A$:
        I = DF%(L):
        DF%(L) = DF%(K):
        DF%(K) = I:
        I = PEEK( - 16336):
     NEXT
```

```
                              +-----------------------+
2850                          ! OUTPUT CROSS REFERENCE !
                              +-----------------------+
2860 PR# 1:
     POKE 33,40:
     POKE 34,0:
     HOME :
     BYTE = 1:
     SPACE = 0:
     RM = 0:
     MAX = MAX + 2:
     MX = WIDTH - SPACE - MAX:
     LNE$ = "":
     TXT$ = " ":
     GOSUB 3240:
     LNE$ = "":
     TXT$ = "BEGIN CROSS REFERENCE......":
     GOSUB 3240
2870 LNE$ = "":
     TXT$ = " ":
     GOSUB 3240:
     SPACE = 1:
     OLD$ = " ":
     I = TRASH:                        ! UNDEFINED SYMBOL FOR TEST
                                         PURPOSES
2880 FOR I = 1 TO KNT
2890    IF CO$(I) = " " THEN
           LNE$ = LEFT$("                              ",MAX):
           GOTO 2980
2900    CO$(I) = "  " + CO$(I)
2910    IF NOT DF%(I) THEN
           CO$(I) = "->" + MID$(CO$(I),3):
           GOTO 2930:                  ! UNDEFINED SYMBOL
2920    IF LEN(LINE$(I)) = 2 THEN
           CO$(I) = " *" + MID$(CO$(I),3):! SYMBOL ONLY OCCURS ON ONE LINE


                                                                  LACRAB
                                                                04/05/82
                                                                PAGE - 9
==========================================================================

2930    LNE$ = LEFT$(CO$(I) + "                              ",MAX)
2940    IF MID$(CO$(I),3,2) <  > MID$(OLD$,3,2) THEN
           GO TO 2970:                 ! CHECK FOR DUPLICATE SYMBOLS
2950    A$ = RIGHT$(CO$(I),1):
        IF A$ = "%" OR A$ = "$" THEN
           IF RIGHT$(OLD$,1) = A$ THEN
              LNE$ = "**" + MID$(LNE$,3)
2960    IF A$ > "%" AND RIGHT$(OLD$,1) > "%" THEN
           LNE$ = "**" + MID$(LNE$,3)
2970    IF CO$(I) <  > " " THEN
           OLD$ = CO$(I)
2980    FOR J = 1 TO LEN(LINE$(I)) STEP 2:! DECODE LINE NUMBERS
3000       L = ASC(MID$(LINE$(I),J,1)) * BLK + ASC(MID$(LINE$(I),J + 1,1))
3010       IF L <  > LOLD THEN
              TXT$ = TXT$ + RIGHT$("      " + STR$(L),6)
3020       LOLD = L:
           IF LEN(TXT$) >  = 249 THEN
              LINE$(I) = MID$(LINE$(I),J + 2):
              CO$(I) = " ":
              I = I - 1:
              GOTO 3040
3030    NEXT J
3040    LOLD = 0
3050    IF LEN(TXT$) < MX THEN
           GO TO 3140:                 ! CHECK LINE LENGTH
3060    FOR J = 1 TO MX - 1:
           K = MX - J
3070       IF MID$(TXT$,K,1) = " " THEN
              GO TO 3090
3080    NEXT J
3090    K = K - 1:
        IF MID$(TXT$,K,1) = " " THEN
           GO TO 3090
3100    B$ = MID$(TXT$,K + 1)
```

```
3110    TXT$ = LEFT$(TXT$,K):
        GOSUB 3240:
        TXT$ = B$
3120    LNE$ = LEFT$("                        ",MAX)
3130    GOTO 3050
3140    GOSUB 3240
3150 NEXT I
3160 LNE$ = "":
     TXT$ = " ":
     GOSUB 3240:
     LNE$ = "NOTES:":
     GOSUB 3240
3170 LNE$ = " * INDICATES SYMBOL REFERENCED ONLY ONCE.":
     GOSUB 3240
3180 LNE$ = "** INDICATES SYMBOL EQUIVALENT TO PREVIOUSLY DEFINED SYMBOL.":
     GOSUB 3240
3190 LNE$ = "-> INDICATES UNDEFINED SYMBOL.":
     GOSUB 3240
3200 LNE$ = "   PROGRAM IS " + STR$(SIZE) + " BYTES LONG.":
     GOSUB 3240
3210 LNE$ = "   COMMENTS ACCOUNT FOR APPROXIMATLY " + STR$(KM) + " BYTES ("
```



```
================================================================================
      + STR$(INT(KM / SIZE * 100)) + "%)."
3220 GOSUB 3240
3230 GOTO 3320
                                  +------------------+
3240                              ! PRINT OUT A LINE !
                                  +------------------+
3250 IF NOT FN PAGE(N) THEN
        FOR T = 1 TO SKIP:
           PRINT " ":
        NEXT :
        SKIP = 6:
        N = N + 11:
        PRINT NAME$:
        PRINT TIME$:
        A$ = "PAGE - " + STR$(INT(N / S6) + 1):
        PRINT SPC( WIDTH - LEN(A$));A$:
        PRINT US$:
        PRINT " ":
        X = FRE(0)
3260 LX = LEN(LNE$):
     PRINT LNE$;SPC( SPACE);LEFT$(TXT$,WIDTH - SPACE - LX):
     IF (LEN(TXT$) + SPACE + LX) <  = WIDTH THEN
        GO TO 3290:                       ! TEXT FITS ON ONE LINE
3270 TXT$ = RIGHT$(TXT$,LEN(TXT$) + LX + SPACE - WIDTH):
     IF RM THEN
        FOR T = 1 TO KOMMENT - 2 - SPACE:
           TXT$ = " " + TXT$:
        NEXT
3280 LNE$ = "      ":
     N = N + 1:
     GOTO 3250
3290 LNE$ = "      ":
     TXT$ = "":
     RM = 0:
     N = N + 1:
     IF BYTE = 0 THEN
        SPACE = SPACE - THN:
        THN = 0
3300 IF NFR THEN
        SPACE = SPACE + 3:
        NFR = 0
3310 RETURN
                                      +---------+
3320                                  ! WRAP UP !
                                      +---------+
3330 IF COMMENTS THEN
        CX = 1:
        GOTO 1850
```

```
3340 PR# 0
3350 IF XREF THEN
       GO TO 2050
3360 PR# 1
3370 FOR I = 1 TO 75 - FN PAGE(N):
       PRINT " ":
     NEXT
```



```
==========================================================================

3380 POKE 175,PEEK(103):
     POKE 176,PEEK(104):                    ! RESET EOP POINTERS
3390 IF PEEK(175) = 255 THEN
       POKE 176,PEEK(104) - 1
3400 POKE 103,1:
     POKE 104,8:                            ! RESET SOP POINTERS
3410 PR# 0:
     HOME :
     PRINT N;" LINES PRINTED."
3420 PRINT CHR$(7);"LISTING COMPLETE...."
3430 END
                          +-----------------------------+
3440                      ! DATA INITIALIZATION SECTION !
                          +-----------------------------+
3441 BYTE = 0:
     LOC = 0:
     B5 = 5:
     B6 = 2049
3450 DIM TKN$(127),CO$(200),LINE$(200),DF%(200)
3460 C1 = 178:
     C2 = 58:
     C3 = 196:
     C4 = 129:
     C5 = 130:
     C6 = 128:
     C7 = 173:
     C8 = 208:
     C9 = 34:
     A1 = 65:
     A2 = 90:
     A3 = 48:
     A4 = 57:
     A5 = 36:
     A6 = 37:
     A7 = 44:
     A8 = 127:
     B1 = 190:
     B2 = 132:
     B3 = 135:
     B4 = 131
3470 FOR I = 1 TO 107
3480   READ TKN$(I)
3490 NEXT I
3500 TKN$(36) = TKN$(36) + ":":
     TKN$(37) = TKN$(37) + ":"
3510 DATA END,,,DATA,INPUT,DEL,DIM,READ,GR,TEXT
3520 DATA  PR#,IN#,CALL,PLOT,HLIN,VLIN,HGR2,HGR,HCOLOR=,HPLOT
3530 DATA  DRAW,XDRAW,HTAB,HOME,ROT=,SCALE=,SHLOAD,TRACE,NOTRACE,NORMAL
3540 DATA  INVERSE,FLASH,COLOR=,POP,VTAB,HIMEM,LOMEM,ONERR,RESUME,RECALL
3550 DATA  STORE,SPEED=,LET,GOTO,RUN,IF,RESTORE,&,GOSUB,RETURN
3560 DATA  REM,STOP,ON,WAIT,LOAD,SAVE,DEF,POKE,PRINT,CONT
3570 DATA LIST,CLEAR,GET,NEW,TAB(," TO",FN,SPC(,,AT
3580 DATA NOT," STEP"," +"," -"," *"," /",^," AND"," OR"
3590 DATA " >"," ="," <",SGN,INT,ABS,USR,FRE,SCRN(,PDL,POS
3600 DATA  SQR,RND,LOG,EXP,COS,SIN,TAN,ATN,PEEK,LEN
```



```
==========================================================================

3610 DATA  STR$,VAL,ASC,CHR$,LEFT$,RIGHT$,MID$
3620 HOME :
     WIDTH = 76:                            ! ASSIGN PAGE WIDTH
```

```
3630 INPUT " ";NAME$:
     INPUT "PROGRAM NAME?";NAME$
3640 PRINT "DATE/TIME?";
3650 GET A$:
     I = ASC(A$):
     IF I = 13 THEN
        GO TO 3690
3660 IF I = 8 THEN
        TIME$ = LEFT$(TIME$,LEN(TIME$) - 1):
        GOTO 3680
3670 TIME$ = TIME$ + A$
3680 VTAB 3:
     HTAB 11:
     PRINT TIME$;:
     GOTO 3650
3690 PRINT " "
3700 FOR I = LEN(NAME$) TO WIDTH - 1
3710    NAME$ = " " + NAME$:
     NEXT
3720 FOR I = LEN(TIME$) TO WIDTH - 1
3730    TIME$ = " " + TIME$:
     NEXT
3740 FOR I = 1 TO WIDTH:
        US$ = US$ + "=":
     NEXT
3750 SKIP = 3:
     BLK = 256
3760 LOC = 2051:                        ! START OF PROGRAM
3770 EOP = PEEK(103) + PEEK(104) * BLK - 2:! END OF PROGRAM POINTER
3780 SPACE = 1:                         ! INITIAL SPACING AFTER LINE
                                          NUMBER
3790 S6 = 66
3800 DEF FN PAGE(N) = N - INT(N / S6) * S6:! NEW PAGE DETECTION FUNCTION
3810 KOMMENT = 41:                      ! TABBING FOR IN-LINE COMMENTS
3820 GOSUB 3850:                        ! DISPLAY MENU
3830 HOME
3840 RETURN
                                   +--------------+
3850                               ! DISPLAY MENU !
                                   +--------------+
3860 HOME :
     INVERSE
3870 FOR I = 1 TO 7
3880    READ J:
        IF J <  > 0 THEN
           VTAB I:
           HTAB J:
           PRINT " ";:
           GOTO 3880
3920 NEXT
3930 DATA 3,11,16,17,18,21,22,23,24,29,33,34,35,36,0
3940 DATA 3,10,12,15,19,21,25,28,30,33,37,0
                                                                  LACRAB
                                                                04/05/82
                                                               PAGE - 13
==========================================================================

3950 DATA 3,9,13,15,21,25,27,31,33,37,0
3960 DATA 3,9,13,15,21,22,23,24,27,31,33,34,35,36,0
3970 DATA 3,9,10,11,12,13,15,21,23,27,28,29,30,31,33,37,0
3980 DATA 3,9,13,15,19,21,24,27,31,33,37,0
3990 DATA 3,4,5,6,7,9,13,16,17,18,21,25,27,31,33,34,35,36,0
4000 HTAB 1
4010 VTAB 9
4020 PRINT "========================================"
4030 NORMAL
4040 VTAB 11:
     PRINT TAB( 15);"SYSTEM MENU"
4050 PRINT TAB( 19);"FOR"
4060 PRINT TAB( 15);"PROGRAM TO:"
4070 VTAB 15
4080 INVERSE :
     PRINT "L";:
     NORMAL :
     PRINT "IST ";
```

```
4090 INVERSE :
     PRINT "A";:
     NORMAL :
     PRINT "ND ";
4100 INVERSE :
     PRINT "C";:
     NORMAL :
     PRINT "ROSS ";
4110 INVERSE :
     PRINT "R";:
     NORMAL :
     PRINT "EFERENCE ";
4120 INVERSE :
     PRINT "A";:
     NORMAL :
     PRINT "PPLESOFT ";
4130 INVERSE :
     PRINT "B";:
     NORMAL :
     PRINT "ASIC"
4140 VTAB 17:
     HTAB 12
4150 PRINT "1) LISTING ONLY"
4160 VTAB 19:
     HTAB 12
4170 PRINT "2) CROSS REFERENCE ONLY"
4180 VTAB 21:
     HTAB 12
4190 PRINT "3) LISTING AND XREF"
4200 VTAB 23:
     HTAB 13:
     FLASH
4210 PRINT "WHICH OPTION?";:
     NORMAL
4220 GET A$
4230 IF A$ = "1" OR A$ = "3" THEN
        LST = 1:                         ! SET LISTING FLAG
```



```
4240 IF A$ = "2" OR A$ = "3" THEN
        XREF = 1:                        ! SET CROSS REFERENCE FLAG
4250 IF A$ < "1" OR A$ > "3" THEN
        PRINT CHR$(7):
        GOTO 4200
4260 RETURN
```

4
GRAPHICS

Graphics

This section includes programs to help you understand and take advantage of the Apple II's superb graphics capabilities.

Dick Vile's "Apple Bits" makes use of the low-resolution graphics feature with utilities to build shapes and perform faster screen displays. Also included are interesting animation examples.

Art Radcliffe's "True 3-D Images" uses the versatility of the Apple's high-resolution system. By developing a stereo-pair of images, your flat monitor is given a new dimension of depth. Try out the noisy coaster and hold on to your seat!

"Apple Color Filter" by Stephen R. Berggren lets you erase any selected color from the high-resolution screen without affecting the other colors. This utility sheds light on how high-resolution color graphics work.

Apple Color Filter

by Stephen R. Berggren

This short machine-language subroutine will allow you to filter out any selected color from the Apple hi-resolution graphics screen.

One of the most fascinating capabilities of the Apple Graphics Tablet is its ability to separate the colors on the high-resolution graphics screen. It can act like a color filter, removing all colors from the screen except a chosen one. This can be extremely useful in doing computer art work, drawing graphs, and, of course, in game graphics. But now you can have a similar capability without buying the graphics tablet. Just use this Apple color filter program.

The color filter is a short machine-language program which can erase any selected color from the high-resolution screen while leaving the other colors unaffected. To use it, simply load it into page 3 of memory, starting at decimal 768. Then POKE a number from 1 to 4 into memory location 769 and run it with a call 768. The number POKEd into 769 determines what color is erased: 1 erases green, 2 erases violet, 3 erases blue and 4 erases orange. The program takes only about one-fourth of a second to filter the entire page-one hi-res screen.

If you are using only green, violet, blue and orange, everything works fine. But the Apple also draws in white — in fact two kinds of white. This can affect the results of the filter operation. The Apple makes its two whites by combining either green and violet (HCOLOR = 3) or blue and orange (HCOLOR = 7). The color filter "sees" the white as a combination of the two colors rather than as a separate color. Thus when told to erase green, it will erase all green, including the green part of any white that is made up of green and violet. This turns the white into violet. Of course, any white made up of blue and orange is left alone. So to erase white, simply erase the two colors that make it up. To avoid changing the white to another color, simply draw it in the colors that you do not plan to filter out later.

How the color filter works delves deeply into the mysteries of Apple color graphics. From what I have been able to deduce, it seems that each byte in the hi-res memory holds seven screen dots. Each set bit in the lower seven bits will turn on one dot. The highest bit determines whether the dots will be green and violet, or blue and orange. On even bytes, bits 0, 2, 4 and 6 create violet or blue while bits 1, 3 and 5 create green or orange. On odd bytes, this sequence is reversed. The color filter masks out all of the bits in the hi-res memory area that would create a particular color. By changing all of these color bits to 0, it eliminates the color. The comments in the source program listing give more detail on how the program operates.

Two bytes of zero-page memory are needed for the indirect addressing. The program uses bytes 6 and 7, but any two consecutive bytes can be used. As written, the program works only on hi-res page one, but by changing the values of LOSCRN to 40 and HISCRN to 60, you can make it work on hi-res page two. Finally, if you don't have an assembler, you can simply load the hexadecimal values listed in the table using the Apple monitor's data entry function.

The colors I have referred to here are the ones I get from my Apple on my television. The colors you get may be different. The best approach is to experiment with the program on your system to see what number inputs erase what colors. The Applesoft BASIC demonstration program listed here should give you a good idea of how the color filter works on your system.

```
0800                ;************************
0800                ;*                      *
0800                ;*  APPLE COLOR FILTER  *
0800                ;*   STEPHEN BERGGREN   *
0800                ;*                      *
0800                ;*  COPYRIGHT (C) 1982  *
0800                ;*    MICRO INK, INC.   *
0800                ;* CHELMSFORD, MA 01824 *
0800                ;*  ALL RIGHTS RESERVED *
0800                ;*                      *
0800                ;************************
0800                ;
0800                ;
0800                ;PUT NUMBER FOR COLOR TO BE REMOVED IN $301
0800                ;1=GREEN, 2=VIOLET, 3=BLUE, AND 4=ORANGE
0800                ;WHITE #3 NOT AFFECTED BY 3 OR 4
0800                ;WHITE #7 NOT AFFECTED BY 1 OR 2
0800                ;
0800                ;TO RUN, 300G FROM MONITOR OR CALL 768 FROM BASIC
0800                ;
0006                SCRLOC   EPZ $06          ;ZERO-PAGE LOC. FOR ADDRESSING
 SCREEN
0020                LOSCRN   EPZ $20          ;HI-BYTE OF ADDRESS OF SCREEN
START
0040                HISCRN   EPZ $40          ;HI-BYTE OF SCREEN END
0800                ;
0300                         ORG $300
0300                         OBJ $800
0300                ;
0300 A2 00                   LDX #$00         ;PUT COLOR VALUE IN X FOR TABL
E INDEXING
0302 A0 00                   LDY #$00         ;PUT 0 IN Y FOR INDIRECT SCREE
N INDEXING
0304 A9 00                   LDA #$00         ;SET SCREEN START ADDRESS IN S
CRLOC
0306 85 06                   STA SCRLOC
0308 A9 20                   LDA #LOSCRN
030A 85 07                   STA SCRLOC+1
030C                ;
030C B1 06          EVNBYT   LDA (SCRLOC),Y   ;GET SCREEN BYTE
030E 30 08                   BMI DOTAB2       ;IF BIT 7 SET, USE TABLE 2
0310 3D 45 03                AND TABLE1,X     ;MASK OFF COLOR BITS USING TAB
LE 1
0313 91 06                   STA (SCRLOC),Y   ;PUT BACK THE BYTE
0315 4C 1D 03                JMP ODDBYT       ;DO THE NEXT BYTE
0318                ;
0318 3D 47 03       DOTAB2   AND TABLE2,X     ;MASK OFF COLOR BITS USING TAB
LE 2
031B 91 06                   STA (SCRLOC),Y   ;PUT BACK THE BYTE
031D                ;
031D E6 06          ODDBYT   INC SCRLOC       ;SET UP FOR NEXT SCREEN BYTE
031F B1 06                   LDA (SCRLOC),Y   ;GET SCREEN BYTE
0321 30 08                   BMI DOTAB4       ;IF BIT 7 SET, USE TABLE 4
0323 3D 49 03                AND TABLE3,X     ;MASK OFF COLOR BITS USING TAB
LE 3
0326 91 06                   STA (SCRLOC),Y   ;PUT BACK THE BYTE
0328 4C 30 03                JMP INCLOC       ;GO INCREMENT SCRLOC
032B                ;
032B 3D 4B 03       DOTAB4   AND TABLE4,X     ;MASK OFF COLOR BITS USING TAB
LE 4
032E 91 06                   STA (SCRLOC),Y   ;PUT BACK THE BYTE
0330                ;
0330 A9 00          INCLOC   LDA #$00         ;INCREMENT SCRLOC LO
0332 38                      SEC
0333 65 06                   ADC SCRLOC
0335 85 06                   STA SCRLOC
0337 90 D3                   BCC EVNBYT       ;IF NOT OVERFLOW, DO ANOTHER 2
 BYTES
0339 A9 00                   LDA #$00         ;INCREMENT SCRLOC HI
033B 38                      SEC
033C 65 07                   ADC SCRLOC+1
033E 85 07                   STA SCRLOC+1
```

```
0340 C9 40                  CMP #HISCRN      ;WAS THAT THE LAST PAGE?
0342 D0 C8                  BNE EVNBYT       ;IF NOT, DO NEXT 2 BYTES
0344 60                     RTS              ;ALL DONE!
0345               ;
0345 00 D5         TABLE1   HEX 00D5
0347 AA FF         TABLE2   HEX AAFF
0349 FF AA         TABLE3   HEX FFAA
034B D5 FF FF      TABLE4   HEX D5FFFFD5AA
034E D5 AA
0350                        END
```

```
1  REM ************************
2  REM *                      *
3  REM *   COLOR FILTER DEMO ,*
4  REM *       BERGGREN       *
5  REM *  COPYRIGHT (C) 1982  *
6  REM *    MICRO INK, INC.   *
7  REM * CHELMSFORD, MA 01824 *
8  REM *  ALL RIGHTS RESERVED *
9  REM *                      *
10  REM ************************
12  REM
14  HGR : HOME : VTAB 22
20  FOR I = 1 TO 7
30  HCOLOR= I
40  HPLOT 0,I * 10 TO 250,I * 10 + 50
50  NEXT I
55  FOR J = 1 TO 5000: NEXT J
60  FOR I = 1 TO 4
70  PRINT : PRINT : PRINT "COLOR FILTER INPUT: "I
80  POKE 769,I
90  CALL 768
100  FOR J = 1 TO 5000: NEXT J
110  NEXT I
120  TEXT
130  END
```

True 3-D Images

by Art Radcliffe

Create stereo-pair images for viewing without accessory devices. The pair of images can be fused into a three-dimensional pattern by placing a piece of paper between the viewer's eyes and the viewing screen so that each eye sees only the appropriate image. With practice the paper is no longer needed. The object used for demonstration is a three-dimensional Lissajous figure.

This article discusses genuine three-dimensional images such as seen through your grandparents' stereopticon, or through more recent systems that require colored eye filters for viewing. The present technique involves not a single projection of the object, but a pair of images which can be fused into one 3-D image without auxiliary contrivances.

The *Scientific American* has published articles accompanied by stereo-pair images, which can be fused into a stereo scene with a little practice. This program was inspired by success with such viewing. Some eye training is required, and some eye strain may be felt initially. What is required is that you stare off into the distance (eyeball axes essentially parallel) while focussing nearby. The muscles which direct your eyeball and the muscles which focus your lens are accustomed to working in a coordinated way for distant or for nearby objects; this muscular habit can readily be broken. It is not at all difficult for me now to glance at a pair of images on the screen from anywhere in the room, and see the 3-D pattern.

The viewing images are produced by running rays from each defined point of the object to points which correspond to eye locations. The object is behind the screen and the eyes are in typical viewing positions. Points are plotted where these rays intercept the display plane.

The object is defined near the origin of an *X, Y, Z* coordinate system, behind the screen plane. We can define object points using the notation: *(X1,Y1,Z1)*, define screen points with: *(X2,Y2,Z2)*, and define the eye locations using: *(X3,Y3,Z3)*. *Z2*, the screen distance from the origin, is set at 200 in the program, and *Z3*, the eye distance from the origin, is set at 300. *Y3* is the same for each eye: 40; and the *X3* values for the two eyes are 40 and 120. The direction from which the object is viewed can be altered by offsetting *X1* and *Y1*.

Use of proportions leads us to the conclusion that *(X2-X1)/(Z2-Z1) = (X3-X1)/(Z3-Z1)* and similarly, *(Y2-Y1)/(Z2-Z1) = (Y3-Y1)/(Z3-Z1)*. From these equations we can derive *X2 = X1 + M(X3-X1)* and *Y2 + Y1 + M(Y3-Y1)* where *M = (Z2-Z1)/(Z3-Z1)*.

Listing 1 is an embellishment, with sound effects, of the program as originally written (see listing 2).

Within the program there are variable substitutions: (X,Y,Z) = *(X1,Y1,Z1)*, (A,B,C) = *(X2,Y2,Z2)* and (D,F,G),(E,F,G) = *(X3,Y3,Z3)*. A Lissajous pattern was chosen for viewing. It can be restricted to a rectangular area, derived from the property of the sine function, being bounded by 1 and – 1. In the program a raised sine is used by adding 1 (line 64) to avoid negative values. Thus, the *X*-coordinates of the object vary according to one sine function, the *Y*-coordinates of the object vary in a coordinated manner according to a second sine function, and the *Z*-coordinate varies according to a third sine function.

Random numbers are used to achieve an almost infinite variety of patterns. It is fun to watch the pattern take shape; the eye can go on a roller-coaster ride with the leading edge of the pattern as it develops on the screen.

There is an inherent limitation to this method in that the display area is limited to the space between the primary pair of images. Use of prismatic glasses might increase the available object size. The program is written for viewing on a twelve-inch diagonal screen. Users with other size displays may want to alter program parameters, first increasing or decreasing the *X* dimension for eye position by altering one or both of parameters D and E. It may also be useful to alter the scale factor N.

Interesting 3-D motion displays could be written in machine language. I can also imagine game possibilities, including visual 3-D Tic Tac Toe.

I have experimented with more general systems using color filters for viewing, and may report on this at some future time. I hope that readers will experiment with this viewing system, perhaps altering parameters of the given program or substituting another object. Data points in three dimensions might be seen as a 3-D swarm of points in which local clusters or correlations could be detected. This is a new way of seeing things.

