$1.20

Apple Assembly Line

Volume 1 -- Issue 12 September, 1981

In This Issue...

Quarterly Disk #4

The fourth Quarterly Disk is now ready, containing all the source code from issues 10 through 12. The cost is only $15, and it will save you a lot of typing and possible searching for typos. All previous Quarterly Disks are still available, at the same price.

Renewing subscriptions

The 4-digit number in the upper right corner of your mailing label is the expiration date of your subscription. The first two digits are the year, and the last two digits are the month of the last issue you have paid for. If it says "8109", this is your last issue. Unless, of course, I receive your renewal check for $12. If your label says 8111 or less, now is the time to renew!

More about the Firmware Card in Slot 4

Michael Sanders' DOS patch for using the Firmware card in slot 4 is really nice. A lot of you have written or called about it, and I use it myself now. In fact, I have changed my HELLO programs to do the patch. All it takes is two POKEs:

```
10 POKE 42424,192 : POKE 42432,193
```

I like doing it this way a lot better than INITting a disk with a modified DOS. If you want to test for the presence of a card before patching, you can do it like this:

```
10  FOR I = 768 TO 817: READ A: POKE I,A: NEXT : CALL 768
20  DATA 173,192,192,162,2,189,0,224,221,44,3,208,16,202,16,245,1
     62,192,142,184,165,232,142,192,165,173,193,192,96,162,2,189,
     0,224,221,47,3,208,242,202,16,245,48,228,32,0,240,76,40,241
30  TEXT : HOME : PRINT  CHR$ (4)"CATALOG"
```

Field Input Routine for Applesoft.....Bob Potts

Inputting strings to an Applesoft program is normally a simple task. What could be easier than "INPUT A$"? But, that method will not allow commas or colons.

Another easy way is to use GET C$ for each character, and append them to a string using A$=A$+C$. But, by the time you add the testing for each input character to find the end of input and other possible control characters, the routine can be terribly slow. Furthermore, it eats up string space like crazy; eventually Applesoft garbage collection starts, and the program dies for a while. Here is the kind of loop I am talking about:

```
10 A$=""
20 GET C$
30 <perform various tests on C$>
40 A$=A$+C$:PRINT C$;
50 GO TO 20
```

As the string increases in length, the speed decreases dramatically. In fact, some characters may be lost if you are a fast typist.

One way to correct this is to use a machine language routine to input each keystroke, test it, and build a string for the Applesoft program. Such a routine was printed in "Apple Assembly Line" issue #7 (April, 1981), pages 6-8. But that routine used the monitor's RDLINE subroutine to input the string. I needed a routine more adapted to inputting a series of fields, using the screen in a "fill-in-the-blanks" mode.

The following program was designed for use in the various branches of the Bank of Louisville. The Apple is used to calculate loans, print the installment notes, and to enter loan applications. A loan application involves filling in the blanks on several screens full of prompts.

To use the input routine, you first position the cursor to the start of field using VTAB and HTAB; then set a field length using the SCALE= statement, and a field code using the ROT= statement. The actual call to the input routine is done with "&INPUT" and the name of your string. Here is an example for inputting a 5-character field starting in column 10 of line 7:

```
10 VTAB 7 : HTAB 10 : SCALE=5 : ROT = 0 : &INPUT A$
```

The input routine allows skipping from field to field, either forward or backward through a form. Backspace and copy (right arrow) are supported. Filling up a field, or hitting RETURN within a field, finish that field and return the value to Applesoft. An EXIT CODE tells the Applesoft program whether a value was returned in the string or some other exit was chosen. You access the exit code with a PEEK(224). Here are the four exit codes and their meanings:

```
EXIT CODE = 0     Field was filled or RETURN typed.
          = 1     ESCAPE was typed at beginning of field.
          = 2     CTRL-F was typed at beginning of field.
          = 3     Left Arrow (backspace) was typed
                  at beginning of field.
```

If the exit code is zero, then the field data you typed is in your string. Otherwise, the string's value is not changed. Finishing a field by either filling it up or hitting RETURN puts the field data into your string, and I then advance to the next field on the form. I use an exit code of 3 (backspace at beginning of field) to mean that the Applesoft program should go back to the previous field on the current form.

How you use the exit codes of 1 and 2 is up to you. You might use an ESCAPE (exit code = 1) to abort the form-filling and return to a main menu. The ESCAPE is now only recognized if you are at the beginning of the field and the field code is non-zero. Of course, you could change that. You might use the control-F to mean you are finished with the current form.

How Does It Work?

Line 1110 sets the origin to $0300. If you already have something else in page 3, you can change the origin to whatever suits your fancy. Just remember to set the correct values for HIMEM and LOMEM to protect it from Applesoft, and vice versa.

Lines 1380-1440 install the ampersand vector. If you BRUN the program, this code is executed. If you BLOAD it, then CALL 768 will execute it. You only have to execute this once in your program. Once done, any occurrence of an ampersand statement in your program will branch to INPUT.FIELD, at line 1460.

Lines 1460-1500 check for the keyword "INPUT", and a string variable name. The three routines (and others used in this program) starting with "AS." are in the Applesoft ROMs. AS.SYNCHR compares the current character with what is in the A-register; if different you get SYNTAX ERROR, and if the same the character pointer is advanced. AS.PTRGET scans a variable name and finds its descriptor in memory. AS.CHKSTR makes sure that the variable is a string (if not you get TYPE MISMATCH). At this point the address of the string descriptor is in $83,84. The address in $83,84 points to 3 bytes which tell the length and address of the string's contents.

Lines 1520-1690 test the input character and branch accordingly. I use MON.RDKEY to read the character, which means that the data could come from any I/O slot as well as the normal Apple Keyboard. You could add more tests here, or remove some. If it is a printing character, we fall into lines 1730-1810 to store the character in the input buffer and on the screen. If the filed is now full, line 1810 jumps to the routine which passes the data to Applesoft. Note that characters stored in the input buffer have the high-bit equal to zero (Applesoft likes them that way). Characters written on the screen have the high-bit set to one, so that they print in NORMAL video.

Lines 1920-1990 handle the backspace character. If you are at the beginning of a field, the routine will return with an exit code of 3. Otherwise, the current character will be replace on the screen with an underline character, and the cursor will be backed up.

Lines 2030-2050 handle the right arrow. Normally this just copies over a character on the screen. Characters are picked up from the screen image, and the treated just as though they came from the keyboard. Note that the right arrow will not advance over an underline character.