```
1  REM  *************************
2  REM  *                       *
3  REM  *     NOISY COASTER     *
4  REM  *     ART RADCLIFFE     *
5  REM  *                       *
6  REM  *   COPYRIGHT (C) 1982  *
7  REM  *     MICRO INK, INC.   *
8  REM  *  CHELMSFORD, MA 01824*
9  REM  *   ALL RIGHT RESERVED  *
10  REM *                       *
11  REM *************************
12  HOME : POKE 36,12: PRINT "NOISY COASTER"
20  DIM A%(299): DIM B%(299): DIM H%(299): DIM S(299)
30 A = B = C = D = E = F = G = H = I = J = 0
40 K = L = M = N = O = P = Q = T = U = V = 0
50 W = X = Y = Z = 0:R =  - 16336:S = .5:LL = 0
60  GOTO 630
65  REM ------------------------
70  PRINT  CHR$ (7): PRINT  CHR$ (7): FOR A = 0 TO 1000: NEXT : PRINT  CHR$
    (7)
80  FOR P = 0 TO 299
90 A =  PEEK (R)
100  HCOLOR= 3: REM  FRONT OF TRAIN
110 B = A%(P):C = B%(P):D = H%(P)
120 E = B + 1:F = C + 1:G = D + 1
130  HPLOT B,F: HPLOT E,C: HPLOT E,F
140  HPLOT D,F: HPLOT G,C: HPLOT G,F
150 Q = P - 10
160 A =  PEEK (R)
170  IF Q < 0 THEN Q = P + 289: REM 0<=Q<=360DEG
180  HCOLOR= 0: REM END OF TRAIN
190 B = A%(Q):C = B%(Q):D = H%(Q)
200 E = B + 1:F = C + 1:G = D + 1
210  HPLOT B,F: HPLOT E,C: HPLOT E,F
220  HPLOT D,F: HPLOT G,C: HPLOT G,F
230 A =  PEEK (R): REM REPLOT TRACK ->
240  HCOLOR= 3: HPLOT B,C: HPLOT D,C
250 A =  PEEK (R)
260  FOR Z = 0 TO LL - B%(P): NEXT : REM  TRAIN SPEED
270 A =  PEEK (R)
280  NEXT P
290  PRINT  CHR$ (7)
300  RETURN
305  REM ------------------------
310  FOR P = 0 TO 299: REM ESTABLISH PATTERN
320 X = S(I) + L:Y = 2 * S(J) + T:Z = S(K)
330 M = (C - Z) / (G - Z)
340 A =  INT (S + X + M * (E - X)):A%(P) = A: REM  LEFT X
350 B =  INT (S + Y + M * (F - Y)) - 50:B%(P) = B: REM  Y
360 H =  INT (S + X + M * (D - X)):H%(P) = H: REM  RIGHT X
370  HPLOT A,B: HPLOT A + 2,B: HPLOT H,B: HPLOT H + 2,B
380  IF LL < B THEN LL = B
390 I = I + U: IF I > 299 THEN I = 0
400 J = J + V: IF J > 299 THEN J = 0
410 K = K + W: IF K > 299 THEN K = 0
420  NEXT P
430  RETURN
435  REM ------------------------
440 O = 8 *  ATN (1) / 300: REM 360DEG/300
450 N = 40: REM OBJECT SCALE FACTOR
460  FOR A = 0 TO 299
470 S(A) = N * (1 +  SIN (A * O)): REM SINE+1>0
480  NEXT A
490 C = 200: REM X COOR'S OF EYES
500 D = 120
510 E = 40: REM Y COOR'S OF EYES
520 F = 40
530 L = 150: REM  X,Y,Z COOR'S OF OBJECT
540 T = 250
550 G = 300: REM # CYCLES IN X,Y,Z ->
560 U =  INT (1 + 5 *  RND (1))
570 V =  INT (1 + 5 *  RND (1)): IF V = U THEN 570
580 W =  INT (1 + 5 *  RND (1)): IF W = V OR W = U THEN 580
590 I =  INT (300 *  RND (1)): REM START POINTS
```

```
600 J =  INT (300 *  RND (1))
610 K =  INT (300 *  RND (1))
620  RETURN
625  REM -----------------------
630  PRINT : PRINT : PRINT "  CREATED BY ART RADCLIFFE, ANN ARBOR   ": PRINT

640  PRINT : PRINT "PLACE 8 INCH BY 12 INCH CARDBOARD       "
650  PRINT "BETWEEN SCREEN AND TIP OF NOSE SO EACH  "
660  PRINT "EYE SEES ONLY IT'S IMAGE.  SOME EYE     "
670  PRINT "TRAINING IS NECESSARY.          "
680  PRINT : PRINT : PRINT : PRINT : PRINT
690  PRINT "PLEASE BE PATIENT WHILE I MEDITATE TO   "
700  PRINT "GET MYSELF READY FOR THIS..............."
705  REM -----------------------
710  GOSUB 440 REM INITIALIZE
720  HOME : HGR : HCOLOR= 3
730 LL = 0: REM LOWEST POINT
740  GOSUB 310 REM LAY TRACK
750  FOR A = 0 TO 999: NEXT
760  GOSUB 70 REM HOLD TIGHT!
770  FOR A = 0 TO 3000: NEXT
780  GOSUB 490 REM REINITIALIZE
790  GOTO 720 REM START OVER
800  END
```

```
1  REM ************************
2  REM *                      *
3  REM *   LISSAJOUS FIGURES  *
4  REM *     ART RADCLIFFE    *
5  REM *                      *
6  REM *  COPYRIGHT (C) 1982  *
7  REM *    MICRO INK, INC.   *
8  REM * CHELMSFORD, MA 01824 *
9  REM * ALL RIGHTS RESERVED  *
10  REM *                     *
11  REM ***********************
14  HGR : HCOLOR= 3: PRINT : PRINT : PRINT "WAIT"
16  DIM S(199)
18 A = B = C = D = E = F = G = H = I = S = 0
20 J = K = L = M = N = O = P = X = Y = Z = 0
22  GOTO 56
24  FOR P = 0 TO 199
26 X = S(I) + L
28 X = S(J) + T
30 Z = S(K)
32 M = (C - Z) / (G - Z)
34 A =  INT (S + X + M * (E - X))
36 B =  INT (S + Y + M * (F - Y))
38 H =  INT (S + X + M * (D - X))
40  HPLOT A,B: HPLOT H,B
42 I = I + U: IF I > 199 THEN I = 0
44 J = J + V: IF J > 199 THEN J = 0
46 K = K + W: IF K > 199 THEN K = 0
48  NEXT
50  FOR Z = 0 TO 5000: NEXT Z
52  HGR
54  GOTO 22
56 O = .04 *  ATN (1)
58 N = 40
60  FOR A = 0 TO 199
62 B = A * O
64 S(A) = N * (1 +  SIN (B))
66  NEXT
68 C = 200
70 D = 120
72 E = 40
74 F = 40
76 G = 300
78 T = 250
80 L = 150
82 U =  INT (1 + 5 *  RND (1))
84 V =  INT (1 + 5 *  RND (1)): IF V = U THEN 84
86 W =  INT (1 + 5 *  RND (1)): IF W = V OR W = U THEN 86
88 I =  INT (199 *  RND (1))
90 J =  INT (199 *  RND (1))
92 K =  INT (199 *  RND (1))
94 S = .5
96  POKE 49234,0
98  GOTO 24
```

Apple Bits

by Richard C. Vile

This article describes several aids to faster and more efficient low-resolution graphics programming, including machine-language routines.

Part 1

This is the first part in a series dealing with the use of Apple II low-resolution graphics features. Some techniques will be described that use machine language to enhance the speed of graphics applications and reduce the amount of memory required in order to represent certain screen patterns.

The basic techniques described will enable you to display patterns 8 × 8 in size or smaller, and consisting of a single color. Larger patterns must be constructed from smaller pieces which fit these requirements. A modification of the machine-language routine will allow multiple colors to be obtained by overlaying.

Bit-encoding a Picture

Consider the following eight hexadecimal numbers:

38,38,12,FE,90,28,44,83

Believe it or not, they contain a picture! To see how, let's first rewrite the numbers in binary, using the following table to convert each hex digit into a 4-bit binary "nibble:"

Hex	Binary		
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

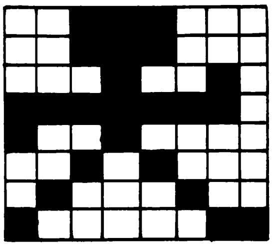
We arrive at the following numbers:

0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	0	0	1	0
1	1	1	1	1	1	1	0
1	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0
1	0	0	0	0	0	1	1

Do you see the picture yet? Just in case you don't, let's transform the pattern of 0's and 1's onto "graph paper" by superimposing a grid of squares on top of the above list, like so:

0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
0	0	0	1	0	0	1	0
1	1	1	1	1	1	1	0
1	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0
0	1	0	0	0	1	0	0
1	0	0	0	0	0	1	1

Now, erase all the 0's and completely blacken the squares containing the 1's. That gives the grid shown next:



Now, of course, you see the "picture." Erasing the grid lines should make the correspondence with the lo-res display pretty obvious as well. The question now becomes: "How do we turn the above process into a program?"

Shown in listing 1 is a machine-language program which will carry out the process. It "assumes" that certain information has been set up for it. This information will be illustrated by listing 2 (in Integer BASIC).

The BASIC program does a series of POKEs which set up the machine-language routine's information:

```
12 POKE 2048,7: POKE 2049,7
```

indicates the width and height of the patterns to be displayed.

```
28 POKE 36,COL: POKE 37,ROW
```

indicates the ROW and COLUMN of the lo-res screen where the upper-left corner of the pattern to be displayed will be.

```
60 POKE 60,(3072+OFFSET)MOD 256

65 POKE 61,(3072+OFFSET)/256
```

stores the address in Apple II RAM where the numerical codes for the pattern to be displayed begin.

The machine-language program is invoked by the line:

```
70 CALL 2058
```

Running the Fireworks Animation

The numerical data the program uses must first be entered into memory. This data resides at locations C00 to D27 (3072-3367). Once you have entered it using the monitor, save it on tape (C00.D27W) or on disk

```
*3DOG
>BSAVE SPARKS,A$C00,L$127
```

to avoid keying it in again later. Likewise, enter the machine-language program using the monitor or the mini-assembler and save it:

```
*800.857W                                  (Tape)
```
or
```
*3DOG
> BSAVE APPLE-BITS,
A$800,L$57                                 (Disk)
```

To run the program, you should issue the command

```
>LOMEM:4096
```

so that BASIC doesn't clobber the machine-language program.

Assuming you are using a disk-based system, the entire sequence of commands needed to run the animation would be:

```
>BLOAD APPLE-BITS
>BLOAD SPARKS
>LOMEM:4096
>RUN FIREWORKS
```

(If you'd rather not key in long command sequences, cook up an EXEC file with the commands in it.)

Listing 1

```
080A-   A5 30      LDA   $30
080C-   8D 04 08   STA   $0804
080F-   AC 00 08   LDY   $0800
0812-   8C 03 08   STY   $0803
0815-   CE 03 08   DEC   $0803
0818-   30 31      BMI   $084B
081A-   AE 01 08   LDX   $0801
081D-   8E 02 08   STX   $0802
0820-   CA         DEX
0821-   30 F2      BMI   $0815
0823-   BD 50 08   LDA   $0850,X
0826-   AC 03 08   LDY   $0803
0829-   31 3C      AND   ($3C),Y
082B-   D0 04      BNE   $0831
082D-   A9 00      LDA   #$00
082F-   85 30      STA   $30
0831-   A5 24      LDA   $24
0833-   18         CLC
0834-   6D 03 08   ADC   $0803
0837-   A8         TAY
0838-   A5 25      LDA   $25
083A-   8E 02 08   STX   $0802
083D-   6D 02 08   ADC   $0802
0840-   20 00 F8   JSR   $F800
0843-   AD 04 08   LDA   $0804
0846-   85 30      STA   $30
0848-   4C 20 08   JMP   $0820
084B-   60         RTS
084C-   80         ???
084D-   10 10      BPL   $085F
084F-   F8         SED
0850-   01 02      ORA   ($02,X)
0852-   04         ???
0853-   08         PHP
0854-   10 20      BPL   $0876
0856-   40         RTI
0857-   80         ???
0858-   A8         TAY
0859-   B0 08      BCS   $0863
085B-   28         PLP
*
```

Listing 2

```
 1 REM ***********************
 2 REM *                     *
 3 REM *      FIREWORKS      *
 4 REM *      R. C. VILE     *
 5 REM *                     *
 6 REM *  COPYRIGHT (C) 1982 *
 7 REM *    MICRO INK, INC.  *
 8 REM * CHELMSFORD, MA 01824*
 9 REM *  ALL RIGHTS RESERVED*
10 REM *                     *
12 REM ***********************
14 GR : PRINT : PRINT : PRINT
15 POKE 2048,7: POKE 2049,7
17 ROW=7+ RND (27)
20 COL=7+ RND (27)
25 COLOR= RND (15)+1
28 POKE 36,COL: POKE 37,ROW
30 FOR J=1 TO RND (10)
40 SPARK=1+ RND (20)
50 OFFSET=SPARK*7
60 POKE 60,(3072+OFFSET) MOD 256
65 POKE 61,(3072+OFFSET)/256
70 CALL 2058
72 FOR DE=1 TO 25: NEXT DE
75 NEXT J
80 COLOR=0: FOR J=0 TO 6: HLIN COL,COL+6 AT ROW+J: NEXT J
85 GOTO 17
```

Part 2

The Pattern Maker Program

This program lets you create patterns and store them in tables for subsequent use by animation programs. It begins by asking a couple of questions:

HEIGHT AND WIDTH OF PATTERNS?
TABLE ADDRESS IN DECIMAL?

The patterns created may be *up to* 8 rows high by 8 columns wide, but may be smaller than that as well. For example, one set of patterns that I use consists of 7 rows by 5 columns. They form a "giant" character set that may be used to create billboard messages on the Apple screen. The table of patterns is stored in Apple RAM and manipulated by PEEKs and POKEs. Thus, it is necessary to tell the program where in memory the table is located. I typically store tables at 3072 ($C00). The tables must be saved on tape or disk for eventual use by animation programs.

The program will display a rectangular border enclosing an area equal in size to the patterns specified, as shown in figure 1. Inside the pattern border you'll see a blinking cursor. You may move this cursor about, inside the border, and either add or delete parts of a pattern in the process.

The pattern maker will respond to any of the following commands:

PATTERN
VERIFY
MODIFY
RECORD
SAME
HELP
QUIT, BYE, STOP, EXIT

The commands are typed in full, or abbreviated to the first letter. If you forget what the commands are, simply type "HELP" or "H" and the menu of commands will be listed for you. (Note: You will probably lose any pattern in progress if you do that.)

The commands have the following effects:

PATTERN: The area inside the border is erased, the cursor appears inside, and the user may begin creating a new pattern.

MODIFY: Recalls a given pattern from the table, so the user may modify it.

SAME: Returns to the same pattern as the one most recently created or modified (allows the user to recover from accidently striking "ENTER" while creating a pattern.)

VERIFY: Displays the numeric codes for the pattern under construction or modification. Mainly included for debugging the pattern maker program itself.

HELP: Displays the menu of commands.

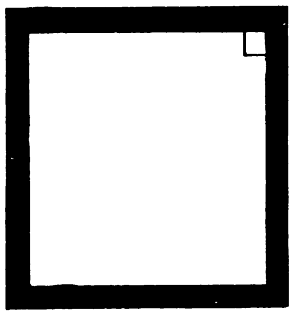
QUIT, BYE, STOP, EXIT: Cause the termination of the program. Note that the screen is cleared and returned to TEXT mode.

The program operates in mixed low-resolution graphics mode and uses the bottom four lines of the screen for entering commands and prompts. The program will prompt you by typing

```
COMMAND?
```

and then wait for a response. If any of the above commands are entered, the program will take the corresponding action, otherwise it simply reprompts the user. The "P", "M", and "S" commands will transfer the cursor inside the rectangle on the graphics portion of the screen. While there, you may enter "cursor control keys" or "pattern control keys" to shift the cursor around the pattern and create or erase parts of the pattern.

Figure 1: Building the Pattern



The cursor control keys and their results are listed in table 1 and the pattern control keys and their results are listed in table 2.

When RECORDing or MODIFYing patterns, the program will request a KEY to associate with the pattern. You should respond to this request by simply striking the desired key (do not hit ENTER, unless that is the desired key). Control keys (except for Control-C) are included. The association that is made "internally" by this is as follows: The program converts the ASCII value of the key struck to a table offset. This offset is then used when storing or retrieving the corresponding pattern from memory. The same idea will be used by animation programs in order to point the machine language driver at the correct positions in memory for a given pattern.

The pattern-maker program does not LOAD and SAVE the pattern tables itself. This is the responsibility of the user. For example, suppose you have created a table which starts at location $C00 and extends as far as $FFF. After exiting the pattern-maker program and returning to the Integer BASIC command level, you would give the following command, assuming that you have a disk-based system:

```
BSAVE PATTERN TABLE XYZ,A$C00,L$7FF
```

To save the same table on tape, you would enter the monitor and (after setting up your recorder, etc.) type

```
*C00.FFFW
```

and wait for the monitor to write it all out to the cassette.

Table 1

KEY	EFFECT
—>	Move the cursor one column to the right. If the cursor is already at the far right of the rectangle, then "wrap" around to the far left of the pattern, but one row further down. If at the extreme bottom right of the pattern, then "wrap" around to the extreme top left of the pattern.
R	Same as —>.
<—	Move the cursor one column to the left. At the extreme positions "wrap" around in a fashion analogous to that described above for the —> or R keys.
L	Same as <— .
U	Move the cursor up one row. (Wrap around also)
D	Move the cursor down one row. (Wrap also)
ENTER	Return the cursor to the command area of the screen.
ESC	Same as for "ENTER".

Table 2

KEY	EFFECT
+	Add a solid blob to the pattern in the position indicated by the current location of the cursor.
–	Erase the part of the pattern (if any) located at the current position of the cursor.

Note: If you store your tables in low memory, be sure to protect them from the BASIC program itself. For example, when I use the area from $C00 (decimal 3072) to $FFF, I first issue the command:

```
LOMEM: 4096
```

Final Note

The pattern-maker program uses the machine-language driver program (in order to support the Modify command). Thus, the following complete sequence of commands would be used to run the pattern maker to add or modify patterns previously saved in file BPATS:

```
> BLOAD BPATS
> BLOAD APPLE-BITS
> LOMEM: 4096
> RUN PATTERN MAKER
```

If no previous file of patterns, such as BPATS, is being used, then the first command in the sequence may be omitted.

```
   0 REM ****************************
   1 REM *                          *
   2 REM *       PATTERN MAKER      *
   3 REM *         R. C. VILE       *
   4 REM *                          *
   5 REM *     COPYRIGHT (C) 1982   *
   6 REM *        MICRO INK, INC.   *
   7 REM *   CHELMSFORD, MA. 01824  *
   8 REM *    ALL RIGHTS RESERVED   *
   9 REM ****************************
  10 DIM PTTERN(7),BITS(7),A$(25): GOSUB 10000
  11 INPUT "COMMAND? ",A$: IF A$="P" OR A$="PATTERN" THEN GOSUB 50
  12 IF A$="V" OR A$="VERIFY" THEN GOSUB 1000
  13 IF A$="M" OR A$="MODIFY" THEN GOSUB 1500
  14 IF A$="R" OR A$="RECORD" THEN GOSUB 2000
  15 IF A$="S" OR A$="SAME" THEN GOSUB 52
  16 IF A$="H" OR A$="HELP" THEN GOSUB 2500
  17 IF A$="Q" OR A$="QUIT" OR A$="BYE" OR A$="STOP" OR A$="EXIT" THEN GOTO
     3025
  45 GOTO 11
  50 FOR I=0 TO 7:PTTERN(I)=0: NEXT I: GR
  51 COLOR=1: HLIN 14,14+WIDTH+1 AT 14: HLIN 14,14+WIDTH+1 AT 14+HEIGHT+
     1: VLIN 14,14+HEIGHT+1 AT 14: VLIN 14,14+HEIGHT+1 AT 14+WIDTH+1
  52 SAVCOLR= SCRN(15+COL,15+ROW):KEY= PEEK (KBD): IF KEY>=128 THEN 57
  54 COLOR=15: PLOT 15+COL,15+ROW: FOR I=0 TO 10: NEXT I: COLOR=0: PLOT
     15+COL,15+ROW: FOR I=0 TO 10: NEXT I: IF SAVCOLR#15 THEN 52
  56 COLOR=15: PLOT 15+COL,15+ROW: COLOR=0: GOTO 52
  57 POKE CLR,0
  58 IF KEY=141 OR KEY=155 THEN RETURN : COLOR=15
  60 IF KEY# ASC("R") AND KEY#149 THEN 70:COL=COL+1: IF COL<WIDTH THEN 52
     :ROW=ROW+1:COL=0: IF ROW<HEIGHT THEN 52:ROW=0: GOTO 52
  70 IF KEY# ASC("L") AND KEY#136 THEN 80:COL=COL-1: IF COL>=0 THEN 52:COL=
     WIDTH-1:ROW=ROW-1: IF ROW>=0 THEN 52:ROW=HEIGHT-1:COL=WIDTH-1: GOTO
     52
  80 IF KEY# ASC("U") THEN 90:ROW=ROW-1: IF ROW>=0 THEN 52:ROW=HEIGHT-1:
     COL=COL-1: IF COL>=0 THEN 52:COL=WIDTH-1: GOTO 52
  90 IF KEY# ASC("D") THEN 100:ROW=ROW+1: IF ROW<HEIGHT THEN 52:ROW=0:COL=
     COL+1: IF COL<WIDTH THEN 52:COL=0: GOTO 52
 100 IF KEY# ASC("+") THEN 110:VALUE=1: GOSUB 500: GOTO 52
 110 IF KEY# ASC("-") THEN 120:VALUE=0: GOSUB 500: GOTO 52
 120 VTAB 23: PRINT "INVALID KEY": FOR K=1 TO 25: NEXT K: VTAB 23: TAB 1
     : PRINT "               ": GOTO 52
 500 TEMP=PTTERN(COL)
 510 FOR B=0 TO 7:BITS(B)=TEMP MOD 2:TEMP=TEMP/2: NEXT B
 515 BITS(ROW)=VALUE
 517 TEMP=BITS(7)
 520 FOR B=6 TO 0 STEP -1
 530 TEMP=2*TEMP+BITS(B)
 540 NEXT B
 550 PTTERN(COL)=TEMP
 551 IF VALUE=0 THEN COLOR=0
 555 PLOT 15+COL,15+ROW
 557 COLOR=15
 560 RETURN
1000 FOR I=0 TO 7: PRINT PTTERN(I);" ";: NEXT I
1010 RETURN
1500 INPUT "WHICH KEY?"
1505 KEY= PEEK (KBD): IF KEY<128 THEN 1505
1510 POKE CLR,0:OFFSET=(KEY-128)*WIDTH
1512 POKE 2048,WIDTH: POKE 2049,HEIGHT
1515 POKE 60,(ADDR+OFFSET) MOD 256
1520 POKE 61,(ADDR+OFFSET)/256
1522 GR
1525 POKE 36,15: POKE 37,15
1530 COLOR=15: CALL 2058
1532 POKE 36,0: POKE 37,23
1535 COLOR=1: HLIN 14,14+WIDTH+1 AT 14: HLIN 14,14+WIDTH+1 AT 14+HEIGHT+
     1
1540 VLIN 14,14+HEIGHT+1 AT 14: VLIN 14,14+HEIGHT+1 AT 14+WIDTH+1
1545 FOR I=0 TO WIDTH-1
1550 PTTERN(I)= PEEK (ADDR+OFFSET+I)
1555 NEXT I
1560 GOTO 52
2000 PRINT "WHICH KEY?"
2001 KEY= PEEK (KBD): IF KEY<128 THEN 2001
```

```
2002 POKE CLR,0:KEY=KEY-128:OFFSET=KEY*WIDTH
2005 FOR I=0 TO WIDTH-1
2010 POKE ADDR+OFFSET+I,PTTERN(I)
2020 NEXT I
2030 RETURN
2500 REM  HELP SUBROUTINE
2501 REM
2510 TEXT : CALL -936
2515 VTAB 2: TAB 2: PRINT "COMMAND";: TAB 12: PRINT "EFFECT"
2520 TAB 2: PRINT "=======";: TAB 12: PRINT "======="
2525 VTAB 5: TAB 2: PRINT "PATTERN";: TAB 12: PRINT "STARTS A NEW PATTERN"

2526 PRINT
2527 TAB 2: PRINT "MODIFY";: TAB 12: PRINT "CALLS UP AN OLD PATTERN FOR"

2529 TAB 12: PRINT "MODIFICATIONS."
2530 PRINT
2531 TAB 2: PRINT "RECORD";: TAB 12: PRINT "SAVES CURRENT PATTERN IN THE"

2533 TAB 12: PRINT "PATTERN TABLE. IT WILL BE"
2535 TAB 12: PRINT "ASSOCIATED WITH A KEY."
2536 PRINT
2537 TAB 2: PRINT "SAME";: TAB 12: PRINT "RETURNS TO PATTERN AREA"
2539 TAB 12: PRINT "WITHOUT DESTROYING THE"
2541 TAB 12: PRINT "CURRENT PATTERN."
2542 PRINT
2543 TAB 2: PRINT "HELP";: TAB 12: PRINT "DISPLAYS THIS MESSAGE."
2585 PRINT : TAB 2: PRINT "  TO QUIT, TYPE ANY OF THE FOLLOWING:"
2587 TAB 2: PRINT "   'QUIT','Q','STOP','BYE', OR 'EXIT'"
2590 GOSUB WAIT
2599 RETURN
3000 REM  WAIT SUBROUTINE
3001 REM
3005 POKE CLR,0
3010 KEY= PEEK (KBD): IF KEY<128 THEN 3010
3015 POKE CLR,0
3020 IF KEY# ASC("Q") THEN RETURN
3025 TEXT : CALL -936: END
10000 TEXT : CALL -936
10005 KBD=-16384:CLR=-16368:WAIT=3000
10007 FOR I=0 TO 7:PTTERN(I)=BITS(I)=0: NEXT I
10010 INPUT "HEIGHT OF PATTERNS ",HEIGHT
10011 IF HEIGHT<9 THEN GOTO 10013
10012 HEIGHT=8: VTAB 23: PRINT "DEFAULTING TO HEIGHT = 8     ";
10013 TAB 1: VTAB 3
10015 INPUT "WIDTH OF PATTERNS ",WIDTH
10016 IF WIDTH<9 THEN GOTO 10018
10017 WIDTH=8: VTAB 23: PRINT "DEFAULTING TO WIDTH = 8      ";
10018 TAB 1: VTAB 5
10020 INPUT "TABLE ADDRESS IN DECIMAL ",ADDR
10025 CALL -958
10030 RETURN
```

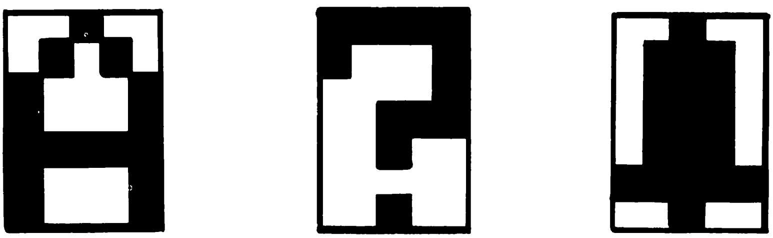
Part 3

Now I'll discuss the use of the machine-language driver program and the pattern-maker program in the creation of "animations" for the low-resolution screen. The major example considered is a program for converting the lo-res screen into a terminal that displays "giant" letters and other patterns. (Note: the information displayed is *not* passed on as commands to BASIC, although with some effort that could be accomplished.)

Giant Letters — The Patterns

The first step in creating any Apple Bits application is to design a set of patterns. In this case, the patterns will be letters and other characters that can be plotted on the screen when their associated keys are struck. The pattern size that works with the Integer BASIC program presented is 5 × 7. By suitable modifications to that program (left as an exercise to the reader), other character pattern sizes can be used as well.

To design your character set, run the pattern-maker program. Following the instructions given in Part 2, create patterns for each character on the Apple keyboard. You can also create patterns for keys which do not produce displayable graphics (control keys). The pattern maker will accept control keys as well as normal keys. For example, for the keys "A," "?," and " G" (Control-G), you might use the following:



When you are satisfied with your results, stop the pattern maker by typing "Q" or "QUIT" and then BSAVE your patterns. This takes a little calculating. Suppose your pattern table was started at location 3072 (decimal, or $C00 hex) and the patterns are, of course, 5 × 7 in size. To store the patterns for the characters Control-A through Z, you would consume 5 × 96, or 480 bytes. Thus,

```
BSAVE LETTERS,A3072,L480
```

would do. I normally just reserve all the space from $C00 to $FFF for patterns; that is more than enough, even for 96 patterns of 8 × 8 characters. I simply use the command:

```
BSAVE LETTERS,A$C00,L$3FF
```

Once you have created your patterns, the program to "drive" the screen is shown in listing 1. Don't forget to set LOMEM:

```
> LOMEM: 4096
```

There are some generally useful points to note in this program. You may be able to make use of them in other programs of your own.

In lines 10 and 15:

```
10 GR : POKE -16302,0 : COLOR=0

15 FOR I=40 TO 47 : HLIN 0,39 AT I : NEXT I
```

The POKE statement selects FULL SCREEN graphics. This causes any information already displayed on the bottom four lines of the screen to suddenly change to "living color." Line 15 blackens the bottom four lines again.

In line 12:

```
12 POKE 32,0 : POKE 33,40 : POKE 34,0 : POKE 35,24
```

These statements set the "text window" back to the full screen. But why do that? This is a graphics program, but it is also a text program as well: the letters are just a bit larger than usual! So when your screen fills with maxi-alphabetics, how do you make room for more? Scroll! Look at line 60:

```
60 FOR J=1 TO 4 : CALL -912 : COLOR=0 : HLIN 0,39 AT 47 : NEXT J
```

The routine at −912 is the normal monitor routine for text scrolling. It uses the settings of the window variables in locations 32 - 35 to determine what portion of the screen to scroll. The GR statement sets these variables so that only the bottom four lines will scroll. Our POKEs in line 12 have fooled the monitor into thinking that the whole screen should be scrolled. The Apple will then scroll the graphics display, without a whimper. Since the lines which appear at the bottom during the scrolling process will be WHITE, we use the HLIN statement to re-blacken them.

If you study the listing further, you will discover that the left and right arrow keys will function in a manner similar to their normal text interpretation. In addition, the ENTER key will cause the display to proceed to the beginning of the next "line." The ESC key functions as a "Clear Screen" key. It also causes the next character to appear at the upper left-hand corner of the display. I leave it to you to dig out the details of these points.

A Random Walk

Listing 2 presents an animation. It causes a little "man" to walk across the screen from the lower right corner to the upper left corner. The actual path taken is different each time, consisting of a random pattern of moves to the left and/or up.

The data for the patterns of program 2 is presented in listing 3.

Computer Choo-Choo

Listing 4 moves a locomotive across the screen from right to left. The train gives off "smoke" as it goes and periodically toots its whistle. The whistle is produced by calling a routine in the Apple Programmer's Aid ROM. If you do not have this installed in your Apple, you will have to locate and remove the CALL statements in the program. They could be replaced by CALLs to your own tone-producing routine.

The data for the locomotive program is presented in listing 5.

Notes on Implementing Animations

In both the random walk program and the locomotive program, only a small number of patterns was needed. Notice that the pattern selected for display by the programs at any given time is specified by a small positive number. For example, examine lines 535 to 540 of listing 2. The patterns are associated with these numbers because of the pattern-maker program. The control keys correspond to numbers 1 through 26. Thus, when you use the pattern maker to create a set of patterns and record a particular one using, say, Control-E, then that pattern becomes the 5th pattern in the table.

To set up the address of this pattern (so the machine-language driver knows which one to display), the statements in lines 536 and 537 of listing 2 would be used. These are similar to the statements appearing in lines 60 and 65 of the Fireworks Animation presented in Part 1.

Let's review the general form of the set-up instructions:

```
POKE 60, (TABLE + OFFSET)MOD 256

POKE 61,(TABLE + OFFSET)/256
```

where,

TABLE — represents the address in Apple II RAM of the very beginning of the Pattern Table. In all of our examples this has been 3072, decimal. However, it could be other values as well.

Note: The numbering of the entries in the table actually begins at 0. The 0th entry is inaccessible, since the pattern maker cannot accept a key whose character code is 0. Also, the entry in the table which corresponds to the Control-C key (number 3) will always contain "garbage." This is the reason for the IF test in line 535 of listing 2.

OFFSET —represents the distance (in bytes) from the beginning of the pattern table at which a given pattern may be found. This offset may be calculated using the formula:

OFFSET = WIDTH * KEY

where,

WIDTH — is the width of the patterns in the table.

KEY — is the number of the pattern you wish to retrieve.

Listing 1

```
   0 REM ***************************
   1 REM *                         *
   2 REM *       LARGE DRIVER      *
   3 REM *        R. C. VILE       *
   4 REM *                         *
   5 REM *    COPYRIGHT (C) 1982   *
   6 REM *      MICRO INK, INC.    *
   7 REM *  CHELMSFORD, MA. 01824  *
   8 REM *   ALL RIGHTS RESERVED   *
   9 REM ***************************
  12 GOSUB 1000
  15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
  20 ROW=0:COL=0
  22 COLOR= RND (15)+1
  25 GOSUB 700
  30 POKE 36,COL: POKE 37,ROW
  35 POKE 60,(3072+5*K1) MOD 256
  40 POKE 61,(3072+5*K1)/256
  42 COLOR= RND (15)+1
  45 CALL 2058
  50 COL=COL+6: IF COL<36 THEN 25
  55 COL=0:ROW=ROW+8: IF ROW<=40 THEN 25
  60 FOR J=1 TO 4: CALL -912: COLOR=0: HLIN 0,39 AT 46: HLIN 0,39 AT 47:
     NEXT J
  65 COLOR= RND (15)+1
  70 ROW=40:COL=0: GOTO 25
 700 KEY= PEEK (KBD): IF KEY<128 THEN 700
 705 POKE CLR,0
 710 K1=KEY-128
 712 IF K1#27 THEN 718
 713 COLOR=0: FOR I=0 TO 47: HLIN 0,39 AT I: NEXT I: COLOR= RND (15)+1
 715 ROW=0:COL=0: GOTO 700
 718 IF K1=13 THEN 785
 719 IF K1=7 THEN 775
 720 IF (K1#8 AND K1#21) THEN RETURN
 722 IF K1#21 THEN 725
 723 K1=32: RETURN
 725 COL=COL-6: IF COL>=0 THEN 750
 730 COL=30:ROW=ROW-8: IF ROW>=0 THEN 750
 735 ROW=0:COL=0
 750 COLOR=0
 755 FOR J=0 TO 7
 760 HLIN COL,COL+5 AT ROW+J
 765 NEXT J
 770 COLOR= RND (15)+1: GOTO 700
 775 PRINT "";: RETURN
 785 ROW=ROW+8: IF ROW>=48 THEN 790
 787 COL=0: GOTO 700
 790 COLOR=0
 792 FOR J=1 TO 4: CALL -912
 793 HLIN 0,39 AT 46: HLIN 0,39 AT 47
 794 NEXT J
 799 ROW=40:COL=0: COLOR= RND (15)+1: GOTO 700
1000 KBD=-16384:CLR=-16368
1009 POKE 2048,5: POKE 2049,7
1010 GR : POKE -16302,0: COLOR=0
1015 POKE 32,0: POKE 33,40: POKE 34,0: POKE 35,24
1016 RETURN
```

Listing 2

```
  0 REM ***************************
  1 REM *                         *
  2 REM *       RANDOM WALK       *
  3 REM *       R. C. VILE        *
  4 REM *                         *
  5 REM *   COPYRIGHT (C) 1982    *
  6 REM *     MICRO INK, INC.     *
  7 REM *  CHELMSFORD, MA. 01824  *
  8 REM *   ALL RIGHTS RESERVED   *
  9 REM ***************************
 10 MOVE=500: GR : POKE -16302,0: COLOR=0
 15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
 21 POKE 2048,8: POKE 2049,8
 32 POKE 36, RND (32): POKE 37, RND (40)
 35 COLOR= RND (15)+1
 40 D= RND (2)
 45 IF D#0 THEN 55
 50 DX=0:DY=-1: GOSUB MOVE: GOTO 35
 55 IF D#1 THEN 65
 60 DX=-1:DY=0: GOSUB MOVE: GOTO 35
 65 IF D#2 THEN 75
 70 DX=1:DY=0: GOSUB MOVE: GOTO 35
 75 DX=-1:DY=0: GOSUB MOVE: GOTO 35
500 COL= PEEK (36):ROW= PEEK (37)
505 COL=COL+DX: IF COL<32 THEN 510: GOSUB 600:COL=0
510 IF COL>0 THEN 515: GOSUB 600:COL=32
515 ROW=ROW+DY: IF ROW<40 THEN 520: GOSUB 600:ROW=0
520 IF ROW>0 THEN 530: GOSUB 600:ROW=40
530 POKE 36,COL: POKE 37,ROW
535 KEY= RND (5)+1: IF KEY=3 THEN 535
536 POKE 61,(3072+8*KEY)/256
537 POKE 60,(3072+8*KEY) MOD 256
540 CALL 2058
545 FOR TIME=1 TO 25: NEXT TIME
555 COLOR=0
560 HLIN COL,COL+7 AT ROW+7
562 VLIN ROW,ROW+7 AT COL+7
570 RETURN
600 COLOR=0: FOR I=0 TO 7: HLIN COL,COL+7 AT ROW+I: NEXT I
610 RETURN
```

Listing 3

```
0C00- FF FF FF 15 1F 7E 7C 78
0C08- 84 48 2B 3F 4B 88 10 00
0C10- 00 98 4B 3F 2B 48 84 00
0C18- 48 77 41 5D 41 77 78 3C
0C20- 00 98 CB 3F 6B C8 04 00
0C28- 00 10 88 6B 1F 2B CC 80
0C30- 5D 7F 08 1C 2A 49 08 01
0C38- 0F 08 78 40 6C 64 7C 64
0C40- 6C 78 48 7F 09 0F 7F 41
0C48- 49 41 7F 59 49 6B 49 4D
0C50- 7F 49 6B 49 7F 7F 49 7F
0C58- 49 7F 77 41 77 41 77 7F
0C60- 49 00 49 7F 22 55 49 55
0C68- 22 10 18 1C 18 10 41 63
0C70- 77 63 41 7F 3E 1C 08 00
0C78- 00 08 1C 3E 7F 08 1C 3E
*
```

Listing 4

```
 0 REM ***************************
 1 REM *                         *
 2 REM *       LOCOMOTIVE        *
 3 REM *       R. C. VILE        *
 4 REM *                         *
 5 REM *   COPYRIGHT (C) 1982    *
 6 REM *     MICRO INK, INC.     *
 7 REM *  CHELMSFORD, MA. 01824  *
 8 REM *   ALL RIGHTS RESERVED   *
 9 REM ***************************
10 MUSIC=-10473: POKE 767,40: POKE 766,30: POKE 765,32:MOVE=500:SMOKE=
   22
11 GR : POKE -16302,0: COLOR=0
15 FOR I=40 TO 47: HLIN 0,39 AT I: NEXT I
21 POKE 2048,8: POKE 2049,8
32 POKE 36,20: POKE 37,24
33 CC= RND (15)+1
35 COLOR=CC
40 D=1
50 DX=-1:DY=0: GOSUB MOVE
55 GOTO 35
500 COL= PEEK (36):ROW= PEEK (37)
505 COL=COL+DX: IF COL<32 THEN 510: GOSUB 600:COL=0
510 IF COL>0 THEN 515: GOSUB 600:COL=32:CC= RND (15)+1
515 REM
530 POKE 36,COL: POKE 37,ROW
535 KEY=1
536 POKE 61,(3072+8*KEY)/256
537 POKE 60,(3072+8*KEY) MOD 256
540 CALL 2058
542 GOSUB 800
545 FOR TIME=1 TO 25: NEXT TIME
550 IF RND (25)=0 THEN GOSUB 700
555 COLOR=0
560 HLIN COL,COL+7 AT ROW+7
562 VLIN ROW,ROW+7 AT COL+7
570 RETURN
600 COLOR=0: FOR I=0 TO 7: HLIN COL,COL+7 AT ROW+I: NEXT I
610 RETURN
700 CALL MUSIC: REM  *****REPLACE WITH RETURN IF YOU DO NOT HAVE THE PROGRAM
    MER'S AID ROM
705 POKE 766,100: FOR I=1 TO 50: NEXT I
710 CALL MUSIC: POKE 766,30: RETURN
800 PLOT COL+1,SMOKE
810 COLOR=0: PLOT COL+2,SMOKE+1
815 IF SMOKE=22 THEN PLOT COL+2,1
818 IF COL=32 THEN PLOT 2,SMOKE+1
820 SMOKE=SMOKE-1
830 IF SMOKE=0 THEN SMOKE=22
840 RETURN
```

Listing 5

```
0C00- FF FF FF 15 1F 7E 7C 78
0C08- FC BF FC 3C FF B9 F9 1F
0C10- 7D FD F0 78 70 FE F2 3E
0C18- 48 77 41 5D 41 77 78 3C
*
```

Numerical Data for Fireworks

```
0C00- FF FF FF 15 1F 15 F5 00
0C08- 00 00 08 00 00 00 00 00
0C10- 14 00 14 00 00 00 22 00
0C18- 00 00 22 00 41 00 00 00
0C20- 00 00 41 00 00 14 08 14
0C28- 00 00 00 22 14 00 14 22
0C30- 00 41 22 00 00 00 22 41
0C38- 00 22 14 08 14 22 00 41
0C40- 22 14 00 14 22 41 41 22
0C48- 14 08 14 22 41 00 00 00
0C50- 08 00 00 00 00 00 08 14
0C58- 08 00 00 00 08 00 22 00
0C60- 08 00 08 00 00 41 00 00
0C68- 08 00 00 08 1C 08 00 00
0C70- 00 08 08 36 08 08 00 08
0C78- 08 00 63 00 08 08 00 08
0C80- 08 3E 08 08 00 08 08 08
0C88- 77 08 08 08 08 08 08 7F
0C90- 08 08 08 12 1F 10 19 15
0C98- 12 11 15 0A 06 1F 04 17
0CA0- 15 09 1F 15 1D 19 05 03
0CA8- 0A 15 0A 17 15 1F 00 0A
0CB0- 00 10 1A 00 FF FF FF 0A
0CB8- 0A 0A FF FF FF 01 15 07
0CC0- FF FF FF 1F 05 1F 1F 15
0CC8- 0A 1F 11 11 1F 11 0E 1F
0CD0- 15 11 1F 05 01 1F 11 19
0CD8- 1F 04 1F 11 1F 11 18 11
0CE0- 1F 1F 06 19 1F 10 10 1F
0CE8- 02 1F 1F 0E 1F 1F 11 1F
0CF0- 1F 05 07 1F 11 17 1F 05
0CF8- 1A 17 15 1D 01 1F 01 1F
0D00- 10 1F 0F 10 0F 1F 08 1F
0D08- 1B 04 1B 03 1C 03 19 15
0D10- 13 FF FF FF FF FF FF 00
0D18- 11 1F FF FF FF FF FF FF
0D20- FF FF FF FF FF FF FF FF
*
```

5
TUTORIAL/REFERENCE

Tutorial/Reference

Any computer can be made easier to program and use if there is information available that explains how it works, or concise and complete documentation. We had these points in mind when we prepared this chapter.

''Apple Byte Table'' by Kim Woodward is a handy reference to byte-values within your system. If you ever need to read a monitor dump, this table will help you decipher the meaning of all those hexadecimal codes. You'll want to keep this table around for other duties, such as converting hex to decimal, or hex to binary.

''How Microsoft BASIC Works'' by Greg Paris explains how the various versions of Microsoft BASIC deal with variable storage. With this knowledge you are able to make a BASIC program more efficient through wise use of variables, or pass variables to assembly language subroutines, etc.

Apple Byte Table

by Kim G. Woodward

This useful reference table will simplify the task of decoding byte-values in the Apple's memory. For all numerical values, hex or decimal, each possible meaning is listed, ranging from ASCII to Applesoft token. If you ever tackle a hex dump, the Apple byte table will prove invaluable.

If you look at a single byte in the Apple or any other 8-bit microcomputer, it will mean different things at different times. Data and instructions are represented in the same manner in the computer: one byte may be data, an address, a token, or a command. I have put together a simple table which will be helpful no matter what the relationship is between the byte and your software. (Columns F, G, H, and I will be especially useful to the Apple owner.) The table is composed of 10 columns which represent:

A. The equivalent decimal value of the byte (assuming the byte is not signed).

B. The equivalent hex value of the byte.

C. The equivalent binary value of the byte (very useful for assembly language masking).

D. The value of the byte if it is looked at as the high byte of an address.

E. The corresponding ASCII character for the byte (if there is one).

F. The equivalent displayed screen character. (*I*-Inverse, *F*-Flashing, *N*-Normal.)

G. The equivalent key to be pressed to get the byte. (If there is one, note all keys > $7F. *C* after character means CTRL key held down.)

H. The corresponding Integer BASIC token for the byte. The Integer BASIC tokens can be found by keying:

```
> CALL -155        Go to monitor
* CA:00 10         Set program start
* 4C:14 10         Set program end
* 1000:13          Set length byte
* 1001:0A 00       Set line number
* 1003:            16 bytes of your choice
* 1013:01          End of line token
*                  Return via CTRL-C
>LIST
```

I. The corresponding Applesoft BASIC token for the byte. The Applesoft tokens can be found by keying:

```
  CALL -155        Go to monitor
* 67:01 08         Set program start
* AF:16 08         Set program end
* 801:16 08        Pointer to next line
* 803:0A 00        Set line number
* 805:             16 bytes of your choice
* 815:00           End of line token
* 816:00 00 00     End of program pointer
* 0G               Back to BASIC
  LIST
```

J. The corresponding 6502 machine language opcode.

Let's note some of the subtleties in the table's usage. First of all, if a particular pattern for a mask operation is needed, then it is a simple matter of looking down the table until the correct binary (column 3) pattern is found. Then on the same line, read the decimal equivalent for a POKE command, or the hex equivalent for assembly language use. In a similar manner you can do the following:

A. Decimal to hexadecimal conversion — scan the table in column 4 to find the highest number not exceeding the decimal number. If the number is negative (such as addresses in Integer BASIC larger than 32767), add 65536 before the conversion. Write down the hex value and subtract the decimal number just found. Then find the decimal remainder in the table and write down the hex value for it. The first hex value is the high byte and the second is the low byte. For example, find the hex equivalent of −936 (clear).

   −936 + 65536 = 64600 : the number to find. Find 64512 ($FC) : highest number less than 64600 -64512 = 88 : find difference. Find 88 ($58) : remainder. Value of −936 decimal is $FC58.

B. Hexadecimal to decimal conversion — separate the hex number into two bytes. Scan the table for the value of the high order byte in column 4. Then scan the table for the value of the low order byte in column 1, add the two numbers together and get the result. For negative addresses (> $7FFF) simply subtract 65536 from the number.

C. Relative addressing — the formula for relative addressing on the 6502 is: address of branch to address - address of branch inst. - 2. For example, to branch from location $345 to $313 you could find the decimal equivalent of $345 as per (A) above, 837, and of $313, 787. Thus 787 -837 - 2 is –52. Add 256 to –52 giving 204. Look up 204 in the table as $CC. $CC is then the relative address offset.

Columns F and G in the table can be found in the *Apple Reference Handbook* by Apple Computer, Inc.

The Apple Byte Table

Dec	Hx	Binary	High	Asc	Sc	Ky	Int Bs	Aps Bs	6502:
000	00	00000000	0	NUL	@I		HIMEM:	NUL	BRK
001	01	00000001	256	SOH	AI		EOS	SOH	ORAIX
002	02	00000010	512	STX	BI		_	STX	
003	03	00000011	768	ETX	CI		:	ETX	
004	04	00000100	1024	EOT	DI		LOAD	EOT	
005	05	00000101	1280	ENQ	EI		SAVE	ENQ	ORAZ
006	06	00000110	1536	ACK	FI		CON	ACK	ASLZ
007	07	00000111	1792	BEL	GI		RUN	BEL	
008	08	00001000	2048	BS	HI		RUN	BS	PHP
009	09	00001001	2304	HT	II		DEL	HT	ORAIM
010	0A	00001010	2560	LF	JI		,	LF	ASLA
011	0B	00001011	2816	VT	KI		NEW	VT	
012	0C	00001100	3072	FF	LI		CLR	FF	
013	0D	00001101	3328	CR	MI		AUTO	CR	ORA
014	0E	00001110	3584	SO	NI		,	SO	ASL
015	0F	00001111	3840	SI	OI		MAN	SI	
016	10	00010000	4096	DLE	PI		HIMEM:	DLE	BPL
017	11	00010001	4352	DC1	QI		LOMEM:	DC1	ORAIY
018	12	00010010	4608	DC2	RI		+	DC2	
019	13	00010011	4864	DC3	SI		-	DC3	
020	14	00010100	5120	DC4	TI		*	DC4	
021	15	00010101	5376	NAK	UI		/	NAK	ORAZX
022	16	00010110	5632	SYN	VI		=	SYN	ASLZX
023	17	00010111	5888	ETB	WI		#	ETB	
024	18	00011000	6144	CAN	XI		>=	CAN	CLC
025	19	00011001	6400	EM	YI		>	EM	ORAY
026	1A	00011010	6656	SUB	ZI		<=	SUB	
027	1B	00011011	6912	ESC	[I		<>	ESC	
028	1C	00011100	7168	FS	\I		<	FS	
029	1D	00011101	7424	GS	]I		AND	GS	ORAX
030	1E	00011110	7680	RS	^I		OR	RS	ASLX
031	1F	00011111	7936	US	_I		MOD	US	
032	20	00100000	8192	SPC	I		^	SPC	JSR
033	21	00100001	8448	!	!I		+	!	ANDIX
034	22	00100010	8704	"	"I		(	"	
035	23	00100011	8960	#	#I		,	#	
036	24	00100100	9216	$	$I		THEN	$	BITZ
037	25	00100101	9472	%	%I		THEN	%	ANDZ
038	26	00100110	9728	&	&I		,	&	ROLZ
039	27	00100111	9984	'	'I		;	'	
040	28	00101000	10240	(	(I		"	(	PLP
041	29	00101001	10496	)	)I		"	)	ANDIM
042	2A	00101010	10752	*	*I		(	*	ROLA
043	2B	00101011	11008	+	+I		!	+	
044	2C	00101100	11264	,	,I		!	,	BIT
045	2D	00101101	11520	-	-I		(	-	AND
046	2E	00101110	11776	.	.I		PEEK	.	ROL
047	2F	00101111	12032	/	/I		RND	/	
048	30	00110000	12288	0	0I		SGN	0	BMI
049	31	00110001	12544	1	1I		ABS	1	ANDIY
050	32	00110010	12800	2	2I		PDL	2	
051	33	00110011	13056	3	3I		RNDX	3	
052	34	00110100	13312	4	4I		(	4	
053	35	00110101	13568	5	5I		+	5	ANDZX
054	36	00110110	13824	6	6I		-	6	ROLZX
055	37	00110111	14080	7	7I		NOT	7	
056	38	00111000	14336	8	8I		(	8	SEC
057	39	00111001	14592	9	9I		=	9	ANDY
058	3A	00111010	14848	:	:I		#	:	
059	3B	00111011	15104	;	;I		LEN(	;	
060	3C	00111100	15360	<	<I		ASC(	<	
061	3D	00111101	15616	=	=I		SCRN(	=	ANDX
062	3E	00111110	15872	>	>I		,	>	ROLX
063	3F	00111111	16128	?	?I		(	?	
064	40	01000000	16384	@	@F		$	@	RTI

Dec	Hx	Binary	High	Asc	Sc	Ky	Int Bs	Aps Bs	6502
065	41	01000001	16640	A	AF		$	A	EORIX
066	42	01000010	16896	B	BF		(	B	
067	43	01000011	17152	C	CF		,	C	
068	44	01000100	17408	D	DF		,	D	
069	45	01000101	17664	E	EF		;	E	EORZ
070	46	01000110	17920	F	FF		;	F	LSRZ
071	47	01000111	18176	G	GF		;	G	
072	48	01001000	18432	H	HF		,	H	PHA
073	49	01001001	18688	I	IF		,	I	EORIM
074	4A	01001010	18944	J	JF		,	J	LSRA
075	4B	01001011	19200	K	KF		TEXT	K	
076	4C	01001100	19456	L	LF		GR	L	JMP
077	4D	01001101	19712	M	MF		CALL	M	EOR
078	4E	01001110	19968	N	NF		DIM	N	LSR
079	4F	01001111	20224	O	OF		DIM	O	
080	50	01010000	20480	P	PF		TAB	P	BVC
081	51	01010001	20736	Q	QF		END	Q	EORIY
082	52	01010010	20992	R	RF		INPUT	R	
083	53	01010011	21248	S	SF		INPUT	S	
084	54	01010100	21504	T	TF		INPUT	T	
085	55	01010101	21760	U	UF		FOR	U	EORZX
086	56	01010110	22016	V	VF		=	V	LSRZX
087	57	01010111	22272	W	WF		TO	W	
088	58	01011000	22528	X	XF		STEP	X	CLI
089	59	01011001	22784	Y	YF		NEXT	Y	EORY
090	5A	01011010	23040	Z	ZF		,	Z	
091	5B	01011011	23296	[	[F		RETURN	[	
092	5C	01011100	23552	\	\F		GOSUB	\	
093	5D	01011101	23808	]	]F		REM	]	EORX
094	5E	01011110	24064	^	^F		LET	^	LSRX
095	5F	01011111	24320	_	_F		GOTO	_	
096	60	01100000	24576	'	F		IF		RTS
097	61	01100001	24832	a	!F		PRINT		ADCIX
098	62	01100010	25088	b	"F		PRINT		
099	63	01100011	25344	c	#F		PRINT		
100	64	01100100	25600	d	$F		POKE		
101	65	01100101	25856	e	%F		,		ADCZ
102	66	01100110	26112	f	&F		COLOR=		RORZ
103	67	01100111	26368	g	'F		PLOT		
104	68	01101000	26624	h	(F		,		PLA
105	69	01101001	26880	i	)F		HLIN		ADCIM
106	6A	01101010	27136	j	*F		,		RORA
107	6B	01101011	27392	k	+F		AT		
108	6C	01101100	27648	l	,F		VLIN		JMPI
109	6D	01101101	27904	m	-F		,		ADC
110	6E	01101110	28160	n	.F		AT		ROR
111	6F	01101111	28416	o	/F		VTAB		
112	70	01110000	28672	p	0F		=		BVS
113	71	01110001	28928	q	1F		=		ADCIY
114	72	01110010	29184	r	2F		)		
115	73	01110011	29440	s	3F		)		
116	74	01110100	29696	t	4F		LIST		
117	75	01110101	29952	u	5F		,		ADCZX
118	76	01110110	30208	v	6F		LIST		RORZX
119	77	01110111	30464	w	7F		POP		
120	78	01111000	30720	x	8F		NODSP		SEI
121	79	01111001	30976	y	9F		NODSP		ADCY
122	7A	01111010	31232	z	:F		NOTRACE		
123	7B	01111011	31488	{	;F		DSP		
124	7C	01111100	31744	|	<FF		DSP		
125	7D	01111101	32000	}	=F		TRACE		ADCX
126	7E	01111110	32256	~	>F		PR#		RORX
127	7F	01111111	32512	RUB	?F		IN#		
128	80	10000000	32768		@N		NUL	END	
129	81	10000001	33024		AN	AC	SOH	FOR	STAIX
130	82	10000010	33280		BN	BC	STX	NEXT	
131	83	10000011	33536		CN	CC	ETX	DATA	
132	84	10000100	33792		DN	DC	EOT	INPUT	STYZ
133	85	10000101	34048		EN	EC	ENQ	DEL	STAZ
134	86	10000110	34304		FN	FC	ACK	DIM	STXZ
135	87	10000111	34560		GN	GC	BEL	READ	

Dec	Hx	Binary	High	Asc	Sc	Ky	Int Bs	Aps Bs	6502
136	88	10001000	34816		HN	HC	BS	GR	DEY
137	89	10001001	35072		IN	IC	HT	TEXT	
138	8A	10001010	35328		JN	JC	LF	PR#	TXA
139	8B	10001011	35584		KN	KC	VT	IN#	
140	8C	10001100	35840		LN	LC	FF	CALL	STY
141	8D	10001101	36096		MN	MC	CR	PLOT	STA
142	8E	10001110	36352		NN	NC	SO	HLIN	STX
143	8F	10001111	36608		ON	OC	SI	VLIN	
144	90	10010000	36864		PN	PC	DLE	HGR2	BCC
145	91	10010001	37120		QN	QC	DC1	HGR	STAIY
146	92	10010010	37376		RN	RC	DC2	HCOLOR=	
147	93	10010011	37632		SN	SC	DC3	HPLOT	
148	94	10010100	37888		TN	TC	DC4	DRAW	STYZX
149	95	10010101	38144		UN	UC	NAK	XDRAW	STAZX
150	96	10010110	38400		VN	VC	SYN	HTAB	STXZY
151	97	10010111	38656		WN	WC	ETB	HOME	
152	98	10011000	38912		XN	XC	CAN	ROT=	TYA
153	99	10011001	39168		YN	YC	EM	SCALE=	STAY
154	9A	10011010	39424		ZN	ZC	SUB	SHLOAD	TXS
155	9B	10011011	39680		[N	ESC	ESC	TRACE	
156	9C	10011100	39936		\N		FSS	NOTRACE	
157	9D	10011101	40192		]N	MCU	GS	NORMAL	STAX
158	9E	10011110	40448		^N	^C	RS	INVERSE	
159	9F	10011111	40704		_N		US	FLASH	
160	A0	10100000	40960		N	SPC	SPC	COLOR=	LDYIM
161	A1	10100001	41216		!N	!	!	POP	LDAIX
162	A2	10100010	41472		"N	"	"	VTAB	LDXIM
163	A3	10100011	41728		#N	#	#	HIMEM:	
164	A4	10100100	41984		$N	$	$	LOMEM:	LDYZ
165	A5	10100101	42240		%N	%	%	ONERR	LDAZ
166	A6	10100110	42496		&N	&	&	RESUME	LDXZ
167	A7	10100111	42752		'N	'	'	RECALL	
168	A8	10101000	43008		(N	(	(	STORE	TAY
169	A9	10101001	43264		)N	)	)	SPEED=	LDAIM
170	AA	10101010	43520		*N	*	*	LET	TAX
171	AB	10101011	43776		+N	+	+	GOTO	
172	AC	10101100	44032		,N	,	,	RUN	LDY
173	AD	10101101	44288		-N	-	-	IF	LDA
174	AE	10101110	44544		.N	.	.	RESTORE	LDX
175	AF	10101111	44800		/N	/	/	&	
176	B0	10110000	45056		0N	0	0	GOSUB	BCS
177	B1	10110001	45312		1N	1	1	RETURN	LDAIY
178	B2	10110010	45568		2N	2	2	REM	
179	B3	10110011	45824		3N	3	3	STOP	
180	B4	101101000	46080		4N	4	4	ON	LDYZX
181	B5	10110101	46336		5N	5	5	WAIT	LDAZX
182	B6	10110110	46592		6N	6	6	LOAD	LDXZY
183	B7	10110111	46848		7N	7	7	SAVE	
184	B8	10111000	47104		8N	8	8	DEF	CLV
185	B9	10111001	47360		9N	9	9	POKE	LDAY
186	BA	10111010	47616		:N	:	:	PRINT	TSX
187	BB	10111011	47872		;N	;	;	CONT	
188	BC	10111100	48128		<N	<	<	LIST	LDYX
189	BD	10111101	48384		=N	=	=	CLEAR	LDAX
190	BE	10111110	48640		>N	>	>	GET	LDXY
191	BF	10111111	48896		?N	?	?	NEW	
192	C0	11000000	49152		@N	@	@	TAB(	CPYIM
193	C1	11000001	49408		AN	A	A	TO	CMPIX
194	C2	11000010	49664		BN	B	B	FN	
195	C3	11000011	49920		CN	C	C	SPC(	
196	C4	11000100	50176		DN	D	D	THEN	CPYZ
197	C5	11000101	50432		EN	E	E	AT	CMPZ
198	C6	11000110	50688		FN	F	F	NOT	DECZ
199	C7	11000111	50944		GN	G	G	STEP	
200	C8	11001000	51200		HN	H	H	+	INY
201	C9	11001001	51456		IN	I	I	-	CMPIM
202	CA	11001010	51712		JN	J	J	*	DEX
203	CB	11001011	51968		KN	K	K	/	
204	CC	11001100	52224		LN	L	L	^	CPY
205	CD	11001101	52480		MN	M	M	AND	CMP
206	CE	11001110	52736		NN	N	N	OR	DEC
207	CF	11001111	52992		ON	O	O	>	
208	D0	11010000	53248		PN	P	P	=	BNE

Dec	Hx	Binary	High	Asc	Sc	Ky	Int Bs	Aps Bs	6502
209	D1	11010001	53504		QN	Q	Q	<	CMPIY
210	D2	11010010	53760		RN	R	R	SGN	
211	D3	11010011	54016		SN	S	S	INT	
212	D4	11010100	54272		TN	T	T	ABS	
213	D5	11010101	54528		UN	U	U	USR	CMPZX
214	D6	11010110	54784		VN	V	V	FRE	DECZX
215	D7	11010111	55040		WN	W	W	SCRN(	
216	D8	11011000	55296		XN	X	X	PDL	CLD
217	D9	11011001	55552		YN	Y	Y	POS	CMPY
218	DA	11011010	55808		ZN	Z	Z	SQR	
219	DB	11011011	56064		[N		[	RND	
220	DC	11011100	56320		\N		\	LOG	
221	DD	11011101	56576		]N	MCU	]	EXP	CMPX
222	DE	11011110	56832		^N	^	^	COS	DECX
223	DF	11011111	57088		_N		_	SIN	
224	E0	11100000	57344		N			TAN	CPXIM
225	E1	11100001	57600		!N			ATN	SBCIX
226	E2	11100010	57856		"N			PEEK	
227	E3	11100011	58112		#N			LEN	
228	E4	11100100	58368		$N			STR$	CPXZ
229	E5	11100101	58624		%N			VAL	SBCZ
230	E6	11100110	58880		&N			ASC	INCZ
231	E7	11100111	59136		'N			CHR$	
232	E8	11101000	59392		(N			LEFT$	INX
233	E9	11101001	59648		)N			RIGHT$	SBCIM
234	EA	11101010	59904		*N			MID$	NOP
235	EB	11101011	60160		+N				
236	EC	11101100	60416		,N			SYNTAX	CPX
237	ED	11101101	60672		-N			RWO GSB	SBC
238	EE	11101110	60928		.N			OUT DTA	INC
239	EF	11101111	61184		/N			ILL QNT	
240	F0	11110000	61440		0N			OVERFLW	BEQ
241	F1	11110001	61696		1N			OUT MEM	SBCIY
242	F2	11110010	61952		2N			UNF STM	
243	F3	11110011	62208		3N			BD SUBS	
244	F4	11110100	62464		4N			RDM ARY	
245	F5	11110101	62720		5N			DIV ZER	SBCZX
246	F6	11110110	62976		6N			ILL DIR	INCZX
247	F7	11110111	63232		7N			TYP MIS	
248	F8	11111000	63488		8N			STR LNG	SED
249	F9	11111001	63744		9N			FRM CPX	SBCY
250	FA	11111010	64000		:N			CANTCNT	
251	FB	11111011	64256		;N			UNDFNC	
252	FC	11111100	64512		<N			ERROR	
253	FD	11111101	64768		=N			(	SBCX
254	FE	11111110	65024		>N			(	INCX
255	FF	11111111	65280		?N			(	

How Microsoft BASIC Works

by Greg Paris

What is a variable? How are variables manipulated? This article gives the answers to both of these questions and discusses the similarity of FNx definitions to variables as well.

All computer languages are, to some extent, symbolic in nature. This means that addresses, constants, and variables may be used throughout a program and manipulated by their labels, instead of using absolute or true values. Although the use of symbols is often merely convenient — as in assembler texts — in many circumstances the concept permits manipulations which otherwise would be impossible. Algebraic variables in BASIC or FORTRAN are just one important case. For these reasons, how a computer language defines and manipulates symbols is fundamental to the structure and operation of whatever interfaces between the user and the opcodes — an interpreter, compiler, etc.

The varieties of symbol types allowed in any language determine, to a great extent, the power of that language to solve certain programming problems. The inherent accuracy of mathematical calculations is another example where the format of variable storage is critical.

For these reasons, a logical first step in dissecting the operation of the BASIC interpreter is to find out how it defines its symbols, and how it stores them.

This article is organized as follows. First, I offer a few definitions. This will level out most readers' backgrounds, and obviously may be skipped if you know the jargon. Next I describe the actual formats of both numeric and string variables. Then I discuss how BASIC uses RAM. Finally, I combine all of the above to describe variable *storage* formats, and explain their coding.

Definitions

I caution the more advanced reader that I am not a software development engineer, and may not use the approved industry-standard terminology.

Legal Variable Name: The BASIC manual defines a legal variable name to be "any alphabetic character, and [it] may be followed by any alphanumeric character... Any alphanumeric characters after the first two are ignored." In addition, one cannot embed reserved words into the variable name (A$ and AAAAAAA are legal variable names; %A is not, and neither is AGOTO).

Variable: To the interpreter, a variable is anything that is not an array (no joke!). Any time you need to refer to only one number, or one string, or one whatever, it will be called a variable. For example, X1 is a floating-point (or FP) variable, X1% is an integer variable, and X1$ is a string variable. They are stored in different ways internally so the interpreter cannot be confused by these three identical variable symbols. *You* may be confused however, so use caution in such cases.

Array: An array is any group of variables which is referred to by a common legal variable name, followed by a list of subscripts — also called indices. The BASIC manual sometimes refers to arrays as "matrices." An array may contain either integer or FP numeric data or strings, but no more than one type per array. You are, in theory, allowed 255 subscripts; the real restriction is the line length which limits you to twenty or so. For example, DIM X1(2) allots space for a singly subscripted FP array, and has room for 3 numbers — X1(0), X1(1), and X1(2). Further, DIM X1%(20) allots space for an array of 21 integer variables, and DIM X1$(10,3) partitions space for a doubly subscripted array of 44 $[(10+1)\times(3+1)]$ different strings. (A technical note: if an array is not dimensioned before it is used, the interpreter will automatically execute a DIM command and thus assign each subscript the default value of 10.)

Header: I define a header as any information about a variable (how it is stored or referred to) that is stored along with the data to which it refers. For example, if the interpreter requires information about an array, including its size, how many subscripts, and the values of those subscripts, then the interpreter will group all this information, along with the variable name, into a header — the small block of "data" which immediately precedes the real data in the array. A header may be as short and simple as the 2 bytes of an encoded variable name, or as detailed as the example just given.

.WOR Address Format: When a 16-bit address is to be stored in an 8-bit machine, it can be stored first byte (MSB) first, second byte (LSB) second, or in the reverse order. In assembler notation, the MSB-first arrangement is often referred to as ".DBY" (for "Double BYte"), whereas the reversed order — LSB-first — is called ".WOR" order (for "WORd"). Almost all addresses handled by the BASIC interpreter are stored in .WOR format, including those that may be embedded in headers.

Numeric Variables

There are two types of numeric data allowed in BASIC: integer and floating-point (FP). An integer number is stored in two bytes, and can represent any integer between +32,767 and −32,768. An FP number is stored in 5 bytes (4 bytes on

OSI) and can represent numbers between $\pm 1.7 \times 10^{38}$ and $\pm 2.94 \times 10^{-39}$, and zero. This format for FP numbers allows at least 9 decimal digit accuracy at all times.

Since FP arithmetic as done by the BASIC interpreter is not germane, I will not detail its function in this article. Suffice it to say that there exists, in zero-page RAM, temporary storage areas for two FP numbers. The one most used is the floating-point accumulator (or FPA) and is located at the addresses shown in figure 1-A. The FPA is five to seven bytes long — the second byte of the FPA contains the sign of the mantissa, which is incorporated into the leftmost bit (MSB) of the mantissa whenever a number is removed from the FPA. (The use of this bit for the sign need not confuse you, since in the FPA this bit is *defined* as being set, unless the number equals zero. Therefore, if it will *always* be 1, then it can be ignored during storage and used for another purpose, namely, to store the sign of the mantissa compactly.) In addition, there is a byte (see figure 1-A) which actually extends the FPA mantissa by 8 bits. It is used internally in all arithmetic operations, but is rounded off and stripped whenever a variable is removed from the FPA. The first byte of the FPA is the exponent of the number plus $80. If the number equals zero, then this byte is zero.

Both types of variables, if referred to before being assigned a specific numeric value (i.e., if you use a previously undefined variable), will be filled with 0's — hence, the default value in each case is zero.

Figure 1-A: Locations of Floating-Point Accumulators.

Computer:	**AIM 65**	**Applesoft**	**OSI (BASIC-in-ROM)**	**Old PET (1.0)**	**New PET (2.0, 4.0)**
Length of FPA	6 bytes	7 bytes	5 bytes	6 bytes	6 bytes
Address of FPA	$00A9- $00AE	$009D- $00A3	$00AC- $00B0	$00B0- $00B5	$005E- $0063
FPA extension	$00B8	($00A3)	$00B2	$00B7	$0065

String Variables

The "value" of a string variable, and the information stored in a string variable (or array) in RAM, are two different things. The two items actually stored in the "variable" or "array" are a pointer (or a list of pointers) in .WOR format to the start of the string, and the length of the string. The string may be embedded in a program line, or stored in "top free space" (high RAM).

If the string is empty ("null"), then the byte for string length is set to zero, and although it will then be ignored, both bytes of the pointer are zeroed. The size of any string is limited to 255 characters because a single byte is used to indicate its length.

User Functions

DEF and FNx are BASIC program statements which allow a user to define a unique function. Each FNx is labeled by a legal variable name, and this is why I discuss this statement in an article on variables. As detailed later, the BASIC interpreter stores a reference to each function definition in a complex header, filed under the variable name which is assigned to it by the user.

How BASIC Uses RAM

A memory map of how BASIC partitions space for its various needs is shown in figure 1-B. "Top free space" may be a new term to some readers. When BASIC is commanded to operate on strings, it designates an area in unused memory as work space (from $UNUN to $TTTT – 1), and then stores the result of any operation in "top free space" (from $TTTT to $NONO – 1).

Also listed in figure 1-B are the zero-page locations which are reserved by BASIC to store pointers to various addresses which are used frequently. These pointers are initialized upon entry into BASIC, and are updated any time the program is changed or run. All pointers are stored in .WOR format.

***Figure 1-B:* BASIC Utility Pointers.**

Computer:	AIM 65	Apple	OSI (BASIC-in-ROM)	Old PET	New PET
Address of pointer to:					
Start of BASIC program (address:)	$0073 ($0212)	$0067 ($0801)	$0079 ($0301)	$007A ($0402)	$0028 ($0402)
Start of variable storage ($PPPP)	$0075	$0069	$007B	$007C	$002A
Start of array storage ($RRRR)	$0077	$006B	$007D	$007E	$002C
Start of free space ($UNUN)	$0079	$006D	$007F	$0080	$002E
Top (end) of free space ($TTTT)	$007B	$006F	$0081	$0082	$0030
Top of memory ($NONO)	$007F	$004C	$0085	$0086	$0034

How Variable Names are Encoded

BASIC reserves 2 bytes for the variable name (symbol). However, since the same name could refer either to an integer, FP variable, or a string, it must distinguish between them. It does this by setting or clearing, in various combinations, the otherwise unused leftmost bit (MSB) of each of the two bytes in the name. All four possible permutations are used. The interpreter performs this encoding during a RUN whenever a new variable name is encountered, and uses the format described in table 1. If a variable name is only a single character, then the second character space allotted to it is filled with 0's, except for the MSB, which is set or cleared as needed.

Storage Formats

Most of the details of variable format and variable name encoding have been described. All that remains is to put the information together and describe what is actually found in memory from $PPPP to $UNUN – 1.

Variables are stored together, but separate from the arrays. However, integer numeric, FP numeric, string, and FNx definition variables are all intermixed. Arrays are stored in the next higher allocated RAM, and are also intermixed. In both cases, the jumbled order is actually a function of when they are defined during the RUNning of a program. Each variable or array that is interpreted is assigned a space in the order in which it is encountered, with the variables and the arrays each shuttled off to their respective spaces.

There is a reason for separating variables from arrays. Each item stored as a variable takes up exactly 7 bytes. This makes searching for variables very easy, as the interpreter's variable pointer need only increment by 7 bytes to look for the next variable. Since arrays can vary greatly in size, this technique is not applicable, and scanning for individual array entries is somewhat more time consuming.

Table 1: Format for encoding different types of variable names.

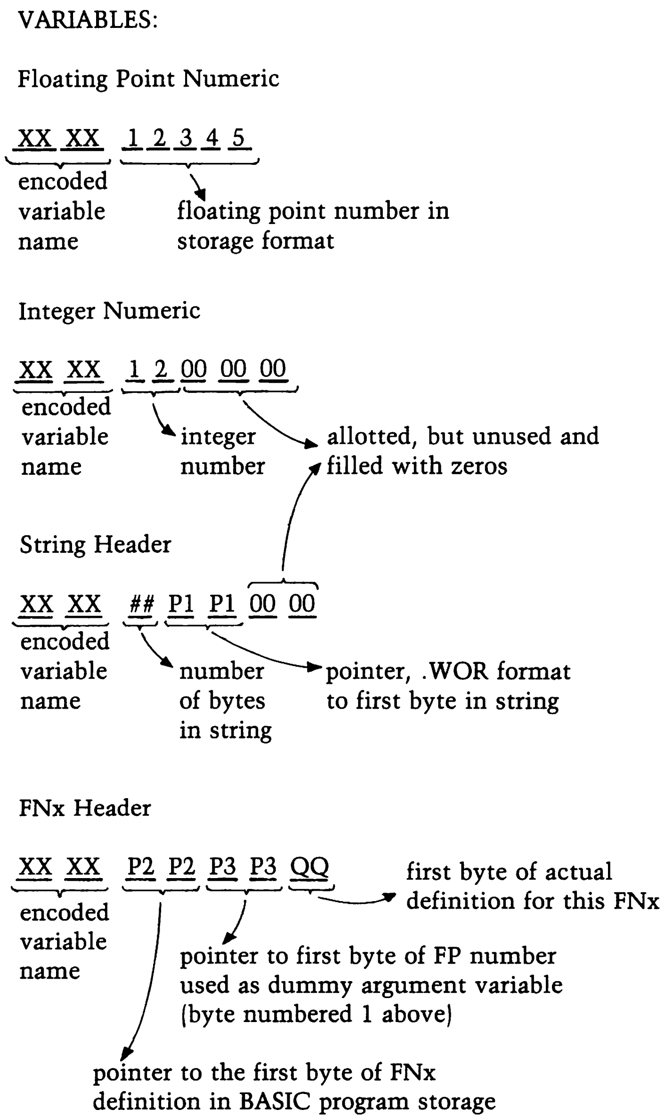
If the legal variable name is AC, then:

if the variable is	then the symbol is encoded as these two bytes:
a floating point numeric (no suffix)	$41, $43 (MSB each byte clear)
an integer numeric (suffix = %)	$C1, $C3 (MSB each byte set)
a string (suffix = $)	$41, $C3 (MSB first byte clear, MSB second byte set)
an FNx definition variable	$C1, $43 (MSB first byte set, MSB second byte clear)

Each time the program begins RUNning, it executes a CLEAR instruction, which erases any reference to any variables and arrays which may have previously been defined. This CLEAR instruction sets the pointers located at $0075, $0077, and $0079 (on the AIM) to the same value — the address of the last byte of program storage, plus one. Similarly, the pointer at $007B ("top free space") is set to equal the address in $007F (top usable memory + 1).

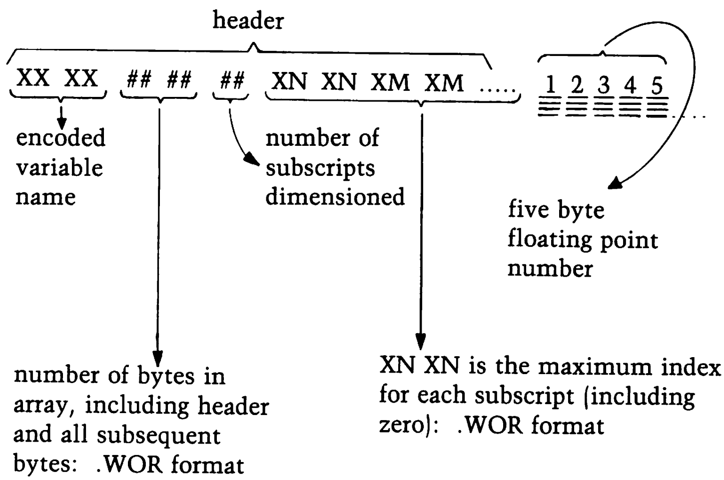
The headers for variables and arrays, and the formats in which they are stored in RAM, are shown in figure 2.

Figure 2: Variable and Array Storage Formats

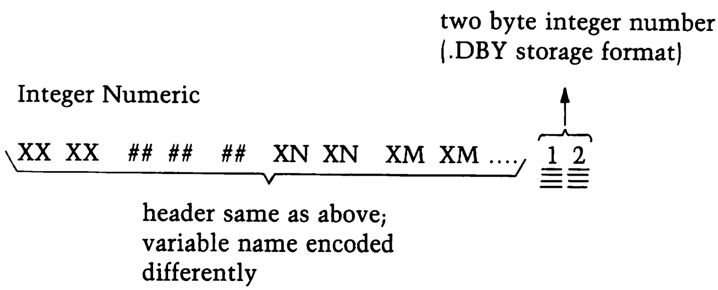


ARRAYS:

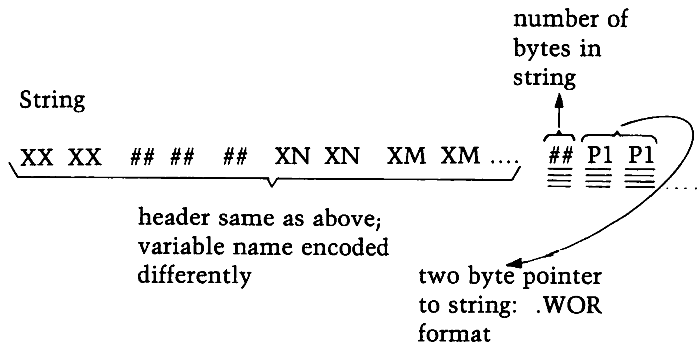
Floating Point Numeric



Integer Numeric



String



The definition of a header should be clearer now. In both types of numeric variables, the header is simply the 2 bytes of the encoded variable name. More complicated arrangements are seen in the FNx header and the various array headers.

Variables: For an FP variable, all 7 bytes are utilized. The last 5 bytes represent the FP number, in RAM storage form as described above.

An integer variable only uses 4 of the 7 bytes allotted to it. Use of integer variables in your program therefore wastes some space, but could save time during interpretation.

The string "variable" has a 5-byte header, made to fill 7 bytes by tacking a bunch of zeros on the end.

The FNx header is very interesting. It is filed as a variable because it is defined with a variable name. Any legal variable name may be used as its label. In addition, any legal variable name may be used as the dummy argument variable, even one used elsewhere in the program, because before the interpreter evaluates an FNx statement, it saves the value which was originally stored in the dummy variable on the stack. If the dummy variable is a new variable, it is automatically created, allotted 7 bytes of space after the FNx header, and appropriately labeled as an FP variable. The FNx header is set up whenever a DEF FNx is performed. If this particular FNx is later redefined, only the original header is changed. The last byte in the header might not be used by the interpreter; it seems to be there only to clear the stack completely during the DEF FNx operation.

Arrays: Not only do arrays have longer headers, but they also utilize space more efficiently. There is no minimum allotment of space, and consequently, no filler bytes are necessary. FNx arrays are not supported in this version of BASIC.

The headers for each type of array are essentially identical in format and content. The first two bytes are the encoded array name (see table 1). The next pair of bytes is a 16-bit number (.WOR format), the total number of bytes in the array. This includes the header with all its subscripts spelled out, and all the space allotted for the variables or string pointers. The fifth byte represents the number of subscripts used. The remainder of the header is a list of subscripts — a series of 16-bit numbers in .WOR format, one for each subscript — in an order that is the REVERSE of the listed order in the DIM statement.

The actual storage format of the array contents is much the same as for a single variable. Each member of an FP array is allotted five bytes for storage, and each member of an integer array is allotted two bytes. Therefore, in contrast to an integer variable, using integer *arrays* not only saves interpreting time but also a tremendous amount of space as well. Each entry in a string array is allotted three bytes, as before.

Within the array, individual members are ordered in straightforward fashion, but not as simply as you'd expect. Just as in the array header, the individual members of an array are in a "reversed" ascending sequence. For example, if the statement DIM A(2,4) has been executed, then the order of members in the array is A(0,0), A(1,0), A(2,0), A(0,1), A(1,1), A(2,1),..., A(1,4), A(2,4). By analogy, this can be extended to any number of subscripts.

An example is seen in figure 3. This program is intended only to demonstrate variable and array assignment. Note that all the pointers — FNQ and strings — point to the beginning of their respective referents. All the variables are ordered in the sequence in which they were interpreted; the arrays are similarly arranged in higher RAM. Note the encoded variable names for each assignment.

Summary

The following conclusions are of interest to anyone wishing to save execution time and/or memory space. 1) The use of an integer *variable* is generally a waste, for two reasons: the integer must be defined by a "%" each time it occurs (at the cost of 1 byte per occurrance), and, since it takes up 5 bytes anyway, even this doesn't save space. 2) An integer *array* really does save space, if it is of sufficient size. 3) You can save a few bytes, and shorten execution time slightly, by using as a dummy argument variable one that has already been used in the program. Its actual value will not be lost during the execution of an FNx.

These storage formats are not specific to one machine, and apply to those versions of Microsoft BASIC which are used on AIM, SYM, PET, OSI, Apple, etc.

Legend for Figure 3

A. Test program in BASIC.

B. Zero page pointers to partitions in RAM (see figure 1-a).

C. Dump of tokenized test program (partial).
Note that D$ is found at $025B, and the definition of FNQ at $0241.

D. Dump of variable and array storage.
Note that the order of space assignment is identical to the discovered order in the program.

E. Contents of "top free space", includes 'value' of E2$, found at $0FF1.

Figure 3

A.

```
10 DIM AA(2),B%(2,3)
20 AA=2:B%=17
30 DEF FNQ(X)=X*AA
40 C=5.7207
50 D$="A STRING"
60 DIM C(2)
70 F%=-24
80 E2$="IS NOT "+D$
90 STOP
```

B.

```
<M>=0073 12 02  BASIC PROGRAM STARTS AT $0212
< > 0075 98 02  VARIABLES START AT $0298
< > 0077 D0 02  ARRAYS START AT $02D0
< > 0079 1D 03  FREE SPACE STARTS AS $031D
< > 007B F1 0F  FREE SPACE ENDS AT $0FF1
< > 007F 00 10  TOP OF MEMORY IS $1000
```

C.

```
<M>=0212 26 02  NEXT LINE IS AT $0226
< > 0214 0A 00  THIS IS LINE 10
< > 0216 85 20  'DIM' TOKEN, SPACE
< > 0218 41 41  'AA'
< > 021A 28 32  '(2'
< > 021C 29 2C  '),'
< > 021E 42 25  'B%'
< > 0220 28 32  '(2'
< > 0222 2C 33  ',3'
< > 0224 29 00  ')', END OF LINE
< > 0226 35 02  NEXT LINE IS AT $0235
< > 0228 14 [illegible]  THIS IS LINE 20
< > 022A 41 41  'AA'
< > 022C AC 32  '=' TOKEN, '2'
< > 022E 3A 42  ':B'
< > 0230 25 AC  '%', '=' TOKEN
< > 0232 31 37  '17'
< > 0234 00     END OF LINE
< > 0235 46 02  NEXT LINE IS AT $0246
< > 0237 1E 00  THIS IS LINE 30
< > 0239 95 20  'DEF' TOKEN, SPACE
< > 023B 9F 51  'FN' TOKEN, 'Q'
< > 023D 28 58  '(X'
< > 023F 29 AC  ')', '=' TOKEN
< > 0241 58 A6  'X', '*' TOKEN
< > 0243 41 41  'AA'
< > 0245 00     END OF LINE
< > 0246 53 02  NEXT LINE IS AT $0253
< > 0248 28 00  THIS IS LINE 40
< > 024A 43 AC  'C', '=' TOKEN
< > 024C 35 2E  '5.'
< > 024E 37 32  '72'
< > 0250 30 37  '07'
< > 0252 00     END OF LINE
< > 0253 65 02  NEXT LINE IS AT $0265
< > 0255 32 00  THIS IS LINE 50
< > 0257 44 24  'D$'
< > 0259 AC 22  '=' TOKEN, '"'
< > 025B 41 20  'A '
< > 025D 53 54  'ST'
< > 025F 52 49  'RI'
< > 0261 4E 47  'NG'
< > 0263 22 00  '"', END OF LINE
```

D.