Lines 2090-2140 handle ESCAPE. As I mentioned earlier, ESCAPE is ignored unless it is typed when the cursor is at the beginning of the field, and the field code is non-zero. This is the only use for the field code in the input routine presented here, but you might think of many more uses and make your own modifications.

Lines 2180-2190 make Applesoft allocate some space for the string in the normal string data space. Then lines 2200-2270 set up the string variable's descriptor to point to this space. Lines 2280-2310 move the string data from the input buffer up to the new place. This code was copied from the "Fast String Input Routine" in AAL #7.

The input routine is presented here in a very simple form; I leave it up to you to modify it to suit your most demanding applications.

```
                        1000 *-------------------------------
                        1010 *       FIELD INPUT SUBROUTINE
                        1020 *       ----------------------
                        1030 *       BY ROBERT W. POTTS
                        1040 *       BANK OF LOUISVILLE
                        1050 *       P. O. BOX 1101
                        1060 *       LOUISVILLE, KY 40201
                        1070 *
                        1080 *       MODIFIED BY BOB SANDER-CEDERLOF
                        1090 *       FOR THE "APPLE ASSEMBLY LINE"
                        1100 *-------------------------------
                        1110         .OR $300
                        1120 *-------------------------------
  0024-                 1130 MON.CH      .EQ $24        MONITOR HORIZONTAL
  0028-                 1140 MON.BASL    .EQ $28
  0071-                 1150 SPC.PNTR    .EQ $71,72
  0083-                 1160 STR.PNTR    .EQ $83,84
                        1170 *-------------------------------
  00E1-                 1180 CT          .EQ $E1        CHARACTER COUNT
  00E7-                 1190 FL          .EQ $E7        FIELD LENGTH   (SET BY "SCALE=FL")
  00F9-                 1200 FLDCOD      .EQ $F9        FIELD CODE     (SET BY "ROT=FC")
  00E0-                 1210 EXITCODE    .EQ $E0        PEEK (224) TO SEE EXIT CODE
                        1220 *-------------------------------
  0200-                 1230 INPUT.BUFFER .EQ $0200
  03F5-                 1240 AMPER.VECTOR .EQ $03F5
                        1250 *-------------------------------
  FD0C-                 1260 MON.RDKEY   .EQ $FD0C      MONITOR CHAR INPUT
  FDED-                 1270 MON.COUT    .EQ $FDED
  FC10-                 1280 MON.BS      .EQ $FC10      MONITOR BACKSPACE
                        1290 *-------------------------------
  DD6C-                 1300 AS.CHKSTR   .EQ $DD6C
  DEC0-                 1310 AS.SYNCHR   .EQ $DEC0
  DFE3-                 1320 AS.PTRGET   .EQ $DFE3
  E452-                 1330 AS.GETSPA   .EQ $E452
  E5E2-                 1340 AS.MOVSTR   .EQ $E5E2
                        1350 *-------------------------------
                        1360 *  SET UP AMPERSAND VECTOR
                        1370 *-------------------------------
  0300- A9 4C           1380 SETUP   LDA #$4C      JMP OPCODE
  0302- 8D F5 03        1390         STA AMPER.VECTOR
  0305- A9 10           1400         LDA #INPUT.FIELD
  0307- 8D F6 03        1410         STA AMPER.VECTOR+1
  030A- A9 03           1420         LDA /INPUT.FIELD
  030C- 8D F7 03        1430         STA AMPER.VECTOR+2
  030F- 60              1440         RTS
                        1450 *-------------------------------
                        1460 INPUT.FIELD
  0310- A9 84           1470         LDA #$84      "INPUT" TOKEN
  0312- 20 C0 DE        1480         JSR AS.SYNCHR  REQUIRE "INPUT" OR SYNTAX ERROR
  0315- 20 E3 DF        1490         JSR AS.PTRGET  GET STRING VARIABLE
  0318- 20 6C DD        1500         JSR AS.CHKSTR  REQUIRE STRING OR MISMATCH
                        1510 *-------------------------------
  031B- A9 00           1520         LDA #0        ZERO OUT CHARACTER COUNT
  031D- 85 E1           1530         STA CT
  031F- 20 0C FD        1540 .1      JSR MON.RDKEY    GET CHARACTER
  0322- 29 7F           1550 .2      AND #$7F      APPLESOFT STYLE
  0324- C9 06           1560         CMP #$06      CONTROL-F?
  0326- F0 2C           1570         BEQ .3        YES
  0328- C9 08           1580         CMP #$08      BACKSPACE?
  032A- F0 30           1590         BEQ .4        YES
```

```
032C- C9 0D     1600          CMP #$0D       RETURN?
032E- F0 53     1610          BEQ .7         YES, END OF FIELD
0330- C9 15     1620          CMP #$15       RIGHT ARROW?
0332- F0 3C     1630          BEQ .5         YES
0334- C9 1B     1640          CMP #$1B       ESCAPE?
0336- F0 3F     1650          BEQ .6         YES
0338- C9 20     1660          CMP #$20       SOME OTHER CONTROL CHARACTER?
033A- 90 E3     1670          BCC .1         YES, IGNORE IT
033C- C9 5B     1680          CMP #$5B       ACCEPTABLE PRINTING CHARACTER?
033E- B0 DF     1690          BCS .1         NO, IGNORE IT
                1700 *--------------------------------
                1710 *  GOT PRINTING CHARACTER - STORE IT
                1720 *--------------------------------
0340- A4 E1     1730          LDY CT         CHARACTER COUNTER
0342- 99 00 02  1740          STA INPUT.BUFFER,Y  STORE IN STRING
0345- 09 80     1750          ORA #$80       TURN ON HIGH BIT
0347- 20 ED FD  1760          JSR MON.COUT PRINT CHARACTER
034A- E6 E1     1770          INC CT         INCREMENT CHARACTER COUNT
034C- A5 E1     1780          LDA CT
034E- C5 E7     1790          CMP FL         IS FIELD FILLED UP?
0350- D0 CD     1800          BNE .1         NO, GET ANOTHER CHARACTER
0352- F0 2F     1810          BEQ .7         ...ALWAYS
                1820 *--------------------------------
                1830 *  HANDLE CONTROL-F
                1840 *--------------------------------
0354- A5 E1     1850 .3       LDA CT         ON FIRST CHARACTER?
0356- D0 C7     1860          BNE .1         NO, GET ANOTHER CHARACTER
0358- A9 02     1870          LDA #2         EXIT CODE = 2
035A- D0 45     1880          BNE .8         ...ALWAYS
                1890 *--------------------------------
                1900 *  HANDLE BACKSPACE
                1910 *--------------------------------
035C- A9 03     1920 .4       LDA #3         EXIT CODE = 3 IF IN 1ST CHAR
035E- C6 E1     1930          DEC CT         DECREMENT CHARACTER COUNTER
0360- 30 3F     1940          BMI .8         ON FIRST POSITION
0362- 20 10 FC  1950          JSR MON.BS     BACKSPACE
0365- A9 DF     1960          LDA #$DF       UNDERLINE
0367- 20 ED FD  1970          JSR MON.COUT PRINT IT
036A- 20 10 FC  1980          JSR MON.BS     BACKSPACE AGAIN
036D- 4C 1F 03  1990          JMP .1         DO AGAIN
                2000 *--------------------------------
                2010 *  HANDLE RIGHT ARROW
                2020 *--------------------------------
0370- A4 24     2030 .5       LDY MON.CH     YES, GET NEXT CHARACTER FROM SCREEN
0372- B1 28     2040          LDA (MON.BASL),Y
0374- 4C 22 03  2050          JMP .2
                2060 *--------------------------------
                2070 *  HANDLE ESCAPE
                2080 *--------------------------------
0377- A5 F9     2090 .6       LDA FLDCOD     FIELD CODE = 0?
0379- F0 A4     2100          BEQ .1         YES, GET ANOTHER CHARACTER
037B- A5 E1     2110          LDA CT
037D- D0 A0     2120          BNE .1         NO, GET ANOTHER CHARACTER
037F- A9 01     2130          LDA #1         EXIT CODE = 1
0381- D0 1E     2140          BNE .8         ...ALWAYS
                2150 *--------------------------------
                2160 *  STORE THE INPUT DATA IN THE STRING
                2170 *--------------------------------
0383- A5 E1     2180 .7       LDA CT         STRING LENGTH
0385- 20 52 E4  2190          JSR AS.GETSPA  GET SPACE IN STRING AREA
0388- A0 00     2200          LDY #0         MOVE DATA INTO VARIABLE
038A- 91 83     2210          STA (STR.PNTR),Y  LENGTH
038C- A5 71     2220          LDA SPC.PNTR
038E- C8        2230          INY
038F- 91 83     2240          STA (STR.PNTR),Y  LO-BYTE OF ADDRESS
0391- A5 72     2250          LDA SPC.PNTR+1
0393- C8        2260          INY
0394- 91 83     2270          STA (STR.PNTR),Y  HI-BYTE OF ADDRESS
0396- A2 00     2280          LDX #INPUT.BUFFER
0398- A0 02     2290          LDY /INPUT.BUFFER
039A- A5 E1     2300          LDA CT         LENGTH
039C- 20 E2 E5  2310          JSR AS.MOVSTR
039F- A9 00     2320          LDA #0         EXIT CODE = 0
03A1- 85 E0     2330 .8       STA EXITCODE
03A3- 60        2340          RTS
```