```
<M>=0298 41 41  FP VARIABLE 'AA'
< > 029A 82 00  VALUE IS 2
< > 029C 00 00
< > 029E 00
< > 029F C2 80  INTEGER VARIABLE 'B'
< > 02A1 00 11  VALUE IS 17
< > 02A3 00 00
< > 02A5 00
< > 02A6 D1 00  FN 'Q'
< > 02A8 41 02  DEFINED AT $0241
< > 02AA AF 02  DUMMY VARIABLE VALUE AT $02AF
< > 02AC 58
< > 02AD 58 00  FP VARIABLE 'X'
< > 02AF 00 00  VALUE IS 0
< > 02B1 00 00
< > 02B3 00
< > 02B4 43 00  FP VARIABLE 'C'
< > 02B6 83 37  VALUE IS 5.7207
< > 02B8 0F F9
< > 02BA 73
< > 02BB 44 80  STRING VARIABLE 'D' (D$)
< > 02BD 08     8 BYTES OF DATA
< > 02BE 5B 02  AT $025B
< > 02C0 00 00
< > 02C2 C6 80  INTEGER VARIABLE 'F' (F%)
< > 02C4 FF E8  VALUE IS -24
< > 02C6 00 00
< > 02C8 00
< > 02C9 45 B2  STRING VARIABLE '2' (E2$)
< > 02CB 0F     15 BYTES OF DATA
< > 02CC F1 0F  AT $0FF1
< > 02CE 00 00

< > 02D0 41 41  FP ARRAY 'AA'
< > 02D2 16 00  USES 22 BYTES
< > 02D4 01     1 SUBSCRIPT
< > 02D5 00 03  SUBSCRIPT = 2
< > 02D7 00 00  ARRAY ELEMENTS ARE ALL 0

< > 02E6 C2 80  INTEGER ARRAY 'B' (B%)
< > 02E8 21 00  USES 33 BYTES
< > 02EA 02     2 SUBSCRIPTS
< > 02EB 00 04  SUBSCRIPT 2 = 3
< > 02ED 00 03  SUBSCRIPT 1 = 2
< > 02EF 00 00  ARRAY ELEMENTS ARE ALL 0
```

E.

```
<M>=0FF1 49 53  'IS'
< > 0FF3 20 4E  ' N'
< > 0FF5 4F 54  'OT'
< > 0FF7 20 41  ' A'
< > 0FF9 20 53  ' S'
< > 0FFB 54 52  'TR'
< > 0FFD 49 4E  'IN'
< > 0FFF 47     'G'
```

6
RECREATION/APPLICATIONS

Recreational/Applications

For your entertainment we've included "Othello" by Charles Taylor, Jr., and "Musical Duets" by Rick Brown. Othello is a fascinating board game for two players that is easy to learn and takes a lifetime to master. Musical Duets is a programmable music player that, with the help of a cassette recorder, even plays in stereo. A rendition of "Blue Bells of Scotland" is included.

For enlightenment, there is "Solar System Simulation" by David Partyka, which displays the motions of the first six planets against a star background. With this simulation you can see how the sky looked in the past or how it will look in the future. An advantage the program has over sky and telescope is that it can be used any time. So, if you ever get the urge to gaze at the stars and planets on a cloudy day or during lunch hour, get your Apple and gaze away!

When you're not keeping your eyes on shooting stars, maybe you should be watching the rise and fall of your stocks. "A Simple Securities Manager" by Ronald Guest can help you keep track of dividends paid, appreciation, and the current status of your portfolio.

A Simple Securities Manager

by Ronald A. Guest

Manage your stocks more carefully in these volatile times! Use this simple program to record security transactions, keep track of gains and losses, and evaluate your holdings at any time.

One of the many uses of a home computer is for record keeping. And one of the most profitable types of record to keep is security transactions. It has become increasingly more important to have accurate information readily at hand; a small computer can be a big help.

I have written a program to assist in making decisions about my holdings. This program runs on a 32K Apple with ROM Applesoft and a Disk II. The output of the program is heavily oriented toward the standard 24 × 40 Apple display, but as you will see, it produces adequate results when used with a hardcopy printer. Three types of reports may be generated, and four types of operations may be performed on the securities data.

The stock manager program is tailored to fit my own needs, and others may require different reports or formats. I will try to provide sufficient information in this article to allow the program to be modified easily.

Reports

Three types of reports which may be requested are: a listing of the data in the current portfolio, a listing of the appreciation in the portfolio, and a (very) rough estimate of the dividends paid by the portfolio. In all three of the reports, the user may select that all securities be listed, that all unsold securities be listed, or that all sold securities be listed.

The LIst report outputs all of the information stored in the disk file for the selected class of holdings. The information printed includes the first five characters of the name, the purchase and sale dates, the purchase and sale prices, the per share dividend, and the number of shares (figure 1). Up to five holdings may be printed per page, and the totals of the purchase prices and sale prices will be printed on the final page. For an explanation of the meaning of the sale date and sale price for a security which has not yet been sold, see the paragraphs on adding an entry and on reading a data file.

The appreciation report lists the dollar and percent gains (losses) for each of the stocks listed. At the end of the report, the total dollar gain and the percent gain (loss) based on the purchase price are printed for the holdings selected (figure 2). If a security was sold 12 or more months after it was purchased, or if the security was purchased 12 or more months prior to the current date, then the name is displayed in inverse video indicating that the holding may be eligible for long-term gain.

A report of the dividends paid for the selected stocks provides an estimate of the dollar amount paid from the time the security was purchased to the time it was sold (or the current date if not yet sold). Only the selected securities with non-zero dividends are listed. The estimate is based on the number of months a security was held (figure 3). Since most securities pay dividends on specific dates, holdings which are quickly sold may show a dividend on the report, but have never been paid out. Since my investment goals are heavily oriented toward capital appreciation, the discrepancy does not bother me. People with different investment goals may wish to improve the estimates.

Operations on Data

The stock manager stores information in a sequential text file. A free format is used which allows each element to vary in length. The first element of the data file is a count of the number of entries in that file. The remainder of the file contains the entries. A security's entry, in the order of appearance, is: name, purchase date, sale date, purchase price, sale price, dividend, and number of shares.

When first run, the stock manager will have no entries, so the first command to execute is the ADd command. ADd requests the information which will be stored in the data file. All dates should be entered in the form MMDDYY with no slashes or other separators. The date must be six characters in length, so each field must be zero-filled. For instance, February 2, 1979 would be entered as 020279. When adding an entry for an as yet unsold security, enter a single blank for the sale date.

After adding all of the entries desired, a WRite command should be performed. WRite will prompt for a file name, and then output the entries to disk. Before any reports are generated, a REad command should be executed. The REad will ask for the file name and then read the data file. After closing the data file, REad will prompt for the current price of all holdings which have not yet been sold. This price is then used in generating reports. Note that the price entered should be the total price, not the per share price.

If an error is made adding an entry, or if a holding is sold, the data may be updated with the CHange command. CHange searches for the given name and then requests the new information. If a holding is to be deleted, enter an * for the

Figure 1

```
         ALL/NOTSOLD/SOLD ALL
     PRESS 'RETURN' WHEN READY
NAME  PDATE  SDATE   PPRICE SPRICE  DIV

CETRI 021379 082779  1517.3   875.5   0
 200

MBI   060179         2832.3  5124.3 3.5
 100

PLUMM 031479 071579  5786.8  8514.1   0
 200

TURKE 052278          827.3  1159.5  .8
 400

4M    120579          879.3   945.8 1.3
 150


TOTALS
          PPRICES 11843
          SPRICES 16619.1
     PRESS 'RETURN' WHEN READY
```

Figure 2

```
CURRENT DATE (MMDDYY) 033180
              ALL/NOTSOLD/SOLD ALL
     PRESS 'RETURN' WHEN READY
   NAME       $GAIN  %GAIN

CETRI         -641.8  -42.3
MBI          2291.95  80.92
PLUMM         2727.3  47.13
TURKE          332.2  40.15
4M             66.45   7.56

TOTALS    $GAIN 4776.1
          %GAIN 40

     PRESS 'RETURN' WHEN READY
```

Figure 3

```
CURRENT DATE (MMDDYY) 033180
              ALL/NOTSOLD/SOLD ALL
     PRESS 'RETURN' WHEN READY
   NAME       $GAIN   %GAIN

MBI            262.5       9
TURKE         586.67      71
4M             48.75       6

TOTALS    $GAIN 897.92
          %GAIN 20

     PRESS 'RETURN' WHEN READY
```

name. Be sure to do a WRite if the changes are to be permanent. If more than one entry in a portfolio has the same name (to the 25th character), the month purchased or some other difference should be introduced to allow a unique search. When the stock manager is EXited, it asks if the file should be updated. An answer of 'yes' will cause a WRite to be performed.

The stock manager was written to allow new commands or data fields to be added easily. To add a command, choose an unused entry in CMD$ (denoted by 'XX') and substitute the first two characters of the new command (lines 130-133). Between lines 330-399, output the command name and description for the menu. On line 510, change the entry in the GOSUB list corresponding to the index into CMD$ to the line number of the new command.

Adding a new data field is just as easy. Simply dimension the new field appropriately in lines 100-110. Then add a line in 36240-36280 to input the field, add a line in 38240-38255 to print the field, and add a line in 40110-40190 to enter the field into the data area. A list of the major variables and their usage is given in table 1 and a list of the subroutines is in table 2.

Users without a disk should change the REad routine to use BASIC READ and DATA statements. The WRite, CHange, and ADd routines can then be deleted since changes to the entries can be made by retyping the appropriate DATA statement. With these modifications, the program should easily run on a 16K cassette system (Applesoft in ROM).

See figure 4 for a sample of the displayed menu.

Figure 4

```
                STOCK MANAGER 1.0
                 BY R.A. GUEST
                      MENU
ADD HOLDING
APPRECIATION
CHANGE HOLDING
DIVIDENDS
LIST HOLDINGS
READ DATA FILE
WRITE DATA FILE
EXIT
          COMMAND: READ
FILE NAME TEMP
MBI
CURRENT PRICE 5124.25
TURKE
CURRENT PRICE 1159.50
4M
CURRENT PRICE 945.75
                      MENU
ADD HOLDING
APPRECIATION
CHANGE HOLDING
DIVIDENDS
LIST HOLDINGS
READ DATA FILE
WRITE DATA FILE
EXIT
          COMMAND:
```

Table 1: List of Variables.

ANS	Indicates what class of stocks to list All(0) / Notsold(1) / Sold(2)
CC	Index of last entry in CMD$
CD$	Current date
CMD$	Array of two character command names
COUNT	Number of holdings in current file
D$	Control-D for DOS
DG	Dollar gain
DV	Array of per share dividends
F$	File name containing stocks
INDEX	Index to stock holdings
LINE	Number of lines being displayed
MN	Number of months between sale (or current) date and purchase date
NM$	Array of stock names
PD$	Array of purchase dates
PP	Array of purchase prices
SD$	Array of sale dates (1 blank if not sold)
SH	Array of number of shares
SP	Array of sale prices
TPP	Total purchase prices
TSP	Total sale prices
TV	Same as TPP
YR	Number of years between sale (or current) date and purchase date

---

Table 2: Routines and Their Uses

20000-21999	Appreciation Report
24000-25999	Change an Entry
28000-29999	Estimated Dividends Report
32000-33999	List Securities Entries
36000-37999	Read Securities from Disk
38000-39999	Write Securities to Disk
40000-41999	Add a New Entry
50000-50500	Print Header for List of Securities
51000-51500	Wait for Return to be Pressed
52000-52500	Print Header for Appreciation and dividend

```
10  REM *************************
12  REM *                       *
14  REM *   STOCK HOLDINGS MGR  *
16  REM *      R. A. GUEST      *
18  REM *                       *
20  REM *  COPYRIGHT (C) 1982   *
22  REM *    MICRO INK, INC.    *
24  REM * CHELMSFORD, MA 01824  *
26  REM *  ALL RIGHTS RESERVED  *
28  REM *                       *
30  REM *************************
40  REM
50  REM
100  DIM NM$(25),PD$(25),SD$(25),PP(25),SP(25),DV(25)
101  DIM CMD$(10),SH(25)
120  REM  ** INIT COMMAND STRINGS **
130 CMD$(0) = "AP":CMD$(1) = "EX":CMD$(2) = "CH"
131 CMD$(3) = "XX":CMD$(4) = "DI":CMD$(5) = "XX"
132 CMD$(6) = "LI":CMD$(7) = "XX":CMD$(8) = "RE"
133 CMD$(9) = "WR":CMD$(10) = "AD"
135 COUNT = 0
140 CC = 10: REM  LAST COMMAND
150 D$ =  CHR$ (4)
200  TEXT : HOME
210  VTAB 8: HTAB 12
220  PRINT "STOCK MANAGER 1.0"
230  VTAB 12: HTAB 13: INVERSE
240  PRINT "BY R.A. GUEST": NORMAL
250  FOR I = 1 TO 1000: NEXT I
300  REM  DISPLAY MENU
310  HOME :T =  FRE (0): REM  CLEAN UP STRINGS
320  VTAB 2: HTAB 18
325  REM  ** PRINT COMMANDS **
330  PRINT "MENU"
340  VTAB 4: INVERSE : PRINT "ADD";: NORMAL : PRINT " HOLDING"
350  INVERSE : PRINT "APPRECIATION"
360  PRINT "CHANGE";: NORMAL : PRINT " HOLDING"
370  INVERSE : PRINT "DIVIDENDS": NORMAL
380  INVERSE : PRINT "LIST";: NORMAL : PRINT " HOLDINGS"
390  INVERSE : PRINT "READ";: NORMAL : PRINT " DATA FILE"
395  INVERSE : PRINT "WRITE";: NORMAL : PRINT " DATA FILE"
399  INVERSE : PRINT "EXIT": NORMAL
400  VTAB 22: HTAB 10
410  INPUT "COMMAND: ";YN$
415  REM  ** SEARCH FOR COMMAND  **
420  FOR I = 0 TO CC: IF CMD$(I) =  LEFT$ (YN$,2) GOTO 500
430  NEXT
440  GOTO 400
500 I = I + 1
510  ON I GOSUB 20000,18000,24000,19000,28000,19000,32000,19000,36000,380
     00,40000
600  GOTO 300
18000  REM  ** EXIT **
18020  INPUT "DO YOU NEED TO UPDATE FILE ";YN$
18040  IF  LEFT$ (YN$,1) = "Y" THEN  GOSUB 38000: REM  CLEAR AND UPDATE
18060  END
19000  REM  ** UNIMPLEMENTED **
19040  PRINT "NO SUCH COMMAND"
19060  RETURN
20000  REM  CAPITAL GAINS(AP)
20010  REM  HOLDINGS >1 YEAR
20020  REM  INVERSED FOR LTG
20080  INPUT "CURRENT DATE (MMDDYY) ";CD$
20100  HOME : VTAB 10: HTAB 13
20120  INPUT "ALL/NOTSOLD/SOLD ";YN$
20140 ANS = 0: IF  LEFT$ (YN$,1) = "N" THEN ANS = 1
20160  IF  LEFT$ (YN$,1) = "S" THEN ANS = 2
20200  REM
20210 INDEX = 0: HOME :LINE = 30:DG = 0:TV = 0
20220  IF INDEX >  = COUNT GOTO 20900: REM  DONE
20230  IF ANS = 0 GOTO 20300
20240  IF (ANS = 1) AND (SD$(INDEX) <  > " ") GOTO 20540
20250  IF (ANS = 2) AND (SD$(INDEX) = " ") GOTO 20540
20260  REM  ** USE 'ADD' TO ENTER INFOR **
20300  REM  OUTPUT HEADER
```

```
20320  IF LINE > 18 THEN  GOSUB 52000
20330 F1 = 0: REM  IF NOT SOLD, USE CURRENT DATA
20340  IF SD$(INDEX) = " " THEN F1 = 1:SD$(INDEX) = CD$
20349  REM  ** CALCULATE YEAR DIFFERENCE **
20350 TP =  VAL ( RIGHT$ (SD$(INDEX),2)) -  VAL ( RIGHT$ (PD$(INDEX),2))
20351 TP = TP * 12: REM  CONVERT TO MONTHS
20355  REM  ** CALCULATE MONTH DIFFERENCE **
20360 TP = TP +  VAL ( LEFT$ (SD$(INDEX),2)) -  VAL ( LEFT$ (PD$(INDEX),2))
20362  REM  ** DELETE ENTRY **
20365  IF TP < 12 GOTO 20395
20370  INVERSE : REM  LONG TERM GAIN
20395  IF F1 THEN SD$(INDEX) = " "
20400  PRINT  LEFT$ (NM$(INDEX),10);: NORMAL : HTAB 12
20410  REM  ** CALCULATE DOLLAR GAIN **
20420 TP$ =  STR$ ( INT ((SP(INDEX) - PP(INDEX)) * 100 + .5) / 100)
20430  IF  LEN (TP$) < 8 THEN TP$ = " " + TP$: GOTO 20430
20440  PRINT TP$;: HTAB 20
20450 DG = DG +  VAL (TP$): REM  TOTAL DOLLAR VALUE
20460 TV = PP(INDEX) + TV: REM  TOTAL VALUE
20465  REM  ** CALCULATE % GAIN **
20470 TT = ( VAL (TP$) / PP(INDEX)) * 100
20480 TT$ =  STR$ ( INT (TT * 100 + .5) / 100): REM  PERCENT GAIN
20490  IF  LEN (TT$) < 7 THEN TT$ = " " + TT$: GOTO 20490
20500  PRINT TT$
20520 LINE = LINE + 1
20540 INDEX = INDEX + 1
20560  GOTO 20220: REM  DO NEXT ONE
20890  REM  ** PRINT TOTALS **
20900  PRINT : PRINT "TOTALS";: HTAB 10: PRINT "$GAIN ";DG
20910  IF TV = 0 GOTO 20940
20920  HTAB 10: PRINT "%GAIN ";( INT ((DG / TV) * 100 + .5))
20940  PRINT
20960  GOSUB 51000: REM  WAIT FOR KEY PRESS
20970  RETURN
24000  REM  ** CHANGE/DELETE HOLDING **
24020  REM  ** INPUT '*' FOR NAME TO DELETE **
24040  REM  ** INPUT A BLANK FOR SALE DATE IF NOT YET SOLD **
24200  INPUT "SEARCH STRING ";TS$
24220  FOR K = 0 TO (COUNT - 1)
24222  IF TS$ =  LEFT$ (NM$(K), LEN (TS$)) GOTO 24300
24225  NEXT K
24240  PRINT "NOT FOUND": FOR KK = 1 TO 300: NEXT : RETURN
24300 TP = COUNT:COUNT = K
24302  PRINT NM$(K): PRINT PD$(K): PRINT SD$(K): PRINT PP(K): PRINT SP(K)
     : PRINT DV(K): PRINT SH(K)
24320  PRINT "ENTER '*' FOR NAME TO DELETE."
24330  FOR KK = 1 TO 400: NEXT
24340  GOSUB 40100: REM  GET FIELDS
24360  IF NM$(K) <  > "*" THEN COUNT = TP: RETURN
24365 COUNT = COUNT - 1
24367  REM  ** MOVE REST DOWN IN LIST **
24370  FOR K = COUNT TO TP - 2
24380 K1 = K + 1
24390 NM$(K) = NM$(K1):PD$(K) = PD$(K1):SD$(K) = SD$(K1)
24400 PP(K) = PP(K1):SP(K) = PP(K1):DV(K) = DV(K1):SH(K) = SH(K1)
24420  NEXT
24440 COUNT = TP - 1
24460  RETURN
26000  REM  ** CLEAR SALE PRICE OF UNSOLDS **
26100  FOR I = 0 TO COUNT - 1
26120  IF SD$(I) = " " THEN SP(I) = 0
26140  NEXT
26200  RETURN
28000  REM  ** ESTIMATE DIVIDEND GAIN **
28020  INPUT "CURRENT DATE (MMDDYY) ";CD$
28040  HOME : VTAB 10: HTAB 13
28060  INPUT "ALL/NOTSOLD/SOLD ";YN$
28080 ANS = 0: IF  LEFT$ (YN$,1) = "N" THEN ANS = 1
28100  IF  LEFT$ (YN$,1) = "S" THEN ANS = 2
28120 INDEX = 0: HOME :LINE = 30:DG = 0:TV = 0
28180  REM  ** TEST IF DONE **
28200  IF INDEX >  = COUNT THEN 28900
28220  IF ANS = 0 GOTO 28280
28240  IF (ANS = 1) AND (SD$(INDEX) <  > " ") GOTO 28620
28260  IF (ANS = 2) AND (SD$(INDEX) = " ") GOTO 28620
```

```
36000  REM  ** READ STOCK LISTING FILE **
36100  INPUT "FILE NAME ";F$
36120  PRINT D$;"OPEN ";F$
36140  PRINT D$;"READ ";F$
36200  INPUT COUNT
36220  FOR I = 0 TO (COUNT - 1)
36240  INPUT NM$(I): INPUT PD$(I): INPUT SD$(I)
36260  INPUT PP(I): INPUT SP(I)
36280  INPUT DV(I): INPUT SH(I)
36285  REM  ** CHECK FOR NOT SOLD **
36290  IF  LEN (SD$(I)) < 6 THEN SD$(I) = " "
36300  NEXT
36320  PRINT D$;"CLOSE ";F$
36325  REM  ** GET PRICES FOR STOCKS NOT SOLD **
36330  FOR I = 0 TO (COUNT - 1)
36340  IF SD$(I) <  > " " GOTO 36370
36350  PRINT NM$(I)
36360  INPUT "CURRENT PRICE ";SP(I)
36370  NEXT
36400  RETURN
38000  REM  ** UPDATE STOCK LISTING FILE **
38050  GOSUB 26000: REM  CLEAR NOT SOLD PRICES
38100  INPUT "FILE NAME ";F$
38120  PRINT D$;"OPEN ";F$
38140  PRINT D$;"WRITE ";F$
38200  PRINT COUNT
38220  FOR I = 0 TO (COUNT - 1)
38240  PRINT NM$(I): PRINT PD$(I): PRINT SD$(I)
38242  PRINT PP(I): PRINT SP(I): PRINT DV(I): PRINT SH(I)
38260  NEXT
38300  PRINT D$;"CLOSE ";F$
38320  RETURN
40000  REM  ** ADD A HOLDING **
40080  HOME : VTAB 4
40100  INPUT "NAME ";NM$(COUNT)
40110  PRINT "INPUT DATES IN THE FORM (MMDDYY)"
40120 NM$(COUNT) =  LEFT$ (NM$(COUNT),25)
40140  INPUT "PURCH DATE ";PD$(COUNT):PD$(COUNT) =  LEFT$ (PD$(COUNT),6)
40145  PRINT "ENTER A SINGLE BLANK IF NOT SOLD"
40150  INPUT "SALE DATE ";SD$(COUNT):SD$(COUNT) =  LEFT$ (SD$(COUNT),6)
40155  IF SD$(COUNT) = "" THEN SD$(COUNT) = " "
40160  INPUT "PURCH PRICE ";PP(COUNT)
40170  INPUT "SALE PRICE ";SP(COUNT)
40180  INPUT "DIVIDEND/SHARE ";DV(COUNT)
40190  INPUT "SHARES ";SH(COUNT)
40300 COUNT = COUNT + 1
40400  RETURN
50000  REM  ** WAIT FOR (CR) THEN **
50010  REM  ** OUTPUT HEADING FOR 'LIST' **
50020  REM
50100  GOSUB 51000: HOME
50110  PRINT "NAME  ";
50120  PRINT "PDATE  ";
50130  PRINT "SDATE   ";
50140  PRINT "PPRICE ";
50150  PRINT "SPRICE  ";
50160  PRINT "DIV "
50170  PRINT
50200 LINE = 2
50300  RETURN
51000  REM  ** WAIT FOR (CR) TO BE PRESSED **
51010  VTAB 23: HTAB 5
51020  PRINT "PRESS 'RETURN' WHEN READY "
51050  POKE  - 16368,0
51100  IF  PEEK ( - 16384) = 141 THEN  RETURN
51200  GOTO 51100
52000  REM  ** WAIT FOR (CR) AND **
52020  REM  ** PRINT HEADER      **
52040  REM  ** FOR APPRECIATION AND DIVIDEND **
52060  GOSUB 51000: HOME : HTAB 4
52080  PRINT "NAME";: HTAB 14
52100  PRINT "$GAIN";: HTAB 21
52120  PRINT "%GAIN"
52140  PRINT
52160 LINE = 2
52180  RETURN
```

```
28270  REM  ** PRINT HEADER **
28280  IF LINE > 18 THEN  GOSUB 52000
28290  REM  ** USE CURRENT DATE OR UNSOLDS **
28300  IF DV(INDEX) = 0 GOTO 28620: REM  DON'T USE
28305 F1 = 0
28310  IF SD$(INDEX) = " " THEN F1 = 1:SD$(INDEX) = CD$
28315  REM  ** CALCULATE MONTHS **
28320 MN =  VAL ( LEFT$ (SD$(INDEX),2)) -  VAL ( LEFT$ (PD$(INDEX),2))
28323  REM  ** CALCULATE YEARS **
28325 YR =  VAL ( RIGHT$ (SD$(INDEX),2)) -  VAL ( RIGHT$ (PD$(INDEX),2))
28327  REM  ** CONVERT TO MONTHS **
28330 MN = MN + YR * 12
28340  IF F1 THEN SD$(INDEX) = " "
28400  PRINT  LEFT$ (NM$(INDEX),10);: HTAB 12
28410  REM  ** ESTIMATE DIVIDENDS PAID **
28420 TP =  INT ((DV(INDEX) * SH(INDEX) * (MN / 12)) * 100 + .5) / 100
28440 TP$ =  STR$ (TP)
28460  IF  LEN (TP$) < 8 THEN TP$ = " " + TP$: GOTO 28460
28480  PRINT TP$;: HTAB 20
28490  REM  ** CALCULATE DOLLAR GAIN AND **
28495  REM  ** TOTAL VALUE **
28500 DG = DG +  VAL (TP$):TV = TV + PP(INDEX)
28510  REM  ** CALCULATE % GAIN **
28520 TT =  INT (( VAL (TP$) / PP(INDEX)) * 100 + .5)
28540 TT$ =  STR$ (TT)
28560  IF  LEN (TT$) < 7 THEN TT$ = " " + TT$: GOTO 28560
28580  PRINT TT$
28600 LINE = LINE + 1
28620 INDEX = INDEX + 1
28640  GOTO 28200
28900  GOSUB 20900: REM  OUTPUT TOTALS
28920  RETURN
32000  REM  ** LIST CURRENT HOLDINGS **
32100  HOME : VTAB 10: HTAB 10
32110  INPUT "ALL/NOTSOLD/SOLD ";YN$
32120 ANS = 0: REM  ALL
32130  IF  LEFT$ (YN$,1) = "N" THEN ANS = 1: REM  NOTSOLD
32140  IF  LEFT$ (YN$,1) = "S" THEN ANS = 2: REM  SOLD
32210 INDEX = 0: HOME :LINE = 30:TPP = 0:TSP = 0
32300  IF INDEX >  = COUNT GOTO 32900
32302  IF ANS = 0 GOTO 32310
32304  IF (ANS = 1) AND (SD$(INDEX) = " ") GOTO 32310
32306  IF (ANS = 2) AND (SD$(INDEX) <  > " ") GOTO 32310
32308 INDEX = INDEX + 1: GOTO 32300
32310  IF LINE > 18 THEN  GOSUB 50000: REM  WAIT AND PRINT HEADER
32320  PRINT  LEFT$ (NM$(INDEX),5);: HTAB 7
32330  PRINT  LEFT$ (PD$(INDEX),6);: HTAB 14
32340  PRINT  LEFT$ (SD$(INDEX),6);: HTAB 21
32350  REM  ** PURCHASE PRICE **
32360 TP$ =  STR$ ( INT (PP(INDEX) * 10.0 + 0.5) / 10.0)
32380  IF  LEN (TP$) < 7 THEN TP$ = " " + TP$: GOTO 32380
32390  PRINT TP$;: HTAB 29
32395  REM  ** SALE PRICE **
32400 TP$ =  STR$ ( INT (SP(INDEX) * 10.0 + 0.5) / 10.0)
32410  IF  LEN (TP$) < 7 THEN TP$ = " " + TP$: GOTO 32410
32420  PRINT TP$;: HTAB 37
32425  REM  ** DIVIDEND **
32430 TP$ =  STR$ ( INT (DV(INDEX) * 10.0 + 0.5) / 10.0)
32440  IF  LEN (TP$) < 3 THEN TP$ = " " + TP$: GOTO 32440
32450  PRINT TP$
32455  REM  ** NUMBER OF SHARES **
32460  PRINT " ";SH(INDEX)
32465  REM  ** COMPUTE TOTAL SALES AND **
32466  REM  ** TOTAL PURCHASE PRICES **
32470 TSP = TSP + SP(INDEX):TPP = TPP + PP(INDEX)
32480  PRINT
32800 LINE = LINE + 3
32810 INDEX = INDEX + 1
32820  GOTO 32300
32880  REM  ** PRINT TOTALS **
32900  PRINT : PRINT "TOTALS"
32910  HTAB 10: PRINT "PPRICES ";TPP
32920  HTAB 10: PRINT "SPRICES ";TSP
32960  GOSUB 51000: REM  WAIT FOR KEY PRESS
32970  RETURN
```

Solar System Simulation

by Dave Partyka

This program will print information about the first six planets of the Solar System, and plot their positions. In the printing mode, information such as distance from the earth and sun, and other data about the earth and planet relation is printed. In the plot mode, the planets' positions against the zodiac, as seen from the earth, are plotted, using hi-res graphics and scaling factors.

This program deals with the first six planets, but instead of being heliocentric (sun centered) it's geocentric (earth centered). It gives a display of the planets as seen from the earth. The planets are displayed against a star background and their motions through the zodiac are very good representations of the actual positions of the planets. Using this program, you can watch as a planet makes its retrograde loop through a constellation, see how close two or more planets come to each other, or watch how close a planet comes to a bright star.

The program is set up in two parts. One part prints values on the screen for each planet and the sun, and the other plots the positions of the planets against a star background. If you choose to print, at the top of the screen is the starting date and the number of days that the display is for. The program then prints the following data for each planet:

D-S; the distance in million miles that the planet is from the sun.

A-S; the angle in degrees that the planet is located around the sun.

D-E; the distance in million miles that the planet is from the earth.

R.A.; the right ascension in hours and minutes that the planet appears from the earth.

DEC.; the declination in degrees and minutes that the planet appears from the earth.

You can display the values for all the planets, or for specific ones. You can display a single day, or a range of days with any number of days between the displays. The program will pause after each display, and then wait for you to press RETURN to continue with the display, or with a set of questions for a new display.

If you choose to plot, another set of questions will be asked. These are needed to set the limits for the star display and to determine if you want point or continuous plots. Just like printing, you can plot for single or multiple days, with any number of days between plots. You can plot single points (with the previous plot erased before the current one is plotted), or continuous plots (where the points aren't erased but remain on the screen). After that you'll be asked for a scaling factor: 0 or 1-20. A scaling factor of zero will display the full star field, right ascension 0 to 24 hours, and declination 90 to −90 degrees.

A scaling factor equal to or greater than 1 (a factor between zero and one is not allowed) displays another question, "Enter center coordinates for R.A. and DEC." This will determine the center coordinates of the display, and is in hours and decimal hours, degrees and decimal degrees. The scaling factor you entered, along with the center coordinates, will determine the right and left, top and bottom limits of the display.

The higher the scaling factor, the less of a constellation you'll see, but the greater the movement of the planet per plot. A scaling factor of 1 displays approximately 18 hours in right ascension and 180 degrees in declination, and a factor of 10 displays, approximately 2 hours in right ascension and 19 degrees in declination.

The only constellations in the star table are for the zodiac. If you want to increase the number of stars within the zodiac, or if you want to add more constellations, it's an easy process. The table is set up with four values per star. The first two are for right ascension in hours, minutes; the next two are for declination in degrees, minutes. The stars in the table don't have to be in any particular order. The whole table is read when the plot portion of the program is used. The only table requirements are the two values for right ascension and two values for declination. If the declination is negative, then both values for declination have to be negative. To end the table, four zeros are necessary — 0,0,0,0.

You may want to split this program to make one that just displays the stars on the screen. Just begin where the question for a scaling factor is asked, and delete everything else that isn't used. You can add more tables to the new program: one for galaxies, another for star clusters, another for nebulae, or even one for the Messier objects. The tables you add will be whatever you need, and by adding more questions, you can display the different tables, either alone or combined.

Let's go through two examples of the program, first for figure 1, and second for figure 2. The first question that will be asked is if you want to display the same planets as your last run. Since this is the first run, enter N. Then it will ask "What planets do you want to display?" Enter a 1 for each planet. Then a starting date is asked. Use 11,1,1979. After that, it says "Enter the number of days to plot." Enter 150. Then it asks to print or plot. Enter a 1 to print. The screen will then clear, print the starting date and the plot day's value at the top of the screen, and then continue to print for the planets and the sun.

After finishing the page, it will pause and display "Press return for next display." After you press return it will start printing again, changing the plot day's value at the top of the page and the values for the planets and the sun. It will continue to do this until the plot day's value is equal to or greater than the day's that you wanted to print for. After that, it will ask you to press return to start again. When you press return, it will ask if you want to display the same planets as your last run.

Figure 1: **Example of the print routine for all planets, starting date 11/1/1979 for 240 days at 50-day intervals at the 150th day.**

		Starting Date 11/1/1979		Plot Days 150	
Earth	D-S.	92.8887	Sun	D-E.	92.8887
	A-S.	189.4489		R.A.	0 34.7
				DEC.	3 44.6
Mercury	D-S.	43.1581	Venus	D-S.	66.8181
	A-S.	245.1156		A-S.	140.7176
	D-E.	77.2616		D-E.	70.0302
	R.A.	22 55.3		R.A.	3 28.3
	DEC.	-8 7.1		DEC.	21 55
Mars	D-S.	154.4251	Jupiter	D-S.	502.2398
	A-S.	170.2956		A-S.	158.0192
	D-E.	73.2592		D-E.	425.652
	R.A.	9 56.5		R.A.	10 15.9
	DEC.	16 7		DEC.	12 9.5
Saturn	D-S.	875.6875			
	A-S.	174.1555			
	D-E.	785.842			
	R.A.	11 35.7			
	DEC.	5 15			

Press return for next display.

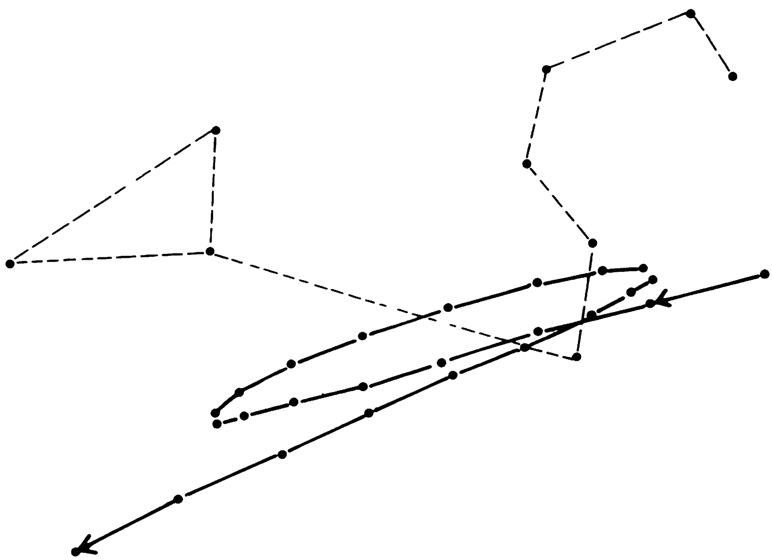
For example 2, enter an N to the last question so that it will ask you which planets you want to display. Enter a 0 (zero) for all the planets except Mars. Enter 11,1,1979 for the starting date, 240 for the number of days to plot, and 10 for the number of days between plots. When it asks to print or plot, enter a 0 (zero) to plot. Three requests will then be made: the first, "enter 0 for point, or 1 for continuous plots." Since we want all the points to remain on the screen, enter 1 for continuous plots. The next question is the scaling factor. Enter a 5. After that will be the center coordinates. Since I already know that the planet Mars will be in the constellation Leo, enter 10.5 for right ascension, and 18 for declination.

When you do plots for other planets and you don't know where they will be, run the print program first and get the right ascension and declination. After entering the center coordinates, the screen will clear and a window will appear on the screen. After a few seconds the constellation Leo will appear as the star table is read, and any stars within the display limits will be plotted. A few more seconds will pass as the rest of the table is read. Once the end of the table is found, the program will beep to signal the start of the calculations.

Since the planet Mars was the only planet picked, the program will calculate the positions of the earth and Mars. The position of the earth is always calculated, but only printed during the print option, (if you choose to print it). The program will continue to plot the position of Mars, beeping each time it starts a new sequence of calculations. It will plot 25 times — one for the starting date and 24 for 240 days, at 10-day intervals.

The program will then do a double beep to signal the end of the simulation and wait until you press return before starting a new sequence of questions. The purpose of the single beep at the beginning of the calculations is to identify what planet is being plotted. The planets are plotted in their order from the sun. If you plot more than one planet in the same display, you can figure out which is which by the plotting order.

Figure 2: **Example of the plot routine for Mars, starting date 11/1/1979 for 240 days at 10-day intervals, continuous plots.**



Since the date doesn't appear anywhere on the display for plotting, you can do a CNTL-C to stop the program, type "TEXT", and then return to see the starting date and the plot day's value. To continue, do POKEs to set graphics mode (−16304) and display the secondary page (−16299), type "CONT" and return. The program will pick up where you left off. If you follow these examples, the results you get should match figure 1 at day 150 for printing, and figure 2 at the end of the plotting sequence. The solid and dotted lines in figure 2 were used to show the motion of Mars and the stars of the constellation Leo, and will not be in the actual display. Once you run the two examples to become familiar with the program, then you can enter any values for the questions to display whatever for whenever you want.

```
10  REM ***********************
15  REM *                     *
20  REM *     SOLAR SYSTEM    *
25  REM *      SIMULATION     *
27  REM *     DAVE PARTYKA    *
30  REM *                     *
35  REM *  COPYRIGHT (C) 1982 *
40  REM *    MICRO INK, INC.  *
44  REM * CHELMSFORD, MA 01824 *
45  REM *  ALL RIGHTS RESERVED *
50  REM *                     *
55  REM *                     *
60  REM ***********************
65  REM
70  REM
75  REM
100  GOTO 650
110  IF TY = 1 THEN 210
120  IF H > TP OR H < BT THEN 210
130  HCOLOR= 0
140  IF RG > LF THEN 180
150  IF F < RG OR F > LF THEN 210
160  HPLOT 279 - (F - RG) * SC,(TP - H) * SC
170  GOTO 210
180  IF F > LF AND F < RG THEN 210
190  IF F =  < LF THEN F = F + 360
200  HPLOT 279 - (F - RG) * SC,(TP - H) * SC
210  IF G > TP OR G < BT THEN  RETURN
220  HCOLOR= 3
230  IF RG > LF THEN 270
240  IF B < RG OR B > LF THEN  RETURN
250  HPLOT 279 - (B - RG) * SC,(TP - G) * SC
260  RETURN
270  IF B > LF AND B < RG THEN  RETURN
280  IF B =  < LF THEN B = B + 360
290  HPLOT 279 - (B - RG) * SC,(TP - G) * SC
300  RETURN
310 D = ZZ -  INT (ZZ / SRD) * SRD
320 B = Q - (D / SRD * Q2)
330  IF Y > 0 THEN RA = 270
340 RV = A - (P / (1 + E *  COS (B)))
350 V = PE / RV - EZ
360  IF V =  > 1 THEN V = VL
370  IF V =  <  - 1 THEN V =  - VL
380 VA =  -  ATN (V /  SQR ( - V * V + 1)) + T
390  IF D > SRD / 2 THEN VA = Q2 - VA
400 VA = VA + J
410 ZX = VA * T1 - C
420  IF ZX > 360 THEN ZX = ZX - 360
430  IF ZX < 0 THEN ZX = 360 + ZX
440 ZX = ZX / T1
450 LA =  SIN (ZX) * I
460 XA = RV *  COS (LA) *  COS (VA)
470 YA = RV *  COS (LA) *  SIN (VA)
480 ZA = RV *  SIN (LA)
```

```
490 XB = XA - X3:YB = YA - Y3:ZB = ZA - Z3
500 VA = VA * T1
510  IF VA > 360 THEN VA = VA - 360
520  IF EE = 0 THEN  RETURN
530 ED =  SQR (XB * XB + YB * YB)
540 X = XB
550 Y = YB *  COS (IN) - ZB *  SIN (IN)
560 Z = YB *  SIN (IN) + ZB *  COS (IN)
570 RA = 90
580  IF Y < 0 THEN RA = 270
590  IF X <  > 0 THEN RA =  ATN (Y / X) * T1
600  IF X < 0 THEN RA = RA + 180
610  IF X > 0 AND Y < 0 THEN RA = RA + 360
620 DZ = Z / ED
630 DC =  ATN (DZ /  SQR (1 - DZ * DZ)) * T1
640  RETURN
650 T = 1.5708:T1 = 57.2957795
660 IN = 23.434 / T1
670 Q = 3.14159265
680 Q2 = 6.2831853
690 VL = .99999999
700  HOME
710  PRINT "DO YOU WANT TO DISPLAY "
720  PRINT : PRINT "THE SAME PLANETS AS YOUR LAST RUN"
730  PRINT : INPUT "Y OR N ";A$
740  IF A$ = "N" THEN 790
750  IF A$ <  > "Y" THEN 710
760  IF S1 <  > 0 THEN 1590
770  IF SC <  > 0 THEN 2785
780  PRINT : PRINT "YOU HAV'NT PICKED THE PLANETS YET": PRINT : PRINT : GOTO
     800
790  HOME
800  PRINT "CHOOSE THE PLANETS YOU WANT TO DISPLAY"
810  PRINT
820  PRINT "ENTER A 1 FOR YES, 0 FOR NO"
830  PRINT
840  REM  SPACIFIC VALUES FOR EACH PLANET
850  REM  S1=ORBITAL PERIOD: P1=A1*(1-E1*E1)/2
860  REM  E1=ECCENTRICITY: U1=P1/E1: K1=1/E1
870  REM  A1=MINIMUM + MAXIMUM DISTANCE FROM SUN
880  REM  J1=LONGITUDE OF PERIHELION IN RADIANS
890  REM  W1=DAYS FROM 0 DEGREES TO PERIHELION FOR 1980
892  REM  C1=ASCENDING NODE IN DEGREES
894  REM  I1=INCLINATION IN DEGREES / T1 TO CONVERT TO RADIANS
900  INPUT "DISPLAY MERCURY       ";ME
910 S1 = 87.969
920 E1 = .2056
930 A1 = 43.403 + 28.597
940 P1 = A1 * (1 - E1 * E1) / 2
950 K1 = 1 / E1
960 U1 = P1 / E1
970 J1 = 77.1 * Q / 180
980 W1 = 37.53
990 C1 = 48.1
1000 I1 = 7 / T1
1010  INPUT "DISPLAY VENUS         ";VE
1020 S2 = 224.701
1030 E2 = .0068
1040 A2 = 67.726 + 66.813
1050 P2 = A2 * (1 - E2 * E2) / 2
1060 K2 = 1 / E2
1070 U2 = P2 / E2
1080 J2 = 131.3 * Q / 180
1090 W2 = 140.5
1100 C2 = 76.5
1110 I2 = 3.4 / T1
1120  INPUT "DISPLAY EARTH         ";EA
1130 S3 = 365.256
1140 E3 = .0167
1150 A3 = 94.555 + 91.445
1160 P3 = A3 * (1 - E3 * E3) / 2
1170 K3 = 1 / E3
1180 U3 = P3 / E3
1190 J3 = 102.6 * Q / 180
1200 W3 =  - 3.82
```

```
1210 C3 = 0
1220 I3 = 0
1230  INPUT "DISPLAY MARS          ";MA
1240 S4 = 686.980
1250 E4 = .0934
1260 A4 = 154.936 + 128.471
1270 P4 = A4 * (1 - E4 * E4) / 2
1280 K4 = 1 / E4
1290 U4 = P4 / E4
1300 J4 = 335.7 * Q / 180
1310 W4 = 287
1320 C4 = 49.4
1330 I4 = 1.85 / T1
1340  INPUT "DISPLAY JUPITER       ";JU
1350 S5 = 4332.125
1360 E5 = .0478
1370 A5 = 507.046 + 460.595
1380 P5 = A5 * (1 - E5 * E5) / 2
1390 K5 = 1 / E5
1400 U5 = P5 / E5
1410 J5 = 13.6 * Q / 180
1420 W5 = 1608
1430 C5 = 100.24
1440 I5 = 1.3 / T1
1450  INPUT "DISPLAY SATURN        ";SA
1460 S6 = 10825.863
1470 E6 = .0555
1480 A6 = 937.541 + 838.425
1490 P6 = A6 * (1 - E6 * E6) / 2
1500 K6 = 1 / E6
1510 U6 = P6 / E6
1520 J6 = 95.5 * Q / 180
1530 W6 = 2090
1540 C6 = 113.51
1550 I6 = 2.49 / T1
1590  HOME
1600  PRINT "ENTER BEGINNING DATE? MM,DD,YYYY": INPUT "
        ";MM,DD,YY
1610 DF = (MM = 2) * 31 + (MM = 3) * 59 + (MM = 4) * 90 + (MM = 5) * 120 +
     (MM = 6) * 151 + (MM = 7) * 181 + (MM = 8) * 212 + (MM = 9) * 243 +
     (MM = 10) * 273 + (MM = 11) * 304 + (MM = 12) * 334
1620 ZY =  INT (YY * 365 +  INT (YY / 4) + DD + DF + 1 -  INT (YY / 100) +
      INT (YY / 400) / 1)
1630  IF  INT (YY / 4) <  > YY / 4 THEN 1680
1640  IF  INT (YY / 400) = YY / 400 THEN 1660
1650  IF  INT (YY / 100) = YY / 100 THEN 1670
1660  IF MM > 2 THEN 1680
1670 ZY = ZY - 1
1680 ZY = ZY - 723180
1690 ZT =  - ZY
1700  PRINT : PRINT : INPUT "ENTER # OF DAYS TO PRINT/PLOT ";DN
1710  PRINT : PRINT : PRINT
1720  INPUT "ENTER # OF DAYS BETWEEN PRINT/PLOTS ";DA
1730  IF DA <  > 0 THEN 1760
1740  PRINT : PRINT
1750  PRINT "0 NOT ALLOWED": GOTO 1710
1760  HOME
1770  INPUT "ENTER 1 TO PRINT, 0 TO PLOT  ";PL
1780  IF PL <  > 0 AND PL <  > 1 THEN 1760
1785  IF PL = 0 THEN  PRINT : PRINT "DO YOU WANT": PRINT : INPUT "POINT (
     0) OR CONTINUOUS (1) PLOTS ";TY
1786  IF TY <  > 0 AND TY <  > 1 THEN 1785
1790  IF PL = 0 THEN  GOSUB 2750
1800  REM   EARTH
1810  HOME :EE = 0
1830 A = A3:P = P3:E = E3:PE = U3:EZ = K3:SRD = S3:J = J3:W = W3:ZZ = ZY +
     W:C = C3:I = I3
1840  GOSUB 310:EE = 1
1845 X3 = XA:Y3 = YA:Z3 = ZA:R3 = RV:V3 = VA
1848  HOME
1850  VTAB 1: HTAB 1: PRINT "STARTING DATE ";MM;"/";DD;"/";YY;"   PLOT DA
     YS ";ZT + ZY
1855  IF PL = 0 THEN  VTAB 23: PRINT "STARTING DATE ";MM;"/";DD;"/";YY;"
      PLOT DAYS ";ZT + ZY: PRINT "": GOTO 1980: REM  EMPTY PRINT IS A CN
     TL-G (BELL)
```

```
1870  IF EA = 0 THEN 1980
1880  VTAB 2: HTAB 1: PRINT "EARTH D-S. "; INT (RV * 10000) / 10000
1890  VTAB 3: HTAB 7: PRINT "A-S. "; INT (V3 * 10000) / 10000
1900  REM   SUN
1910 XB =  - X3:YB =  - Y3:ZB =  - Z3:ED = R3
1920  GOSUB 540
1930  VTAB 2: HTAB 21: PRINT "SUN    D-E. "; INT (ED * 10000) / 10000
1940  VTAB 3: HTAB 28: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
     (RA / 15) * 15) * 40) / 10
1950  IF DC < 0 THEN DC =  - DC:DB = 1
1960  VTAB 4: HTAB 28: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC)
     ) * 600) / 10
1970  IF DB = 1 THEN  VTAB 4: HTAB 32: PRINT "-":DB = 0
1980  REM   MERCURY
1990  IF ME = 0 THEN 2130
2000 A = A1:P = P1:E = E1:PE = U1:EZ = K1:SRD = S1:J = J1:W = W1:ZZ = ZY +
     W:C = C1:I = I1
2010  GOSUB 310: IF PL = 1 THEN 2050
2020 F = F1:H = H1:B = RA:G = DC: GOSUB 110
2030 F1 = RA:H1 = DC: GOTO 2130
2040  IF PL = 0 THEN  GOSUB 110
2050  VTAB 6: HTAB 1: PRINT "MERC  D-S. "; INT (RV * 10000) / 10000
2060  VTAB 7: HTAB 7: PRINT "A-S. "; INT (VA * 10000) / 10000
2070  VTAB 8: HTAB 7: PRINT "D-E. "; INT (ED * 10000) / 10000
2080  VTAB 9: HTAB 7: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
     (RA / 15) * 15) * 40) / 10
2090  IF DC < 0 THEN DC =  - DC:DB = 1
2100  VTAB 10: HTAB 7: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC)
     ) * 600) / 10
2110  IF DB = 1 THEN  VTAB 10: HTAB 11: PRINT "-":DB = 0
2120  REM   VENUS
2130  IF VE = 0 THEN 2260
2140 A = A2:P = P2:E = E2:PE = U2:EZ = K2:SRD = S2:J = J2:W = W2:ZZ = ZY +
     W:C = C2:I = I2
2150  GOSUB 310: IF PL = 1 THEN 2180
2160 F = F2:H = H2:B = RA:G = DC: GOSUB 110
2170 F2 = RA:H2 = DC: GOTO 2260
2180  VTAB 6: HTAB 21: PRINT "VENUS  D-S. "; INT (RV * 10000) / 10000
2190  VTAB 7: HTAB 28: PRINT "A-S. "; INT (VA * 10000) / 10000
2200  VTAB 8: HTAB 28: PRINT "D-E. "; INT (ED * 10000) / 10000
2210  VTAB 9: HTAB 28: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
     (RA / 15) * 15) * 40) / 10
2220  IF DC < 0 THEN DC =  - DC:DB = 1
2230  VTAB 10: HTAB 28: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC
     )) * 600) / 10
2240  IF DB = 1 THEN  VTAB 10: HTAB 32: PRINT "-":DB = 0
2250  REM   MARS
2260  IF MA = 0 THEN 2390
2270 A = A4:P = P4:E = E4:PE = U4:EZ = K4:SRD = S4:J = J4:W = W4:ZZ = ZY +
     W:C = C4:I = I4
2280  GOSUB 310: IF PL = 1 THEN 2310
2290 F = F4:H = H4:B = RA:G = DC: GOSUB 110
2300 F4 = RA:H4 = DC: GOTO 2390
2310  VTAB 12: HTAB 1: PRINT "MARS  D-S. "; INT (RV * 10000) / 10000
2320  VTAB 13: HTAB 7: PRINT "A-S. "; INT (VA * 10000) / 10000
2330  VTAB 14: HTAB 7: PRINT "D-E. "; INT (ED * 10000) / 10000
2340  VTAB 15: HTAB 7: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
     (RA / 15) * 15) * 40) / 10
2350  IF DC < 0 THEN DC =  - DC:DB = 1
2360  VTAB 16: HTAB 7: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC)
     ) * 600) / 10
2370  IF DB = 1 THEN  VTAB 16: HTAB 11: PRINT "-":DB = 0
2380  REM   JUPITER
2390  IF JU = 0 THEN 2520
2400 A = A5:P = P5:E = E5:PE = U5:EZ = K5:SRD = S5:J = J5:W = W5:ZZ = ZY +
     W:C = C5:I = I5
2410  GOSUB 310: IF PL = 1 THEN 2440
2420 F = F5:H = H5:B = RA:G = DC: GOSUB 110
2430 F5 = RA:H5 = DC: GOTO 2520
2440  VTAB 12: HTAB 21: PRINT "JUPTR  D-S. "; INT (RV * 10000) / 10000
2450  VTAB 13: HTAB 28: PRINT "A-S. "; INT (VA * 10000) / 10000
2460  VTAB 14: HTAB 28: PRINT "D-E. "; INT (ED * 10000) / 10000
2470  VTAB 15: HTAB 28: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
     (RA / 15) * 15) * 40) / 10
2480  IF DC < 0 THEN DC =  - DC:DB = 1
```

```
2490  VTAB 16: HTAB 28: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC
    )) * 600) / 10
2500  IF DB = 1 THEN  VTAB 16: HTAB 32: PRINT "-":DB = 0
2510  REM   SATURN
2520  IF SA = 0 THEN 2640
2530 A = A6:P = P6:E = E6:PE = U6:EZ = K6:SRD = S6:J = J6:W = W6:ZZ = ZY +
    W:C = C6:I = I6
2540  GOSUB 310: IF PL = 1 THEN 2570
2550 F = F6:H = H6:B = RA:G = DC: GOSUB 110
2560 F6 = RA:H6 = DC: GOTO 2640
2570  VTAB 18: HTAB 1: PRINT "SATN  D-S. "; INT (RV * 10000) / 10000
2580  VTAB 19: HTAB 7: PRINT "A-S. "; INT (VA * 10000) / 10000
2590  VTAB 20: HTAB 7: PRINT "D-E. "; INT (ED * 10000) / 10000
2600  VTAB 21: HTAB 7: PRINT "R.A. "; INT (RA / 15);"  "; INT ((RA -  INT
    (RA / 15) * 15) * 40) / 10
2610  IF DC < 0 THEN DC =  - DC:DB = 1
2620  VTAB 22: HTAB 7: PRINT "DEC. "; INT (DC);"  "; INT ((DC -  INT (DC)
    ) * 600) / 10
2630  IF DB = 1 THEN  VTAB 22: HTAB 11: PRINT "-":DB = 0
2640 ZY = ZY + DA
2650  IF ZT + ZY > DN THEN 2700
2660  IF PL = 0 THEN 2690
2670  VTAB 23: HTAB 1: PRINT "PRESS RETURN FOR NEXT DISPLAY": GET A$
2680  VTAB 23: HTAB 1: PRINT "                              "
2690  GOTO 1830
2700 ZY = 0:DE = 0
2710  PRINT "";""
2720  INPUT "PRESS ENTER TO START AGAIN";A$
2730  TEXT : RESTORE
2740  GOTO 650
2750  HCOLOR= 3
2760  PRINT : INPUT "ENTER FACTOR: 0 OR 1 - 20 ";SC
2770  IF SC <  > 0 THEN 2785
2780 RG = 0:LF = 360:BT =  - 90:TP = 110:SC = .75: GOTO 2890
2785  IF SC < 1 THEN 2760
2800  PRINT : PRINT "ENTER CENTER COORDINATES": PRINT
2810  PRINT " R.A.  DEC.": PRINT
2820  INPUT "HH.HH , DD.DD  ";R,D
2830 RG = R * 15 - 139 / SC
2840 LF = R * 15 + 139 / SC
2850 BT = D - 95 / SC
2860 TP = D + 95 / SC
2870  IF RG < 0 THEN RG = RG + 360
2880  IF LF > 360 THEN LF = LF - 360
2890  HGR2
2900  HPLOT 0,0 TO 279,0
2910  HPLOT  TO 279,191
2920  HPLOT  TO 0,191
2930  HPLOT  TO 0,0
2940  READ B,B1,G,G1
2950 B = B * 15 + B1 * .25:G = G + G1 / 60
2960  IF B = 0 AND G = 0 THEN  RETURN
2970  GOSUB 210: GOTO 2940
2980  REM  PISCES
2990  DATA  1,11,24,19,1,17,27,0,1,18,28,29,1,9,29,49,0,55,28,43,0,47,27,
    26,0,53,26,56,1,28,15,5,1,43,8,54,1,59,2,31
3000  DATA  1,39,5,14,1,28,5,53,1,11,7,19,1,0,7,37,0,46,7,19,23,57,6,35,2
    3,37,5,21,23,40,1,30,23,25,6,6,23,18,5,6,23,15,3,1,23,24,0,59
3010  REM  ARIES
3020  DATA  1,51,19,3,1,52,20,34,2,1,25,42
3030  REM  PLEIADES
3040  DATA  3,42,24,8,3,42,23,57,3,42,24,18,3,43,24,13,3,43,24,24,3,45,23
    ,57,3,43,23,48
3050 TAURUS
3060  DATA  5,23,28,34,4,39,22,52,5,35,21,7,5,4,18,35,4,33,16,25,4,26,15,
    51,4,17,15,31,4,23,17,49,4,26,19,4
3070  REM  GEMINI
3080  DATA  6,12,22,31,6,20,22,32,6,41,25,11,7,8,30,20,7,31,32,0,7,42,28,
    9,7,17,22,5,7,1,20,39,6,35,16,27,6,42,12,57
3090  REM  CANCER
3100  DATA  8,14,9,20,8,18,24,11,8,30,20,37,8,29,18,16,8,42,18,20,8,40,21
    ,39,8,56,12,3,8,44,28,57
3110  REM  LEO
3120  DATA 9,43,24,0,9,50,26,15,10,14,23,40,10,17,20,6,10,5,17,0,10,6,12,
    13,11,11,20,48,11,47,14,51,11,12,15,42
```