Here is a brief sample showing how you might use the input routine to fill in five fields:

```
10  PRINT  CHR$ (4)"BRUN B.INPUT ROUTINE"
20  DIM V(5),H(5),L(5),T$(5),A$(5)
30  FOR I = 1 TO 5: READ V(I),H(I),L(I),T$(I): NEXT
40  DATA 5,7,30,NAME
50  DATA 7,7,3,AGE
60  DATA 7,27,3,WEIGHT
70  DATA 9,7,12,STATE
80  DATA 9,27,5,ZIP
90  TEXT : HOME : VTAB 23: INVERSE : PRINT " TYPE CTRL-F WHEN FOR
     M FINISHED ": NORMAL
100  FOR I = 1 TO 5: VTAB V(I): HTAB H(I) -  LEN (T$(I)) - 1: PRINT
     T$(I)" ";: FOR J = 1 TO L(I): PRINT  CHR$ (95);: NEXT : NEXT

110 I = 1
120  VTAB V(I): HTAB H(I): SCALE= L(I): ROT= 0
130  &  INPUT A$(I):XC =  PEEK (224)
140  ON XC + 1 GOTO 200,300,400,500
200 I = I + 1: IF I > 5 THEN 110
210  GOTO 120
300  END : REM ESCAPE
400  REM CONTROL-F
410  HOME : FOR I = 1 TO 5: PRINT A$(I): NEXT : END
500  REM BACKSPACE
510 I = I - 1: IF I = 0 THEN I = 5
520  GOTO 120
```

CHRGET and CHRGOT in Applesoft

On pages 13 and 14 of the September 1981 Kilobaud Microcomputing (Robert Baker's Pet-Pourri column) there is a good description of the CHRGET/CHRGOT duo. These two subroutines (really two entry points into one routine) seem to be common to the Microsoft Basics, at least the 6502 versions.

What are they? When Applesoft initializes itself one of the tasks is to copy a short subroutine into page zero, from $00B1 through $00C8. There is no difference between the PET and the Apple versions, except that the PET version is copied into $0070-0087. Here is the code:

```
                     1000 *-------------------------------
                     1010 *      APPLESOFT CHRGET/CHRGOT SUBROUTINES
                     1020 *-------------------------------
                     1030        .OR $00B1
                     1040 *-------------------------------
00B8-                1050 TXTPTR .EQ $B8       INSIDE 'LDA' INSTRUCTION
                     1060 *-------------------------------
00B1- E6 B8          1070 CHRGET INC TXTPTR    INCREMENT ADDRESS OF NEXT CHARACTER
00B3- D0 02          1080        BNE CHRGOT
00B5- E6 B9          1090        INC TXTPTR+1
                     1095 *-------------------------------
00B7- AD 88 88       1100 CHRGOT LDA $8888     PICK UP THE NEXT CHARACTER
00BA- C9 3A          1110        CMP #$3A      TEST IF COLON
00BC- F0 0A          1120        BEQ .1        YES, Z AND C SET, RETURN
00BE- C9 20          1130        CMP #$20      TEST IF BLANK
00C0- F0 EF          1140        BEQ CHRGET    YES, IGNORE IT
00C2- 38             1150        SEC           DO DIGIT TEST
00C3- E9 30          1160        SBC #$30
00C5- 38             1170        SEC           SET Z IF VALUE WAS $00 (EOL TOKEN)
00C6- E9 D0          1180        SBC #$D0      AND CLEAR CARRY IF DIGIT ($30-39)
00C8- 60             1190 .1     RTS
```

Almost every time Applesoft wants to look at a character from your program or even from the input buffer, it does so by calling this subroutine. The CHRGET entry increments the address used to pick up the next character, and then falls into CHRGOT. In either case, the character is picked up and several tests are performed. Blanks are passed over, ignored. Colon (end of statement) and $00 (end of line) set the Z status bit. Digits clear CARRY, non-digits set CARRY. The calling program can use these status bits. For example:

```
    JSR CHRGET
    BEQ END         BRANCH IF COLON OR END-OF-LINE
    BCC DIGIT       BRANCH IF CHAR IS DIGIT (0-9)
```

The article in Kilobaud suggests patching this routine at $00BA to jump to your own code. Your program can trap certain characters for special functions, in much the same way as the "&" is now handled by Applesoft. You just have to be sure that you execute the instructions your JMP overlayed before returning to the remainder of CHRGET. It appears that many of the enhancement packages available for PET Basic use this scheme.

Why use this patching scheme instead of the "&" for special functions? Because your special functions can be made to appear an integral part of the language, without the telltale ampersand. Because even special codes inside expressions or other statements can be trapped. Because you want to encode or otherwise obfuscate your program for security. Because you just want to be different. Of course, the disadvantage is that the entire operation of Applesoft is slowed down by the amount of time your extra testing takes, since every character retrieved by the interpreter will go through your routine as well as the standard CHRGET.

```
                     1000 *-------------------------------
                     1010 *       SAMPLE APPLESOFT FILTER PROGRAM
                     1020 *-------------------------------
                     1030         .OR $BA
00BA- 4C 00 03       1040         JMP FILTER
                     1050 *-------------------------------
                     1060         .OR $300
0300- C9 23          1070 FILTER  CMP #'#       CHECK FOR "#" CHARACTER
0302- D0 06          1080         BNE .1        NO, PASS UNMOLESTED
0304- 20 12 03       1090         JSR WHATEVER.YOU.WANT
0307- 4C B1 00       1100         JMP $B1
030A- C9 3A          1110 .1      CMP #$3A      CHECK FOR COLON
030C- B0 03          1120         BCS .2        YES, RETURN JUST CHRGET WOULD
030E- 4C BE 00       1130         JMP $BE       NO, RECONNECT WITH CHRGET
0311- 60             1140 .2      RTS
                     1150 *-------------------------------
                     1160 WHATEVER.YOU.WANT
0312- 20 E2 FB       1170         JSR $FBE2     RING BELL
0315- 60             1180         RTS
```

Here is a sample patch program, just show how it is done. Any time the patch discovers a "#" character, it will ring the Apple's bell. The sample Applesoft lines show what I mean. If you want to try out the patch, assemble it and then call Applesoft. Then get to the monitor and patch CHRGET like this:

```
]CALL -151
*BA:4C 00 03
*3D0G
```

Then enter some Applesoft lines with embedded "#" characters, and RUN.

```
]LIST

10  PRINT #1;#2;#3;#4;#5;#6;#7
20 A# = 3:B# = 4: PRINT A + B
```

If you think of some really practical ways to use patches like this, let me know about them.

Decision Systems

Decision Systems
P.O. Box 13006
Denton, TX 76203
817/382-6353

DIS-ASSEMBLER

DSA-DS dis-assembles Apple machine language programs into forms compatible with LISA, S-C ASSEMBLER (3.2 or 4.0), Apple's TOOL-KIT ASSEMBLER and others. DSA-DS dis-assembles instructions or data. Labels are generated for referenced locations within the machine language program.
$25, Disk, Applesoft (32K, ROM or Language card)

OTHER PRODUCTS

ISAM-DS is an integrated set of Applesoft routines that gives indexed file capabilities to your **BASIC** programs. Retrieve by key, partial key or sequentially. Space from deleted records is automatically reused. Capabilities and performance that match products costing twice as much.
$50 Disk, Applesoft.

PBASIC-DS is a sophisticated preprocessor for structured **BASIC**. Use advanced logic constructs such as **IF...ELSE...**, **CASE**, **SELECT**, and many more. Develop programs for Integer or Applesoft. Enjoy the power of structured logic at a fraction of the cost of **PASCAL**.
$35. Disk, Applesoft (48K, ROM or Language Card).

FORM-DS is a complete system for the definition of input and output froms. **FORM-DS** supplies the automatic checking of numeric input for acceptable range of values, automatic formatting of numeric output, and many more features.
$25 Disk, Applesoft (32K, ROM or Language Card).

UTIL-DS is a set of routines for use with Applesoft to format numeric output, selectively clear variables (Applesoft's **CLEAR** gets everything), improve error handling, and interface machine language with Applesoft programs. Includes a special load routine for placing machine language routines underneath Applesoft programs.
$25 Disk, Applesoft.

SPEED-DS is a routine to modify the statement linkage in an Applesoft program to speed its execution. Improvements of 5-20% are common. As a bonus, **SPEED-DS** includes machine language routines to speed string handling and reduce the need for garbage clean-up. Author: Lee Meador.
$15 Disk, Applesoft (32K, ROM or Language Card).

(Add $4.00 for Foreign Mail)

***Apple II is a registered trademark of the Apple Computer Co.**

Leaving the S-C Assembler II

How do you get out of the assembler? I suppose I could have made a QUIT or EXIT command, but I didn't. If you want to go to Applesoft or Integer BASIC, type FP or INT. You will then be instantly in the version of Basic you wanted. However, you will still be hooked into the Assembler's output subroutine. If you load a small program and LIST it, you will find that tapping the space bar will stop the listing and restart it, just as inside the assembler. Notice I said a "small" program; a large program might over-write part of the assembler, causing the computer to hang up.

What you must do is type FP or INT, and then PR#0. The PR#0 unhooks the assembler output routine, and you are free.

Now, if you are sure that you have not over-written the assembler with your Applesoft or Integer BASIC program, and you want to return to the assembler, you can do so by typing CALL 4096. I use this for going back and forth rapidly when I am testing &-routines and the like.

What if you want to leave the assembler to go to the monitor? First of all, remember that you can use all of the monitor commands without ever leaving the assembler, by typing a dollar sign and then the monitor command. But if you really want out, how do you get there? If you have an old monitor ROM (not AUTOSTART), hitting RESET will get you to the monitor. With the Autostart ROM, you can type $FF59G or $FF69G. The first will unhook DOS, while the second will leave DOS hooked in. (The second is the same as the Basic command CALL-151.) Still another way is to patch the Autostart ROM RESET vector at $3F2 (type "$3F2:69 FF 5A"), so that RESET enters the monitor.