```
3130  REM  VIRGO
3140  DATA  11,43,6,49,11,48,2,3,12,17,-0,-23,12,39,-1,-11,12,53,3,40,13,
      0,11,14
3200  DATA  13,7,-5,-16,13,23,-10,-54,14,13,-5,-46,14,40,-5,-27,14,44,2,6
      ,13,59,1,47,13,32,-0,-20
3270  REM  LIBRA
3280  DATA  14,48,-15,-50,15,10,-19,-28,15,14,-9,-12,15,33,-14,-37
3320  REM  SCORPIUS
3330  DATA  15,57,-22,-29,16,3,-19,-40,16,18,-25,-28,16,28,-26,-19,16,33,
      -28,-7,16,47,-34,-12,16,48,-37,-58,16,50,-42,-17
3420  DATA  17,9,-43,-11,17,34,-42,-58,17,44,-40,-7,17,39,-39,-0,17,30,-3
      7,-4
3470  REM  SAGITTARIUS
3480  DATA  18,3,-30,-26,18,14,-36,-47,18,21,-34,-25,18,18,-29,-51,18,25,
      -25,-27,18,43,-27,-3,18,52,-26,-22,18,59,-29,-57,19,4,-27,-45
3570  REM  CAPRICORNUS
3580  DATA  20,15,-12,-40,20,24,-18,-23,20,36,-15,-8,21,3,-17,-26,21,19,-
      17,-3,21,37,-16,-53,21,44,-16,-21
3650  DATA  21,40,-19,-6,21,34,-19,-41,21,26,-22,-2,21,24,-22,-38,21,4,-2
      5,-12,20,49,-27,-6,20,43,-25,-27
3720  REM  AQUARIUS
3740  DATA  22,3,-0,-34,22,23,1,7,22,26,-0,-17,22,33,-0,-23,22,50,-7,-51,
      22,47,-13,-51,22,52,-16,-5,23,12,-6,-19,23,13,-9,-22,23,16,-9,-53,23
      ,40,-14,-49
3830  REM  END OF TABLE (ZEROS)
3840  DATA 0,0,0,0

]
```

Othello

by Charles F. Taylor, Jr.

This program simulates the popular board game Othello. Designed for two players, the program maintains the Othello board on the Apple lo-res graphics screen. Written in Applesoft BASIC, Othello should be easily modifiable to other dialects of BASIC.

Most computer game programs are designed for one user. The computer plays the role of opponent, scorekeeper, referee, and manager of the display. This results in a "man-against-machine" scenario. The objective is to "beat the computer" and thereby establish your intellectual superiority over silicon circuitry. (Never mind that you are really playing against an algorithm designed by another person.)

This game program is designed for two persons. The computer no longer is the opponent, but plays the role of slave, keeping track of the board position, checking for illegal moves, keeping score, and managing the display.

Background

I wrote this program for my ten-year-old son. Othello is a good game for interaction across the generation gap because it is more than challenging enough for me, but not too difficult for my son. He beats me more often than I care to admit!

Perhaps the best way to describe the game of Othello is to describe how it is played as a board game, without the aid of the computer. The playing board is eight squares by eight squares, much like a checker or chess board, except that all squares are usually the same color. The playing pieces are disks, black on one side and white on the other. Each player starts with 32 pieces; one player is designated "white" and the other "black."

The game begins with two pieces of each color in the center of the board in the configuration shown in figure 1. White has the first turn. He must place a white piece (a piece with the white side up) in such a manner as to "capture" a black piece. A piece is captured when it is "surrounded" by pieces of the opposite color, either horizontally, vertically, or diagonally. Captured pieces are turned over and become the color of the captor. More than one piece can be captured at a time.

Figure 2 illustrates the capture of two black pieces by a white piece. A move is not legal unless it accomplishes one or more captures. The game is won by either capturing all of your opponent's pieces, or by having more pieces than your opponent at the end of the game.

Implementation

The program was written in Applesoft BASIC on an Apple II Plus. Low-resolution graphics are used to display the game board, thus pieces are shown as square rather than round. The selection of colors is easily changed to suit your own display (see lines 280 - 300). I am currently using a "green screen" monitor and find it hard to judge colors as they might appear on another display.

The program is shown in listing 1. The coding is straightforward, but perhaps a few comments are in order. The board is represented internally by the array "BOARD." The function "FN M2(Q)" finds the modulus base 2 of a number (the remainder after integer division by 2) and is used to compute whose turn it is. The legality of each move is checked. The subroutine at 1430 searches for and executes all possible captures, beeping for each capture. The score is displayed after each move.

Play

To move, a player types the row and column where he wants to place his piece. Columns are labeled A-H, left to right; rows are labeled 1-8, bottom to top. The lower left corner is then A1, the lower right corner H1, and so on. Should you ever find yourself in a position such that no legal moves are possible, type "P" for "Pass." Play tends to ebb and flow like the tides, but without any predictability. A player can be comfortably ahead at one moment and hopelessly behind the next.

Figure 1

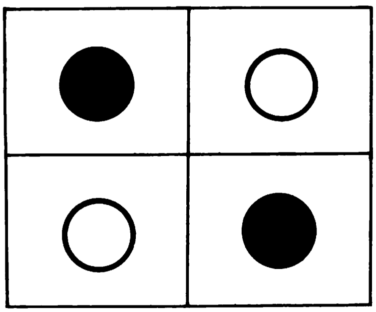
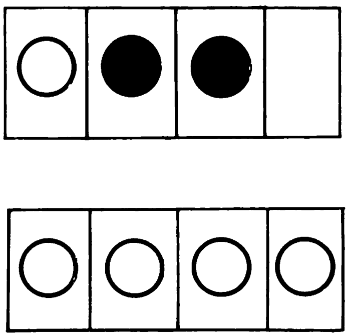


Figure 2



```
1  REM  ************************
2  REM  *                      *
3  REM  *    GAME OF OTHELLO   *
4  REM  *      C.F. TAYLOR     *
5  REM  *                      *
6  REM  *  COPYRIGHT (C) 1982  *
7  REM  *    MICRO INK, INC.   *
8  REM  * CHELMSFORD, MA 01824 *
9  REM  *  ALL RIGHTS RESERVED *
10  REM *                      *
11  REM ************************
12  REM
13  REM
14  REM
15  REM  INITIALIZE
16  DIM BOARD(9,9)
17  DIM CC(2): REM  HOLDS CURRENT COLOR
18  DIM PROMPT$(2)
19  DIM SC(2)
20  DIM DX(8): DIM DY(8)
21  DEF  FN M2(Q) = Q -  INT (Q / 2) * 2
22 PROMPT$(1) = "INPUT WHITE MOVE:"
23 PROMPT$(2) = "INPUT BLACK MOVE:"
24 BLACK = 0
25 WHITE = 15
26 CC(1) = WHITE
27 CC(2) = BLACK
28 BC = 12: REM  BACKGROUND COLOR
29 TC = 13: REM  TITLE COLOR
30 DC = 4: REM  BORDER COLOR
31  DATA  0,1,1,1,0,-1,-1,-1
32  DATA  1,1,0,-1,-1,-1,0,1
33  FOR I = 1 TO 8: READ DX(I): NEXT I
34  FOR I = 1 TO 8: READ DY(I): NEXT I
35  FOR I = 0 TO 9
36  FOR J = 0 TO 9
37 BOARD(I,J) = 0
38  NEXT J,I
39  GOSUB 78
40  COLOR= WHITE
41 X = 5:Y = 5
42 BOARD(X,Y) = 1
43  GOSUB 126: REM  CALL BLOT
44 X = 4:Y = 4
45 BOARD(X,Y) = 1
46  GOSUB 126: REM  CALL BLOT
47 SC(1) = 2
48  COLOR= BLACK
49 X = 4:Y = 5
50 BOARD(X,Y) = 2
51  GOSUB 126: REM  CALL BLOT
52 X = 5:Y = 4
53 BOARD(X,Y) = 2
54  GOSUB 126: REM  CALL BLOT
55 SC(2) = 2
56 TURN = 2
57  REM  BEGIN MAIN LOOP
58  FOR Q = 1 TO 100
59 TURN =  FN M2(TURN) + 1
60  COLOR= CC(TURN)
61  PRINT "SCORE IS:  WHITE ";SC(1);"  BLACK ";SC(2)
62  PRINT PROMPT$(TURN)
63  GOSUB 133: REM  CALL GETMOVE
64  IF PASS THEN 70
65  IF BOARD(X,Y) <  > 0 THEN 62
66  GOSUB 143: REM  CALL MOVES
67  IF FLAG = 0 THEN 62
68  IF ((SC(1) + SC(2)) = 64) THEN 71
69  IF ((SC(1) = 0) OR (SC(2) = 0)) THEN 71
70  NEXT Q
71  IF SC(1) > SC(2) THEN  PRINT "WHITE WINS!": GOTO 74
72  IF SC(1) < SC(2) THEN  PRINT "BLACK WINS!": GOTO 74
73  PRINT "IT'S A TIE!!"
74  PRINT "FINAL SCORE: WHITE ";SC(1);"  BLACK ";SC(2)
75  INPUT "WOULD YOU LIKE TO PLAY AGAIN?";A$
```

```
76  IF  LEFT$ (A$,1) = "Y" THEN 35
77  END
78  REM  SUBROUTINE TO DRAW OTHELLO BOARD
79  GR
80  COLOR= BC
81  FOR I = 0 TO 39
82  HLIN 1,39 AT I
83  NEXT I
84  COLOR= TC: REM  TITLE COLOR
85  REM  PLOT "OTHELLO"
86  REM  FIRST "O"
87  VLIN 1,5 AT 7
88  PLOT 8,1
89  PLOT 8,5
90  VLIN 1,5 AT 9
91  REM  NEXT "T"
92  HLIN 11,13 AT 1
93  VLIN 2,5 AT 12
94  REM  NEXT "H"
95  VLIN 1,5 AT 15
96  PLOT 16,3
97  VLIN 1,5 AT 17
98  REM  NEXT "E"
99  VLIN 1,5 AT 19
100  HLIN 20,21 AT 1
101  PLOT 20,3
102  HLIN 20,21 AT 5
103  REM  NEXT TWO "L"S
104  VLIN 1,5 AT 23
105  HLIN 24,25 AT 5
106  VLIN 1,5 AT 27
107  HLIN 28,29 AT 5
108  REM  FINALLY ANOTHER "O"
109  VLIN 1,5 AT 31
110  PLOT 32,1
111  PLOT 32,5
112  VLIN 1,5 AT 33
113  REM  NOW DO BOARD ITSELF
114  COLOR= DC: REM  BORDER COLOR
115  FOR I = 7 TO 39 STEP 4
116  HLIN 4,36 AT I
117  NEXT I
118  FOR I = 4 TO 36 STEP 4
119  VLIN 8,38 AT I
120  NEXT I
121  RETURN
122  REM  SUBR MAP FINDS SCREEN COORDS (XS,YS) GIVEN BOARD COORDS (X,Y)
123 XS = 1 + 4 * X
124 YS = 40 - 4 * Y
125  RETURN
126  REM  SUBR BLOT FILLS IN A SQUARE WITH THE CURRENT COLOR
127  GOSUB 122
128 X2 = XS + 2
129  HLIN XS,X2 AT YS
130  HLIN XS,X2 AT YS + 1
131  HLIN XS,X2 AT YS + 2
132  RETURN
133  REM  SUBR GETMOVE
134  INPUT MOVE$
135 PASS = 0
136  IF  LEFT$ (MOVE$,1) = "P" THEN PASS = 1: RETURN
137  IF  LEN (MOVE$) <  > 2 THEN 134
138 X =  ASC ( LEFT$ (MOVE$,1)) - 64
139  IF X < 1 OR X > 8 THEN 134
140 Y =  ASC ( RIGHT$ (MOVE$,1)) - 48
141  IF Y < 1 OR Y > 8 THEN 134
142  RETURN
143  REM  FIND AND EXECUTE MOVES
144 FLAG = 0
145 OP = 3 - TURN: REM  COLOR OF OPPONENT
146  FOR I = 1 TO 8
147 NR = 0
148 XN = X:YN = Y
149 XN = XN + DX(I):YN = YN + DY(I)
150  IF BOARD(XN,YN) = OP THEN NR = NR + 1: GOTO 149
```

```
151  IF (BOARD(XN,YN) = 0) OR (NR = 0) THEN 170
152  REM  IF WE GET HERE, CAPTURE IS POSSIBLE
153 FLAG = 1
154  COLOR= CC(TURN)
155  IF BOARD(X,Y) <  > 0 THEN 159
156  GOSUB 126: REM  CALL BLOT
157 BOARD(X,Y) = TURN
158 SC(TURN) = SC(TURN) + 1
159  FOR J = 1 TO NR
160 XN = XN - DX(I):YN = YN - DY(I)
161 BOARD(XN,YN) = TURN
162 XTEMP = X:YTEMP = Y
163 X = XN:Y = YN
164  GOSUB 126: REM  CALL BLOT
165 X = XTEMP:Y = YTEMP
166  PRINT  CHR$ (7)
167 SC(TURN) = SC(TURN) + 1
168 SC(OP) = SC(OP) - 1
169  NEXT J
170  REM
171  NEXT I
172  RETURN
```

Musical Duets

by Rick Brown

Music generated by the Apple II, without extra firmware, is usually limited to one voice. Here are two Applesoft programs which, with the help of an ordinary amplifier, add a new dimension to Apple music — harmony.

Anyone who has ever done any serious game-playing on the Apple II surely realizes how a catchy tune played through the Apple's speaker can enhance a program. A short machine language program is all that is needed to generate notes with a wide range of frequencies and durations. Such a tone-generating program is very nice, but it only generates one voice, which is to say, only one note at any given time can be played through the speaker. The usual way to acquire extra voices is to open the piggy bank and buy a music board or some other peripheral device designed for synthesizing music. For the serious music lover, it may be that nothing less will do. But can anything be done to satisfy the rest of us, whose standards (or finances) may not be as high? I chose to try to add, through software, a second voice to the Apple.

Now, before we go further, a little information about how a tone-generating program works is in order.The assembly language instruction LDA $C030 will toggle the Apple's speaker once every time it is executed, resulting in a little "click." Any sound whatsoever coming from the speaker is nothing but a series of such clicks, and the nature of the sound depends only on the interval of time between one click and the next. In the simplest case, this time interval is constant, and a steady, single-frequency, "pure" tone is generated. One convenient way to control the length of the pause between clicks is to use a "do-nothing" loop in the program, which generates a pause that is proportional to the number of times the loop is executed. The longer the pause between clicks, the lower the frequency of the resultant tone.

It occurred to me that it might be possible, by interleaving two such "do-nothing" loops, to superimpose one tone upon another and thus create the Apple's second voice. Consider two tones, one with a frequency of 500 Hz, and the other with a frequency of 300 Hz. To generate the first, we make the speaker click at intervals of 0.002s (s = seconds); that is, at these instants: 0,000s, 0.002s, 0.004s, 0.008s, 0.010s, etc.

Similarly, the 300 Hz tone would click at these instants: 0.0000s, 0.0033s, 0.0067s, 0.0100s, etc. Now, to generate both tones simultaneously, we should (it would seem) click the speaker at these instants: 0s, 0.002s, 0.0033s, 0.004s, 0.0067s, 0.008s, 0.01s, and so on. The problem of the two tones "clicking" at the same instant (e.g., at 0s and at 0.01s) is taken care of by a sort of "phase shift" inherent in the way the two "do-nothing" loops are interleaved.

Well, it all looks good on paper, and it might even work, were we using sinusoidally varying pulses instead of instantaneous clicks. But in fact, what results from the above technique is one of the most awful noises I've ever heard coming from the Apple speaker.

A More Promising Technique

All is not lost. There is another assembly language instruction, LDA $C020, which toggles not the speaker, but the cassette output. This produces a "click" on a cassette recording. Or, if the output jack is connected to an amplifier, an audible click is produced. This is the secret to the second voice. There are several ways to amplify the signal. Perhaps the simplest is to plug an external speaker into your cassette recorder, and set the recorder in the "record" mode. Then, any input to the microphone jack will be amplified through the external speaker. Alternatively, you could patch from the cassette output jack to the computer to the auxiliary input of a stereo set. This method will probably give you more control over volume and tone. Now, by clicking the Apple speaker at a fixed interval, and clicking the alternate speaker at a different fixed interval, we can produce two distinct simultaneous tones. The Apple now harmonizes with itself!

Making Music

The core of the programs presented here is a machine language routine which generates two simultaneous notes of different pitches (P1 and P2), and different durations (D1 and D2). These notes are stored in two tables: one contains the melody and the other contains the harmony. After a note (either melody or harmony) is completed, the routine fetches the next pitch and duration from the appropriate table, and plays the next note. When a duration of zero is encountered in either table, the song is considered to be complete, and the machine language routine terminates. A listing of this routine is given in figure 1.

For each note, the pitch and duration take up one byte apiece. Thus there are 256 variations of pitch, and 255 possible durations (recall that a duration of zero will end the song). The value of P (the pitch) is proportional to the time delay between two successive "clicks" of the speaker, so that the highest values of P will produce the lowest notes. Because of this, P should be considered proportional to the wavelength, rather than to the frequency, of the note.

Although we have 256 wavelengths to choose from, most of them produce notes which are "between the keys of a piano." In other words, in order to make use of the isotonic scale to which we are accustomed, and in which music is commonly written, we must use only twelve notes per octave, and discard those values of P which produce non-isotonic notes. The range of 256 wavelengths available to us covers exactly eight octaves. The maximum number of isotonic notes we can use is 8 × 12, or 96. (In practice, the number is limited still further, as explained below.)

The ratio of wavelengths of two consecutive notes on the isotonic scale is a constant 2 (1/12), or about 1.059, so that the ratio of wavelengths of two notes an octave apart is always 2:1. Thus wavelengths 128 and 64 are an octave apart, as are wavelengths 20 and 10, 2 and 1, and so forth. This fact imposes an obvious limitation on the higher notes.

Suppose we have a very high note — say of wavelength 4. The note one octave higher, then, has a wavelength of 2. Now, since the program uses only integers to represent wavelengths, it cannot generate the 11 isotonic notes between these two wavelengths (in fact, it can only generate one, corresponding to wavelength 3).

Another problem arising out of the use of integers for wavelengths is that the higher notes have an unavoidable tendency to go off-key. Suppose that the exact isotonic wavelength of a particular note (a low note, in this example) is calculated to be 154.43 on a scale from 1 to 256. This is rounded off to 154, creating a relative error of 0.29%. Consider now, a much higher note, whose exact wavelength is 15.43. This is rounded to 15, causing a much higher relative error of 2.8%, and it is this *relative* error (rather than the absolute error), which is detected by the ear.

Taking into account the limitations discussed earlier, I designed the program to use the lowest 65 isotonic notes available, covering a little more than five octaves, and using wavelengths from 6 to 256 (the latter wavelength is represented by zero in the routine). The highest notes are still a bit off-key, but generally they are rarely used and won't create much of a problem. As far as the durations of the notes are concerned, they remain, as far as the ear can tell, faithfully proportional to their numerical values, throughout the range from 1 to 255.

The two programs presented here can be used to play duets. However, the main purpose of the first program is to assemble the note tables from the data input by the user and to save the song on disk, while the second program is used only to load and play previously-recorded songs.

The Note-Table Assembler Program

This program provides an easy way to input a song, listen to it, edit it according to taste, and finally to save it on disk for later use. The song is input to the program through the use of DATA statements, which are typed in by the user each time the program is run. All such DATA statements must have line numbers greater than 696. The elements in these DATA statements will indicate the key signature (if any), the name and relative duration of each note, and the end of each part (melody or harmony) of the song. There are also special DATA elements which indicate that a particular part of the song is to be repeated. To facilitate the entry of these data, the notes are called by their alphabetic names (A,B,C,D,E,F,G) and converted by the program to the appropriate numerical values. The key signature, by default, determines whether a given note is to be played sharp, flat, or natural, but the signature may be overridden by appending the character "#" (sharp), "&" (flat), or "N" (natural) to the note's name.

Notes of different octaves are indicated by a single digit appended to the note name. If no such digit appears, octave 0 (zero) is assumed (this is the lowest octave which can be notated). Thus, G3 is one octave above G2, and D#1 is one octave above D#. The lowest letter-name within an octave is A, and the highest is G. Thus A2 is just a little above G1, while G#4 and A&5 designate the same note. A detailed description of the formats of the data elements follows:

1. *Key Signature (optional):* If the music is written in a key other than C, the first two data elements should indicate the key signature. The first element should consist of the word "SHARP" or "FLAT", and the second element should be a string consisting of the letter names (in any order) of the notes to be sharped or flatted. Example:

```
730 DATA FLAT,ADBE
```

2. *Note Names:* Each note name is an alphanumeric data item of the form XYM, where:

   X is one of the letters A, B, C, D, E, F, G, or R (rest)...

   Y is an optional character indicating sharp (#), flat (&), or natural (N). Any of these characters will override the key signature...

   M is a number from 0 to 9, indicating which octave the note belongs to. (However, the range within one song is limited to 65 notes, or about 5½ octaves.) M can be omitted if it equals zero.

   If X equals "R", then Y and M are omitted. Each note name must be followed by its note-duration.

3. *Note Duration:* This is a numerical quantity indicating the *relative* duration of the note that precedes it (the absolute duration will be calculated later). For example, if a quarter-note is given a duration of 1, then a half-note would have a duration of 2, etc. Example:

```
740 DATA F1,.5,F#1,1,R,2,BN,1.5
```

4. *Repeat Flags:* An asterisk followed by a single digit is a repeat flag. Repeat flags should be placed at the beginning and end of any segment of the song which is to be repeated. Repeat flags do not actually initiate a repetition, but merely

serve as pointers which the REPEAT keyword (see below) can refer to. The repeat flags marking the beginning and end of the segment must contain different digits. Example:

```
850 DATA G,3,*1,F,2,D,2,A,1,*2
```

5. *Repeat:* When the word REPEAT is used in a DATA statement, it indicates that all the notes between some pair of previous repeat flags are to be repeated. The two DATA elements following REPEAT must be single-digit integers indicating which two of the preceding repeat flags delimit the segment to be repeated. For example,

```
800 DATA REPEAT,2,5
```

will cause everything between flags *2 and *5 (including, possibly, other REPEATs) to be repeated, assuming flags *2 and *5 have occurred as previous DATA elements. A particular repeat flag may appear in several places without error; a REPEAT command referring to that flag will always use the most recent occurrence.

6. *END1:* In a duet, the data element "END1" must follow the first part (melody) of the song.

7. *Second Part:* Note names and durations for the second part (harmony) of the song must follow "END1", in the format indicated in 2 and 5. The key signature (if any) is still in effect and should not be repeated here.

8. *END2:* The DATA element "END2" must follow the second part (harmony) of the song.

The above format applies to duets. There is also an option for entering and playing 1-part solos. To do this, enter key signature, note names, note durations and REPEAT specifications for one part, as described above, but following the last note duration, enter the string "ENDSOLO" as the last data element. This will cause the same tune to be played through both speakers. Figure 2 has been included on disk under the name "BROWN NOTES" and can be EXEC'ed into the Note-Table Assembler program.

Running the Program

Before running the program as shown, you may find it necessary to change the value of M in line 10. HIMEM will be set to this value, which will be the highest byte occupied by the note tables, plus 1. The value shown in the listing is for a 48K system without DOS. Modify line 10 if necessary, then save the program on disk as shown (without any DATA statements).

Now, each time you load the program, type in the DATA statements according to the format explained above, remembering to give them line numbers higher than 696. Caution: for alphanumeric data, trailing blanks are considered to be part of the string, and may cause the data to be misinterpreted by the program. Avoid trailing blanks!

After all the necessary DATA statements have been entered, type "RUN". In a few seconds, you will see the prompt "TEMPO,KEY?" The tempo you input will be proportional to the *length* of the song, so that higher values will actually produce slower music. Notice that this is opposite from the usual interpretation of tempo. The tempo is multiplied by the relative note duration obtained from the DATA statement, the product is rounded to the nearest integer, and the final value is POKEd into the note table. So, for best results, you should input a tempo which, when multiplied by the note duration, always yields an integer (thus avoiding any rounding error). In no case may the product of the tempo and the relative note duration exceed 255. A product of 255 will produce a note about 3.0 seconds long. All other durations are proportionally shorter.

The KEY is an integer value (positive, negative, or zero) indicating how many semitones the song will be shifted up or down on the isotonic scale. Thus, for example, a key of 22 is one octave (12 semitones) higher than a key of 10. If the input key causes any note to fall outside the available range of 65 notes, an error message will be given.

After the tempo and key have been input, the program begins assembling the note tables. As the program processes the DATA statements, error or warning messages may be given, generated either by the program or by Applesoft. These messages are described in detail in table 1.

Program Commands

After the note tables are assembled, you will be prompted with a question mark. In response to this, you may type one of the following commands:

GO plays the song, in harmony and stereo, with as many repetitions as desired. (Be sure your amplifier is properly connected.)

SWAP causes parts 1 and 2 to switch speakers. Before this command is executed, part 1 plays through the Apple speaker, part 2 through your amplifier. Another SWAP will restore the original speakers.

CHANGE allows you to change the tempo and key, and reassemble the note tables.

EDIT lists the DATA statements and ends the program, allowing you to modify the song.

SAVE requests a song title, then saves the note tables on disk. Since the program uses the GET command to input the title, any characters may be input, including colons, commas, and quotes. A carriage return terminates the input and causes recording instructions to be displayed.

Table 1: Error/Warning Messages

MESSAGE	PROBABLE CAUSE
ILLEGAL QUANTITY ERROR	Tempo = 0
BAD SUBSCRIPT ERROR	Illegal note name in DATA statement
OUT OF DATA ERROR	No "END2", or no "ENDSOLO"
SYNTAX ERROR	Bad DATA statement format; data type mismatch
ERROR: KEY IS TOO HIGH	Key would cause notes to be outside of allowable range
ERROR: KEY IS TOO LOW	(same as above)
ERROR: TEMPO IS TOO LONG	Tempo * Relative Duration > 255 for some note
ERROR: INSUFFICIENT MEMORY FOR NOTE TABLES	DATA statements plus note tables take up too much memory
WARNING: PART X IS XXX UNITS SHORTER THAN PART X. SONG WILL END EARLY.	The sums of the durations obtained from the DATA statements do not match. Song will play up to the end of the shorter part.
WARNING: DURATIONS OF SOME NOTES WERE ROUNDED TO THE NEAREST INTEGER. TUNES MAY NOT BE SYNCHRONIZED.	Tempo * Relative Duration does not equal an integer for some note(s).

The Playback Program

After I wrote the program just described (the first version of which did not include the SAVE command), it occurred to me that you could spend a lot of time inputting a masterpiece, and lose it all when the computer was turned off. Of course, it's always possible to save the entire program, and thus preserve the DATA statements, but this can run into a lot of disk space if you make a habit of it. Another drawback of this method is that every time the program is reloaded, the note tables have to be re-assembled, a process which can take several minutes for long songs. With all this in mind, I added the SAVE feature to the note-table assembler program, and wrote another program whose sole purpose was to load and play previously recorded songs. Since this playback program loads note tables which are already assembled, we do not experience the delay associated with assembling, and of course a lot of time and tape is saved for anyone who wants to build up a library of songs.

As can be seen from the listing, line 10 of this program is the same as line 10 of the note-table assembler program. If necessary, modify this line as previously described before running the program.

In line 180, ET is set to the beginning address of the file BLOADed in line 130. The addresses PEEKed in line 180 are for a 48K system The correct addresses for a smaller system can be found on page 144 of the DOS 3.3 manual.

After typing "RUN", you will be prompted with a question mark. In response to the question mark, any of the following commands can be typed:

GO plays the song. Same as the GO command described earlier.

SWAP switches the speakers. Same as the SWAP command described earlier.

CAT prints a catalog of the files on the disk.

LOAD allows you to load and play another song from disk.

Note that there are no CHANGE or EDIT commands here; this is a "read-only" type program. When running the first program, then, you should be sure the tempo and key are adjusted to their most pleasing values before SAVEing the song.

A Sample Song

In figure 2, the DATA statements for a short song are given. This is a folk song entitled "Blue Bells of Scotland." The recommended tempo and key for this song are 30, 20. These DATA statements illustrate several techniques which come in handy when you're inputting a song:

1. Input one measure per DATA statement. This way, if you get a warning that the two parts are not of the same length, you can simply check each DATA statement until you find the measure that doesn't "add up." This technique also helps you to relate the DATA statements to the sheet music.

2. Choose note durations which will take the least amount of typing. In this example, quarter notes are represented by 1, and eighth notes by .5. If a song contains a preponderance of eighth notes, on the other hand, it might be wiser to represent eighth notes by 1, and quarter notes by 2, etc., so that you would not have to type in so many decimal points. This would simply require a corresponding adjustment in the TEMPO when the program is run.

3. Number the DATA statements so that a measure in the melody can be easily related to the corresponding measure in the harmony. In the example, DATA statements of corresponding measures have line numbers separated by 100.

The Applesoft programs described provide a convenient method for transferring a song from sheet music to the computer. However, the assembly language routine can be used independently, as long as note tables are created, and the pointers to the beginnings of the note tables are initialized. Thus it is possible to experiment with more exotic kinds of music, using all 256 wavelengths instead of just the 65 to which my note-table assembler is limited. CALL 777 will start the song playing. If the song is interrupted (as with a RESET), CALL 840 will cause it to pick up where it left off.

Figure 2: Blue Bells of Scotland