And how do you get back to the assembler from the monitor, without disturbing or losing your source code? Simply type "1003G" and you will be there. If you type "1000G" you will also get to the assembler, but all your source code will be gone, just as though you had typed the "NEW" command.

A New, Fancier .AS Directive

Many times I write text printing loops that depend on the sign bit of each byte to indicate the end of text. I might set up the text this way:

```
.AS /THIS IS THE TEXT I WANT TO PRIN
.AS -/T/
```

This assembles with the sign bits off (0) on all the characters of the text except the last one. I can terminate my printing loop by testing that bit. A little later, I will show you an example of just such a loop.

But when there are many messages, I get tired of using separate lines for the last character of each message! Why not have an assembler directive which automatically sets the sign bit of the last character the opposite of the sign bits of the rest of the message? Since Version 4.0 of the S-C Assembler II has a .US directive for me, the user, to program....

The only problem is that how to program for the .US directive has never been revealed. Until now.

The following little program will implement just the directive I want, and install it as the .US directive. It uses five programs inside the assembler (see lines 1100-1140). The code is patterned directly after the code for the .AS directive, which starts at $203C in most copies of Version 4.0.

> NOTE: You should check your assembler to make sure that the four bytes starting at $203C are "A0 00 84 04"; if they are, you can use the same addresses for the five routines as I have shown here. (If not, send me your original Version 4.0 disk for a free update. Be sure to adequately protect the disk for shipping, because your new copy will come back on the same disk.)

Line 1000 sets the origin of the code to $0F00. You could use some other origin, like $0300, if you wish. Just be sure it is an area of memory that you will not be using for some other purpose wile you are assembling. Line 1010 directs the object code to a BRUNnable file named B.US.DIRECTIVE.

The code from 1160 to 1210 is executed when you BRUN B.US.DIRECTIVE. It stores the address of DIR.US in the .US vector at the beginning of the assembler. You can read a little about this on page 15 of the Version 4.0 update manual.

Lines 1030-1050 define a few variables. WBUF is the line buffer the assembler uses, starting at $0200. The assembler unpacks a line from the source code into this buffer, and then proceeds to analyze it. DLIM and HIBIT are temporary locations in page zero where I will save the delimiter character and the high-bit setting.

The meat of the directive is in lines 1230-1510. If you disassemble the code at $203C in the S-C Assembler II, you will see a marked similarity here. You might also try disassembling the code for the GNNB and GNC subroutines.

GNC retrieves the next character from WBUF and increments the pointer. The character is tested. Carry status is set if the end-of-line token was picked up. Equal status is set if a blank or end-of-line token was picked up. GNNB calls on GNC until a non-blank character is found. GNC returns with the character in the A-register, and the pointer to the next character in the Y-register.

Lines 1240-1310 scan from the end of the opcode field to try to find the delimiter. If no non-blank character is found after the opcode field, you will get the "BAD ADDRESS ERROR". If a minus sign is found, $80 is stored in HIBIT instead of $00. This value will be merged with every character between the delimiters, to set or clear the high-bit of each byte. When the delimiter is found, it is stored in DLIM.

Lines 1320-1350 check to make sure that there are some characters after the delimiter before the next occurrence of the delimiter. For example, if you write ".US //", I want to assemble no bytes and go on. If I find the end-of-line token, you will get the error message.

Lines 1360-1430 are a loop to output the bytes one by one. I have to look ahead to see if the next character is the delimiter again. If not, then I will output the current character (by now accessed with "LDA WBUF-2,Y", because Y has been advanced). If the next one is the delimiter, then the current one is the last character of the string; I will have to go to ".3", to handle the last character.

Lines 1450-1490 handle the last character of the string between the delimiters. The high-bit is first set just like all the rest of the bytes at line 1460, and then reversed with the EOR #$80 at line 1470.

There is no end to the detail we could get into by describing how EMIT, CMNT, and ERBA work. I will leave them for you to puzzle over at your leisure. (Can't give away the whole plot in chapter 11)

```
                      1000          .OR $F00
                      1010          .TF B.US.DIRECTIVE
                      1020 *--------------------------------
0200-                 1030 WBUF     .EQ $0200
00DA-                 1040 DLIM     .EQ $DA
0004-                 1050 HIBIT    .EQ $04
                      1060 *--------------------------------
                      1070 *  THE FOLLOWING VALUES ARE FOR VERSION 4.0
                      1080 *  OF S-C ASSEMBLER II (DISK)
                      1090 *--------------------------------
1283-                 1100 GNNB     .EQ $1283     GET NEXT NON-BLANK CHAR
128B-                 1110 GNC      .EQ $128B     GET NEXT CHAR
188E-                 1120 CMNT     .EQ $188E     FINISH THE LINE
1932-                 1130 ERBA     .EQ $1932     ERROR: BAD ADDRESS
19FA-                 1140 EMIT     .EQ $19FA     EMIT A BYTE OF OBJECT CODE
                      1150 *--------------------------------
                      1160 ACTIVATE.US
0F00- A9 0B           1170          LDA #DIR.US   STORE ADDRESS IN .US VECTOR
0F02- 8D 0D 10        1180          STA $100D     INSIDE S-C ASSEMBLER II VER
0F05- A9 0F           1190          LDA /DIR.US   DISK VERSION 4.0
0F07- 8D 0E 10        1200          STA $100E
0F0A- 60              1210          RTS
                      1220 *--------------------------------
                      1230 DIR.US
0F0B- A0 00           1240          LDY #0        START WITH HI-BIT EQUAL TO ZERO
0F0D- 84 04           1250 .1       STY HIBIT     SET HI-BIT ZERO OR ONE
0F0F- 20 83 12        1260          JSR GNNB      GET NEXT NON-BLANK AFTER OPCODE
0F12- B0 32           1270          BCS ERBA2     END OF LINE IS BAD NEWS
0F14- A0 80           1280          LDY #$80      IN CASE WE NEED HI-BIT OF ONE
0F16- C9 2D           1290          CMP #$2D      CHECK FOR MINUS SIGN
0F18- F0 F3           1300          BEQ .1        YES, WE NEED HI-BIT OF ONE
0F1A- 85 DA           1310          STA DLIM      NOT MINUS, MUST BE DELIMITER
0F1C- 20 8B 12        1320          JSR GNC       GET NEXT CHARACTER
0F1F- B0 25           1330          BCS ERBA2     END OF LINE IS BAD NEWS
0F21- C5 DA           1340          CMP DLIM      SEE IF DELIMITER ALREADY
0F23- F0 1E           1350          BEQ .4        YES, NO STRING IN BETWEEN
0F25- 20 8B 12        1360 .2       JSR GNC       GET NEXT CHARACTER
0F28- B0 1C           1370          BCS ERBA2     END OF LINE IS BAD NEWS
0F2A- C5 DA           1380          CMP DLIM      SEE IF DELIMITER YET
0F2C- F0 0B           1390          BEQ .3        YES, FINISH UP AND RETURN
0F2E- B9 FE 01        1400          LDA WBUF-2,Y  NO, GET PREVIOUS CHAR
0F31- 05 04           1410          ORA HIBIT     MERGE WITH SELECTED HI-BIT
0F33- 20 FA 19        1420          JSR EMIT      EMIT THE OBJECT CODE BYTE
0F36- 4C 25 0F        1430          JMP .2        GO FOR ANOTHER ONE
                      1440 *--------------------------------
0F39- B9 FE 01        1450 .3       LDA WBUF-2,Y  GET PREVIOUS CHAR
0F3C- 05 04           1460          ORA HIBIT     MERGE WITH SELECTED HI-BIT
0F3E- 49 80           1470          EOR #$80      TOGGLE HI-BIT SINCE LAST CHAR
0F40- 20 FA 19        1480          JSR EMIT      EMIT THE OBJECT CODE BYTE
0F43- 4C 8E 18        1490 .4       JMP CMNT      FINISH PROCESSING THE LINE
                      1500 *--------------------------------
0F46- 4C 32 19        1510 ERBA2    JMP ERBA      BAD ADDRESS ERROR
```

The following program shows how I might use the new .US directive I have just built. It prints the line of text from line 1230 ten times on the screen. The .US directive assures that I can tell when I am at the end of the text string by looking at the sign bit. That is just what the BMI opcode at line 1110 is doing. Lines 1070, 1080, 1190, and 1200 are the looping code to make ten copies of the line. Lines 1090-1150 print the message except for the last character; lines 1170-1180 print that last character and a carriage return.

```
                        1000 *-------------------------------
                        1010 *      DEMONSTRATE USE OF .US DIRECTIVE
                        1020 *-------------------------------
FDED-                   1030 MON.COUT   .EQ $FDED
FD8E-                   1040 MON.CROUT  .EQ $FD8E
                        1050 *-------------------------------
                        1060 DEMO.US
0800- A9 0A             1070         LDA #10       DO 10 LINES
0802- 8D 20 08          1080         STA LINE.COUNT
0805- A0 00             1090 .3      LDY #0
0807- B9 20 08          1100 .1      LDA TEXT,Y    GET CHAR FROM TEXT STRING
080A- 30 08             1110         BMI .2
080C- 09 80             1120         ORA #$80      MAKE NORMAL VIDEO
080E- 20 ED FD          1130         JSR MON.COUT
0811- C8                1140         INY           NEXT CHARACTER
0812- D0 F3             1150         BNE .1        ...ALWAYS
                        1160 *-------------------------------
0814- 20 ED FD          1170 .2      JSR MON.COUT
0817- 20 8E FD          1180         JSR MON.CROUT
081A- CE 20 08          1190         DEC LINE.COUNT
081D- D0 E6             1200         BNE .3
081F- 60                1210         RTS
                        1220 *-------------------------------
                        1230 TEXT    .US /THIS IS MY MESSAGE/
0820-                   1240 LINE.COUNT .BS 1
                        1250 *-------------------------------
```

Commented Listing of DOS 3.3 RWTS

Last March I started out this series of DOS listings with the RWTS portion of DOS 3.2.1. Since then I have printed all of DOS 3.2.1 and DOS 3.3 from $B800 thru $BFFF, except for DOS 3.3 RWTS. Somehow it almost was overlooked, but here it is now.

There are minor differences between the two versions of RWTS, which you can find by comparing the listing from the March 1981 issue of AAL and this one. The differences start at line 1810. I suppose the changes are meant to be improvements, but most of them seem to make very little difference.