```
800 DATA G,1
801 DATA *1,C1,2,B1,1,A1,1
802 DATA G,2,A1,1,B1,.5,C1,.5
803 DATA E,1,E,1,F,1,D,1
804 DATA C,3,*2,G,1
805 DATA REPEAT,1,2,G,1
806 DATA E,1,C,1,E,1,G,1
807 DATA C1,2,A1,1,B1,.5,C1,.5
808 DATA B1,1,G,1,A1,1,F#,1
809 DATA G,2,A1,1,B1,1
810 DATA REPEAT,1,2
811 DATA END1
900 DATA R,1
901 DATA *3,R,1,E,1,*4,F,1,F,1
902 DATA E,2,F,2
903 DATA G,1,C,1,D,1,F,1
904 DATA E,3,*5,R,1
905 DATA REPEAT,3,5,R,1
906 DATA C1,3,D1,1
907 DATA A1,2,F,1,G,.5,A1,.5
908 DATA D1,2,C1,2
909 DATA B1,1,D1,1,G,1,F,1
910 DATA E,2,REPEAT,4,5
911 DATA END2
```

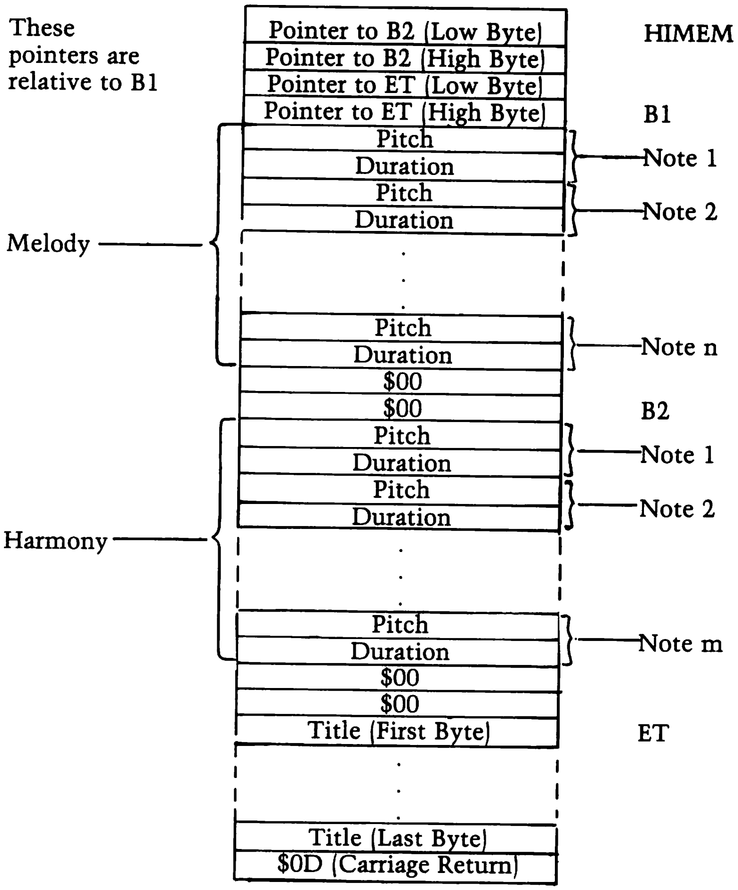
When you create the note tables "by hand", (without the aid of the note-table assembler program), follow the structure illustrated in figure 3, POKEing the first note into the *highest* memory location, and working your way down. The first pointer (decimal locations 773,774) should be set to the location of the first pitch of the first part, *plus one*. Similarly, the second pointer (decimal locations 775,776) should be set to the location of the first pitch of the *second* part, plus one. In the case of solos, the first part *is* the second part, so both pointers are set to the same location. By judicious placement of these pointers, you can play duets, play solos, create a short delay between the two speakers for an "echo" effect, or even "listen" to the computer's ROM. For another interesting effect, execute the following instruction:

```
POKE 835,80 - PEEK(835)
```

Then, when you do a CALL 777, both parts of the song will be sent through the same speaker. This will provide an excellent demonstration of why I chose to use two speakers instead of one.

Whether you use the machine language routine independently, or with the programs described in this article, or within your own BASIC programs, there is plenty of room for experimentation.

Figure 3: Structure of Note Tables for Duets



```
0800              ;************************
0800              ;*                      *
0800              ;*  2-TONE GEN. ROUTINE *
0800              ;*      RICK BROWN      *
0800              ;*                      *
0800              ;*      TONE GEN        *
0800              ;*                      *
0800              ;*  COPYRIGHT (C) 1982  *
0800              ;*    MICRO INK, INC.   *
0800              ;* CHELMSFORD, MA 01824 *
0800              ;*  ALL RIGHTS RESERVED *
0800              ;*                      *
0800              ;************************
0800              ;
0800              ;
0006              INDX1L    EPZ $06
0007              INDX1H    EPZ $07
0008              INDX2L    EPZ $08
0009              INDX2H    EPZ $09
0800              ;
0800              ;
0300              I         EQU $300
0301              P1        EQU $301
0302              D1        EQU $302
0303              P2        EQU $303
0304              D2        EQU $304
0305              I1L       EQU $305
0306              I1H       EQU $306
0307              I2L       EQU $307
0308              I2H       EQU $308
0800              ;
0800              ;
0309                        ORG $309
0309                        OBJ $800
0309              ;
0309 AD 05 03               LDA I1L           ;INITIALIZE
030C 85 06                  STA INDX1L        ;POINTERS
030E AD 06 03               LDA I1H           ;TO
0311 85 07                  STA INDX1H        ;BEGINNING
0313 AD 07 03               LDA I2L           ;ADDRESSES
0316 85 08                  STA INDX2L        ;OF
0318 AD 08 03               LDA I2H           ;NOTE
031B 85 09                  STA INDX2H        ;TABLES
031D A9 00                  LDA #$00
031F 8D 00 03               STA I
0322 20 60 03               JSR READ1         ;FETCH FIRST NOTE OF MELODY
0325 20 84 03               JSR READ2         ;FETCH FIRST NOTE OF HARMONY
0328 CA           LBL1      DEX
0329 F0 07                  BEQ TONE1
032B EA                     NOP               ;THESE TWO INSTRUCTIONS CAUSE
032C AD 11 11               LDA $1111         ;A 6-CYCLE TIME DELAY
032F 4C 38 03               JMP LBL2
0332              ;
0332 AD 30 C0     TONE1     LDA $C030         ;CLICK SPEAKER AFTER P1 LOOPS
0335 AE 01 03               LDX P1            ;RESET X-REGISTER
0338 88           LBL2      DEY
0339 F0 07                  BEQ TONE2
033B EA                     NOP               ;THESE TWO INSTRUCTIONS CAUSE
033C AD 11 11               LDA $1111         ;A 6-CYCLE TIME DELAY
033F 4C 48 03               JMP LBL3
0342              ;
0342 AD 20 C0     TONE2     LDA $C020         ;CLICK SPEAKER AFTER P2 LOOPS
0345 AC 03 03               LDY P2            ;RESET Y-REGISTER
0348 CE 00 03     LBL3      DEC I             ;AFTER 256 LOOPS, CHECK FOR EN
D OF NOTES
034B D0 DB                  BNE LBL1
034D CE 02 03               DEC D1            ;END OF MELODY NOTE?
0350 D0 03                  BNE LBL4          ;NO, CHECK HARMONY NOTE
0352 20 60 03               JSR READ1         ;YES, FETCH NEXT NOTE OF HARMO
NY
0355 CE 04 03     LBL4      DEC D2            ;END OF HARMONY NOTE?
0358 D0 CE                  BNE LBL1          ;NO, LOOP AGAIN
035A 20 84 03               JSR READ2         ;YES, FETCH NEXT NOTE OF HARMO
NY
035D 4C 28 03               JMP LBL1          ;THEN LOOP AGAIN
```

```
0360                ;
0360 A2 00          READ1    LDX #$00
0362 A5 06                   LDA INDX1L
0364 D0 02                   BNE LBL5
0366 C6 07                   DEC INDX1H
0368 C6 06          LBL5     DEC INDX1L
036A A1 06                   LDA (INDX1L,X)
036C 8D 01 03                STA P1
036F A5 06                   LDA INDX1L
0371 D0 02                   BNE LBL6
0373 C6 07                   DEC INDX1H
0375 C6 06          LBL6     DEC INDX1L
0377 A1 06                   LDA (INDX1L,X)
0379 8D 02 03                STA D1         ;DURATION OF MELODY NOTE
037C D0 02                   BNE LBL7
037E 68                      PLA            ;IF D1=0, POP RETURN ADDRESS
037F 68                      PLA            ;OFF STACK, SO RTS WILL END PR
OGRAM
0380 AE 01 03       LBL7     LDX P1
0383 60                      RTS
0384                ;
0384 A0 00          READ2    LDY #$00
0386 A5 08                   LDA INDX2L
0388 D0 02                   BNE LBL8
038A C6 09                   DEC INDX2H
038C C6 08          LBL8     DEC INDX2L
038E B1 08                   LDA (INDX2L),Y
0390 8D 03 03                STA P2         ;PITCH (WAVELENGTH) OF HARMONY
 NOTE
0393 A5 08                   LDA INDX2L
0395 D0 02                   BNE LBL9
0397 C6 09                   DEC INDX2H
0399 C6 08          LBL9     DEC INDX2L
039B B1 08                   LDA (INDX2L),Y
039D 8D 04 03                STA D2         ;DURATION OF HARMONY NOTE
03A0 D0 02                   BNE LBL10
03A2 68                      PLA            ;IF D2=0, POP RETURN ADDRESS
03A3 68                      PLA            ;OFF STACK, SO RTS WILL END PR
OGRAM
03A4 AC 03 03       LBL10    LDY P2
03A7 60                      RTS
03A8                         END
```

```
0  REM  ************************
1  REM  *                      *
2  REM  * NOTE-TABLE ASSEMBLER *
3  REM  *      RICK BROWN      *
4  REM  *                      *
5  REM  *  COPYRIGHT (C) 1982  *
6  REM  *    MICRO INK, INC.   *
7  REM  * CHELMSFORD, MA 01824 *
8  REM  *  ALL RIGHTS RESERVED *
9  REM  *                      *
10  REM ************************
11  REM
12  REM
13 M = 38400: REM  M=HIGHEST AVAILABLE ADDRESS
20 B1 = M - 4: HIMEM: M
30  DIM N%(65),P%(7),M(10),L(10)
40  DEF  FN HI(X) =  INT (X / 256)
50  DEF  FN LO(X) = X -  FN HI(X) * 256
55  REM   LOAD MACHINE LANGUAGE PROGRAM
60  PRINT  CHR$ (4)"BLOAD BROWN/TONE GEN.CODE"
120 N%(0) = 1:N%(1) = 0
125  REM  SET ISOTONIC WAVELENGTHS
130  FOR I = 2 TO 65
140 N%(I) = 256 / (2 ^ ((I - 1) / 12)) + .5
150  NEXT I
153  REM  ABCDEFG
155 P%(1) = 0:P%(2) = 2:P%(3) = 3:P%(4) = 5
156 P%(5) = 7:P%(6) = 8:P%(7) = 10
160 E = M -  FRE (0) - 65536 * ( FRE (0) < 0) + 200: HIMEM: E
165 D$ =  CHR$ (4)
170 B$ =  CHR$ (7) + "ERROR: "
180  RESTORE : INPUT "TEMPO,KEY? ";TM,K%:L = 0:F1 = 0
190  READ P$: IF P$ = "SHARP" OR P$ = "FLAT" THEN 680
200  RESTORE :LN = 0
210  FOR I = B1 - 1 TO E STEP  - 2
220  READ P$: IF  LEFT$ (P$,3) = "END" THEN 370
230  IF P$ = "R" THEN P = 0: GOTO 330
235  IF  LEFT$ (P$,1) = "*" THEN MK =  VAL ( MID$ (P$,2)):M(MK) = I:L(MK)
     = L: GOTO 220
237  IF  LEFT$ (P$,6) = "REPEAT" THEN 692
240 P = P%( ASC (P$) - 64) + 12 *  VAL ( RIGHT$ (P$,1)) + K%
250 A$ =  MID$ (P$,2,1)
255  IF A$ = "N" THEN 310
260  IF A$ = "#" THEN P = P + 1: GOTO 310
270  IF A$ = "&" THEN P = P - 1: GOTO 310
280  IF LN = 0 THEN 310
290  FOR J = 1 TO LN
295  IF  MID$ (SF$,J,1) =  LEFT$ (P$,1) THEN P = P + Q: GOTO 310
300  NEXT
310  IF P < 1 THEN  PRINT B$;"KEY IS TOO LOW": GOTO 180
320  IF P > 65 THEN  PRINT B$;"KEY IS TOO HIGH": GOTO 180
330  READ DD:L = L + DD:DD = DD * TM:D =  INT (DD + .5)
340  IF D > 255 THEN  PRINT B$;"TEMPO IS TOO LONG": GOTO 180
350  IF D <  > DD THEN F1 = 1
355  REM  POKE PITCH & DURATION INTO NOTE TABLE
360  POKE I,N%(P): POKE I - 1,D: GOTO 390
370  POKE I,0: POKE I - 1,0
375  IF  LEFT$ (P$,7) = "ENDSOLO" THEN B2 = B1:ET = I - 2:L2 = L1: GOTO 4
     00
380  IF  LEFT$ (P$,4) = "END2" THEN ET = I - 2:L2 = L - L1: GOTO 400
385 B2 = I - 1:L1 = L
390  NEXT I
395  PRINT B$;"INSUFFICIENT MEMORY": PRINT "FOR TUNE TABLES": HIMEM: M: END

400  POKE M - 1, FN LO(B1 - B2): POKE M - 2, FN HI(B1 - B2)
405  POKE M - 3, FN LO(B1 - ET): POKE M - 4, FN HI(B1 - ET)
410  IF L1 <  > L2 THEN SH = .5 * (3 -  SGN (L2 - L1)): PRINT : PRINT "WA
     RNING: PART ";SH;" IS "; ABS (L1 - L2);" UNITS SHORTER": PRINT "THAN
      PART ";3 - SH;". SONG WILL END EARLY."
420  IF F1 THEN  PRINT : PRINT "WARNING: DURATIONS OF SOME NOTES WERE": PRINT
     "ROUNDED TO THE NEAREST INTEGER.  TUNES": PRINT "MAY NOT BE SYNCHRON
     IZED."
430  POKE 773, FN LO(B1): POKE 774, FN HI(B1)
440  POKE 775, FN LO(B2): POKE 776, FN HI(B2)
450  PRINT : INPUT COM$
```

```
460  IF  LEFT$ (COM$,2) <  > "GO" THEN 500
470  INPUT "REPETITIONS? ";R
480  FOR I = 1 TO R
490  CALL 777: NEXT I: GOTO 450
500  IF  LEFT$ (COM$,6) = "CHANGE" THEN 180
510  IF  LEFT$ (COM$,4) = "EDIT" THEN  HIMEM: M: LIST 697,: END
515  IF  LEFT$ (COM$,4) = "SWAP" THEN  POKE 819,80 -  PEEK (819): POKE 83
     5,80 -  PEEK (835): GOTO 450
520  IF  LEFT$ (COM$,4) <  > "SAVE" THEN  PRINT "WHAT?": GOTO 450
530  PRINT "TITLE (1-30 CHARACTERS):"
535 FILE$ = ""
540  FOR I = 1 TO 31
550  GET P$: IF P$ =  CHR$ (8) THEN I = I + 1: PRINT " "; CHR$ (8); CHR$
     (8);: GOTO 550
552  IF P$ = "," THEN P$ = ";"
555  IF P$ =  CHR$ (21) THEN 550
557  IF P$ =  CHR$ (24) THEN  PRINT  CHR$ (92): GOTO 535
560  PRINT P$;: IF P$ =  CHR$ (13) THEN 580
565 FILE$ = FILE$ + P$
570  NEXT I: PRINT : PRINT B$;"TITLE TOO LONG": GOTO 530
580  PRINT D$"BSAVE ";FILE$;",A";ET;",L";M - ET
590  PRINT D$"LOCK ";FILE$
600  GOTO 450
680 Q = 1: IF P$ = "FLAT" THEN Q =  - 1
690  READ SF$:LN =  LEN (SF$): GOTO 210
692  READ M1,M2: IF I + M(M2) - M(M1) < E THEN 395
694  IF M(M2) >  = M(M1) THEN 220
696  FOR K = M(M1) TO M(M2) + 1 STEP  - 1: POKE I + K - M(M1), PEEK (K): NEXT
     :I = I + K - M(M1):L = L + L(M2) - L(M1): GOTO 220
```

```
1   REM  ************************
2   REM  *                      *
3   REM  *    MUSICAL DUETS     *
4   REM  *      RICK BROWN      *
5   REM  *                      *
6   REM  *  COPYRIGHT (C) 1982  *
7   REM  *    MICRO INK, INC.   *
8   REM  * CHELMSFORD, MA 01824 *
9   REM  *  ALL RIGHTS RESERVED *
10   REM *                      *
11   REM ************************
12 M = 38400: REM  MUST BE SAME ADDRESS AS IN ASSEMBLER-PROGRAM
15   REM  LOAD MACHINE LANGUAGE PROGRAM
60   PRINT  CHR$ (4)"BLOAD BROWN/TONE GEN.CODE"
80   DEF  FN HI(X) =  INT (X / 256)
90   DEF  FN LO(X) = X -  FN HI(X) * 256
95   HOME : GOTO 240
100  HIMEM: M:B1 = M - 4
110  PRINT
120  INPUT "TITLE? ";FILE$
130  PRINT  CHR$ (4);"BLOAD ";FILE$
150 B2 = B1 - ( PEEK (M - 1) + 256 *  PEEK (M - 2))
170 T = B1 - ( PEEK (M - 3) + 256 *  PEEK (M - 4))
180 ET =  PEEK (43634) +  PEEK (43635) * 256: REM  CONTAINS BEGINNING ADD
     RESS OF FILE$(FOR 48K SYSTEM)
190  HIMEM: ET
220  POKE 773, FN LO(B1): POKE 774, FN HI(B1)
230  POKE 775, FN LO(B2): POKE 776, FN HI(B2)
240  PRINT : INPUT COM$
250  IF COM$ <  > "GO" THEN 280
260  INPUT "REPETITIONS? ";R
270  FOR I = 1 TO R: CALL 777: NEXT I: GOTO 240
280  IF COM$ = "LOAD" THEN 100
290  IF COM$ <  > "SWAP" THEN 330
300  POKE 819,80 -  PEEK (819): POKE 835,80 -  PEEK (835)
310  GOTO 240
330  IF COM$ <  > "CAT" THEN  PRINT "WHAT?": GOTO 240
340  PRINT  CHR$ (4)"CATALOG": GOTO 240
```

Language Index

Author Index
(Biographies included)

Mossberg, Sanford M. 100
Although a physician by profession, also a programming enthusiast, and columnist for a club newsletter.

Paris, Greg 164
Involved in neurobiology research; interested in programming microcomputer-based instrumentation.

Partyka, David 186
Works as a programmer on an IBM 3031 OS for the May Department Stores Co.

Radcliffe, Art 131
Has acquired 32 patents in computer and communication circuits and systems while working for IT & T, Radiation Inc., and Burroughs over 25 years. Also worked in optics and holography.

Rosenberg, David L. 39
Analyst with the Management Sciences department of Holiday Inns, Inc. Has been in computer field for nine years.

Taylor, Charles F., Jr. 196
Member of the faculty at the Naval Postgraduate School in Monterey, CA, where he teaches Operations Research and Computer Science.

Vile, Richard C., Jr. 137
Manager of a Software Technology group for Bell Northern Research, in Ann Arbor, MI.

Woodward, Kim G. 157
Works as a computer scientist for the U.S. Coast Guard in Washington, D.C.

```
A 002 HELLO
B 003 DISASSMB
B 003 DISASSMB-SC
B 006 CROSS ASSEMBLER
B 006 CROSS-SLOT-ZERO
B 006 FIAT
A 003 FIAT-LOADER
A 002 ERROR1-CALLER
B 002 ERROR1
B 002 ERROR2
A 008 AMPER-SEARCH1
A 010 AMPER-SEARCH2
B 005 AMPER-SEARCH
B 002 LINE FINDER
A 016 VARIABLE LISTER
B 002 APPLESOFT VARIABLE LISTER OBJ
B 002 SHELL-METZNER SORT
A 031 STOCK HOLDINGS MGR
A 039 SOLAR SYSTEM SIMULATION
A 014 OTHELLO
B 002 TONE GEN
A 015 NOTE TABLE ASSEMBLER
T 004 BROWN NOTES
A 006 MUSICAL DUETS PLAYBACK
B 002 COLOR FILTER
A 003 COLOR FILTER DEMO
A 005 LISSAJOUS FIGURES
A 011 NOISY COASTER
B 004 APPLE-BITS
I 004 FIREWORKS
B 010 SPARKS
B 010 LARGE LETTERS
I 016 PATTERN MAKER
I 007 LARGE DRIVER
I 006 RANDOM WALK
B 003 LITTLE MEN
B 003 TRAIN
I 006 LOCOMOTIVE
B 002 GI
B 003 GETLNA
A 031 TRICK DOS
A 042 LACRAB
A 006 LACRAB INSTRUCTIONS
T 002 LIST
A 012 BFILE PARAMETER LIST
```

Warranty

MICRO on the Apple

Although we've worked to create as perfect a diskette as possible, including hiring a reputable, reliable disk manufacturer to copy the diskettes, there is no guarantee that this diskette is error-free.

To cover the few instances of defective diskettes, we are providing the following warranty (this card must be filled out and returned to MICRO INK, Inc., immediately after purchase):

If within one month of purchase you find your diskette is defective, return the diskette to MICRO, along with $1.00 to cover shipping and handling charges.

If after one month of purchase, but within no time limit, this diskette proves defective, return it to MICRO with $6.00 to cover replacement cost, shipping and handling.

Your date of purchase must be validated by your dealer; if purchased directly from MICRO, the valid date appears on this card.

Defective diskettes must be returned to MICRO to enable our quality assurance personnel to test and check the diskette. We need to know what caused the defect to avoid similar problems in the future.

We recommend that you try LOADing or BLOADing each program on the diskette immediately after purchase to ensure that the diskette is not defective.

Signature — Date of purchase — (Volume 2)

Address (please print):

Name

Street

City — State/Province/Country — Code

Other Products from MICRO

In addition to the MICRO on the Apple series, MICRO INK, Inc., produces several other products, including MICRO magazine, a monthly journal which reports on new 6502/6809 microprocessor family applications, systems, and developments. Other books published include the Best of MICRO series (anthologies of some of the best general-interest articles from MICRO), and *What's Where in the Apple*, (a detailed Atlas and memory map for the Apple II computer).

Ask your dealer for MICRO, or subscribe by completing this form:

	Yearly Rates Surface	(U.S Dollars) Air Mail
United States	$24.00	n/a
Canada	27.00	n/a
Europe	27.00	$42.00
Mexico, Central America Middle East, North Africa Central Africa	27.00	48.00
South America, South Africa Far East, Australasia	27.00	72.00

MICRO Books

	At Your Dealer	Ordered by Mail Surface	Ordered by Mail Air Mail (Not U.S./Canada)
Best of MICRO, Vol. 1	$ 6.00	$ 8.00	$12.00
Best of MICRO, Vol. 2	8.00	10.00	15.00
Best of MICRO, Vol. 3	10.00	12.00	18.00
What's Where in the Apple	14.95	16.95	19.95
MICRO on the Apple (each volume)	24.00		

Note: Circle desired item.

Subscription rates are subject to change without notice. These prices are current as of January 1982.

- ☐ Check enclosed for $ ____________
- ☐ Bill VISA
- ☐ Bill MasterCard

____

Signature Card Number Expires

Please print

____

Name

____

Street

____

City State/Province/Country Code

Notice to Purchaser

When this book is purchased, this pocket should contain

A. One floppy disk entitled *MICRO on the Apple, Volume 3.*
B. A warranty card pertaining to the disk.

If either is missing, make sure you ask the seller for a copy.

MICRO on the Apple, Volume 3

19 Articles with More Than 40 Programs on Diskette!

MICRO INK, publisher of *MICRO, The 6502/6809 Journal*, is pleased to bring you *MICRO on the Apple*, Volume 3, the third in a series of books for the Apple.

Volume 3 offers interesting and useful programming aids, applications, and reference material, along with entertainment. Chapter titles include Applesoft Aids, Machine-Language Aids, I/O Enhancements, Graphics, Tutorial/Reference, and Recreation/Applications. The accompanying diskette is 16-sector, DOS 3.3 format.

Preliminary article selections were made by Ford Cavallari, Editor of Volumes 1 and 2. Final article selection, program testing and modifications were made by Apple expert Tim Osborn, who serves as an Editor on the staff of MICRO. Many authors provided their own updates and improvements.

$24.95 in U.S./Canada
(Including floppy disk)

ISSN 0275-3537
ISBN 0-938222-08-2

MICRO INK, Inc.
P.O. Box 6502
Chelmsford, Massachusetts 01824