One critical major difference: DOS 3.2.1 and previous versions use sector numbers which are actually written in the headers. DOS 3.3 uses two different sets of sector numbers: physical and logical. The physical sector numbers are recorded in the sector header blocks; logical sector numbers are used in RWTS calls and File Manager calls. The translation is performed using the table at line 4280, which I have called the PHYSICAL.SECTOR.VECTOR. This table is accessed at line 3310: the logical sector number is in the Y-register, and indexes into the physical sector vector to pick up a physical sector number.

```
                1000 *--------------------------------
                1010 *      DOS 3.3 DISASSEMBLY   $BD00-BEAE
                1020 *      BOB SANDER-CEDERLOF       3-3-81
                1030 *--------------------------------
0478-           1040 CURRENT.TRACK      .EQ $478
0478-           1050 DRIVE.1.TRACK      .EQ $478 THRU 47F (INDEX BY SLOT)
04F8-           1060 DRIVE.2.TRACK      .EQ $4F8 THRU 4FF (INDEX BY SLOT)
04F8-           1070 SEARCH.COUNT       .EQ $4F8
0578-           1080 RETRY.COUNT        .EQ $578
05F8-           1090 SLOT               .EQ $5F8
06F8-           1100 SEEK.COUNT         .EQ $6F8
                1110 *--------------------------------
C080-           1120 PHASE.OFF          .EQ $C080
C081-           1130 PHASE.ON           .EQ $C081
C088-           1140 MOTOR.OFF          .EQ $C088
C089-           1150 MOTOR.ON           .EQ $C089
C08A-           1160 ENABLE.DRIVE.1     .EQ $C08A
C08B-           1170 ENABLE.DRIVE.2     .EQ $C08B
C08C-           1180 Q6L                .EQ $C08C
C08D-           1190 Q6H                .EQ $C08D
C08E-           1200 Q7L                .EQ $C08E
C08F-           1210 Q7H                .EQ $C08F
                1220 *--------------------------------
002D-           1230 SECTOR     .EQ $2D
002A-           1240 TRACK      .EQ $2A
002F-           1250 VOLUME     .EQ $2F
0035-           1260 DRIVE.NO   .EQ $35
003C-           1270 DCT.PNTR   .EQ $3C,3D
003E-           1280 BUF.PNTR   .EQ $3E,3F
0046-           1290 MOTOR.TIME .EQ $46,47
0048-           1300 IOB.PNTR   .EQ $48,49
                1310 *--------------------------------
B800-           1320 PRE.NYBBLE         .EQ $B800
B82A-           1330 WRITE.SECTOR       .EQ $B82A
B8DC-           1340 READ.SECTOR        .EQ $B8DC
B944-           1350 READ.ADDRESS       .EQ $B944
B8C2-           1360 POST.NYBBLE        .EQ $B8C2
B9A0-           1370 SEEK.TRACK.ABSOLUTE .EQ $B9A0
BA00-           1380 DELAY.LOOP         .EQ $BA00
                1390 *--------------------------------
0010-           1400 ERR.WRITE.PROTECT  .EQ $10
0020-           1410 ERR.WRONG.VOLUME   .EQ $20
0040-           1420 ERR.BAD.DRIVE      .EQ $40
```

```
                    1430 *-------------------------------
                    1440         .OR $BD00
                    1450         .TA $800
                    1460 *-------------------------------
BD00- 84 48         1470 RWTS    STY IOB.PNTR SAVE ADDRESS OF IOB
BD02- 85 49         1480         STA IOB.PNTR+1
BD04- A0 02         1490         LDY #2
BD06- 8C F8 06      1500         STY SEEK.COUNT  UP TO 2 RE-CALIBRATIONS
BD09- A0 04         1510         LDY #4
BD0B- 8C F8 04      1520         STY SEARCH.COUNT
BD0E- A0 01         1530         LDY #1       POINT AT SLOT# IN IOB
BD10- B1 48         1540         LDA (IOB.PNTR),Y  SLOT# FOR THIS OPERATION
BD12- AA            1550         TAX
BD13- A0 0F         1560         LDY #15      POINT AT PREVIOUS SLOT#
BD15- D1 48         1570         CMP (IOB.PNTR),Y  SAME SLOT?
BD17- F0 1B         1580         BEQ .3       YES
BD19- 8A            1590         TXA          SAVE NEW SLOT ON STACK
BD1A- 48            1600         PHA
BD1B- B1 48         1610         LDA (IOB.PNTR),Y  GET OLD SLOT#
BD1D- AA            1620         TAX
BD1E- 68            1630         PLA          STORE NEW SLOT #
BD1F- 48            1640         PHA          INTO OLD SLOT# SPOT
BD20- 91 48         1650         STA (IOB.PNTR),Y
                    1660 *-------------------------------
                    1670 *       SEE IF OLD MOTOR STILL SPINNING
                    1680 *-------------------------------
BD22- BD 8E C0      1690         LDA Q7L,X    GO INTO READ MODE
BD25- A0 08         1700 .1      LDY #8       IF DATA DOES NOT CHANGE
BD27- BD 8C C0      1710         LDA Q6L,X       FOR 96 MICROSECONDS,
BD2A- DD 8C C0      1720 .2      CMP Q6L,X       THEN THE DRIVE IS STOPPED
BD2D- D0 F6         1730         BNE .1       WOOPS! IT CHANGED!
BD2F- 88            1740         DEY          TIME UP YET?
BD30- D0 F8         1750         BNE .2       NO, KEEP CHECKING
BD32- 68            1760         PLA          GET NEW SLOT # AGAIN
BD33- AA            1770         TAX
                    1780 *-------------------------------
BD34- BD 8E C0      1790 .3      LDA Q7L,X    SET UP TO READ
BD37- BD 8C C0      1800         LDA Q6L,X
BD3A- A0 08         1810         LDY #8
BD3C- BD 8C C0      1820 .31     LDA Q6L,X    GET CURRENT DATA
BD3F- 48            1830         PHA          7 CYCLE DELAY
BD40- 68            1840         PLA
BD41- 48            1850         PHA          7 CYCLE DELAY
BD42- 68            1860         PLA
BD43- 8E F8 05      1870         STX SLOT
BD46- DD 8C C0      1880         CMP Q6L,X    SEE IF DATA CHANGED
BD49- D0 03         1890         BNE .32      YES, IT CHANGED
BD4B- 88            1900         DEY
BD4C- D0 EE         1910         BNE .31      KEEP WAITING
BD4E- 08            1920 .32     PHP          SAVE ANSWER ON STACK
BD4F- BD 89 C0      1930         LDA MOTOR.ON,X   TURN ON MOTOR
BD52- A0 06         1940         LDY #6       COPY POINTERS INTO PAGE ZERO
BD54- B1 48         1950 .4      LDA (IOB.PNTR),Y
BD56- 99 36 00      1960         STA DCT.PNTR-6,Y
BD59- C8            1970         INY          DCT.PNTR .EQ $3C,3D
BD5A- C0 0A         1980         CPY #10      BUF.PNTR .EQ $3E,3F
BD5C- D0 F6         1990         BNE .4
BD5E- A0 03         2000         LDY #3       GET MOTOR ON TIME FROM DCT
BD60- B1 3C         2010         LDA (DCT.PNTR),Y
BD62- 85 47         2020         STA MOTOR.TIME+1  HIGH BYTE ONLY
BD64- A0 02         2030         LDY #2       GET DRIVE #
BD66- B1 48         2040         LDA (IOB.PNTR),Y
BD68- A0 10         2050         LDY #16      SEE IF SAME AS OLD DRIVE#
BD6A- D1 48         2060         CMP (IOB.PNTR),Y
BD6C- F0 06         2070         BEQ .5       YES
BD6E- 91 48         2080         STA (IOB.PNTR),Y  UPDATE OLD DRIVE #
BD70- 28            2090         PLP          SET Z STATUS
BD71- A0 00         2100         LDY #0       TO FLAG MOTOR OFF
BD73- 08            2110         PHP
BD74- 6A            2120 .5      ROR          CHECK LSB OF DRIVE #
BD75- 90 05         2130         BCC .6       DRIVE 2
BD77- BD 8A C0      2140         LDA ENABLE.DRIVE.1,X
BD7A- B0 03         2150         BCS .7          ...ALWAYS
BD7C- BD 8B C0      2160 .6      LDA ENABLE.DRIVE.2,X
BD7F- 66 35         2170 .7      ROR DRIVE.NO SET SIGN BIT IF DRIVE 1
BD81- 28            2180         PLP          WAS MOTOR PROBABLY OFF?
BD82- 08            2190         PHP
BD83- D0 0B         2200         BNE .9       NO, DEFINITELY ON
```

```
                        2210 *-------------------------------
                        2220 *       DELAY FROM 150 TO 180 MILLISECONDS,
                        2230 *       DEPENDING ON WHAT GARBAGE IS IN A-REG
                        2240 *-------------------------------
BD85- A0 07             2250         LDY #7         YES, WAIT A WHILE
BD87- 20 00 BA          2260 .8      JSR DELAY.LOOP
BD8A- 88                2270         DEY            BUT IT WORKS ANYWAY....
BD8B- D0 FA             2280         BNE .8
BD8D- AE F8 05          2290         LDX SLOT       RESTORE SLOT#
                        2300 *-------------------------------
BD90- A0 04             2310 .9      LDY #4         GET TRACK #
BD92- B1 48             2320         LDA (IOB.PNTR),Y
BD94- 20 5A BE          2330         JSR SEEK.TRACK
BD97- 28                2340         PLP            WAS MOTOR DEFINITELY ON?
BD98- D0 11             2350         BNE PROCESS.COMMAND  YES, MOTOR ON
BD9A- A4 47             2360         LDY MOTOR.TIME+1   SEE IF NEED TO WAIT
BD9C- 10 0D             2370         BPL PROCESS.COMMAND   NO
                        2380 *-------------------------------
                        2390 *       MOTOR WAS OFF, SO WAIT REST OF MOTOR ON TIME
                        2400 *       FOR APPLE DISK II, MOTOR ON TIME IS 1 SECOND.
                        2410 *       PART OF THIS TIME IS COUNTED DOWN WHILE SEEKING
                        2420 *       FOR THE TRACK.
                        2430 *-------------------------------
BD9E- A0 12             2440 .10     LDY #18        ABOUT 100 MICROSECONDS PER TRIP
BDA0- 88                2450 .11     DEY
BDA1- D0 FD             2460         BNE .11
BDA3- E6 46             2470         INC MOTOR.TIME
BDA5- D0 F7             2480         BNE .10
BDA7- E6 47             2490         INC MOTOR.TIME+1
BDA9- D0 F3             2500         BNE .10
                        2510 *-------------------------------
                        2520 *       MOTOR ON AND UP TO SPEED, SO LET'S
                        2530 *       FIND OUT WHAT THE COMMAND IS AND DO IT!
                        2540 *-------------------------------
                        2550 PROCESS.COMMAND
BDAB- A0 0C             2560         LDY #12        GET COMMAND
BDAD- B1 48             2570         LDA (IOB.PNTR),Y
BDAF- F0 5A             2580         BEQ .8         NULL COMMAND, LET'S LEAVE
BDB1- C9 04             2590         CMP #4         FORMAT?
BDB3- F0 58             2600         BEQ .9         YES
BDB5- 6A                2610         ROR            SET CARRY=1 IF READ, =0 IF WRITE
BDB6- 08                2620         PHP            SAVE ON STACK
BDB7- B0 03             2630         BCS .1         READ
BDB9- 20 00 B8          2640         JSR PRE.NYBBLE  WRITE
BDBC- A0 30             2650 .1      LDY #48        UP TO 48 RETRIES
BDBE- 8C 78 05          2660         STY RETRY.COUNT
BDC1- AE F8 05          2670 .2      LDX SLOT       GET SLOT NUMBER AGAIN
BDC4- 20 44 B9          2680         JSR READ.ADDRESS
BDC7- 90 24             2690         BCC .5         GOOD ADDRESS READ
BDC9- CE 78 05          2700 .21     DEC RETRY.COUNT
BDCC- 10 F3             2710         BPL .2         KEEP TRYING
BDCE- AD 78 04          2720 .3      LDA CURRENT.TRACK  GET TRACK WE WANTED
BDD1- 48                2730         PHA            SAVE IT
BDD2- A9 60             2740         LDA #96        PRETEND TO BE ON TRACK 96
BDD4- 20 95 BE          2750         JSR SETUP.TRACK
BDD7- CE F8 06          2760         DEC SEEK.COUNT
BDDA- F0 28             2770         BEQ .6         NO MORE RE-CALIBRATES
BDDC- A9 04             2780         LDA #4
BDDE- 8D F8 04          2790         STA SEARCH.COUNT
BDE1- A9 00             2800         LDA #0         LOOK FOR TRACK 0
BDE3- 20 5A BE          2810         JSR SEEK.TRACK
BDE6- 68                2820         PLA            GET TRACK WE REALLY WANT
BDE7- 20 5A BE          2830 .4      JSR SEEK.TRACK
BDEA- 4C BC BD          2840         JMP .1
                        2850 *-------------------------------
BDED- A4 2E             2860 .5      LDY $2E        TRACK# IN ADDRESS HEADER
BDEF- CC 78 04          2870         CPY CURRENT.TRACK
BDF2- F0 1C             2880         BEQ .10        FOUND RIGHT TRACK
BDF4- AD 78 04          2890         LDA CURRENT.TRACK
BDF7- 48                2900         PHA            SAVE TRACK WE REALLY WANT
BDF8- 98                2910         TYA            SET UP TRACK WE ACTUALLY FOUNG
BDF9- 20 95 BE          2920         JSR SETUP.TRACK
BDFC- 68                2930         PLA            TRACK WE WANT
BDFD- CE F8 04          2940         DEC SEARCH.COUNT
BE00- D0 E5             2950         BNE .4         TRY AGAIN
BE02- F0 CA             2960         BEQ .3         TRY TO RE-CALIBRATE AGAIN
```

```
                    2970 *-------------------------------
                    2980 *       DRIVE ERROR, CANNOT FIND TRACK
                    2990 *-------------------------------
BE04- 68            3000 .6      PLA             REMOVE CURRENT.TRACK
BE05- A9 40         3010         LDA #ERR.BAD.DRIVE
BE07- 28            3020 .7      PLP
BE08- 4C 48 BE      3030         JMP ERROR.HANDLER
                    3040 *-------------------------------
                    3050 *       NULL COMMAND, ON THE WAY OUT....
                    3060 *-------------------------------
BE0B- F0 39         3070 .8      BEQ RWTS.EXIT
                    3080 *-------------------------------
                    3090 *       FORMAT COMMAND
                    3100 *-------------------------------
BE0D- 4C AF BE      3110 .9      JMP FORMAT
                    3120 *-------------------------------
                    3130 *       READ OR WRITE COMMAND
                    3140 *-------------------------------
BE10- A0 03         3150 .10     LDY #3          GET VOLUME# WANTED
BE12- B1 48         3160         LDA (IOB.PNTR),Y
BE14- 48            3170         PHA             SAVE DESIRED VOLUME# ON STACK
BE15- A5 2F         3180         LDA VOLUME
BE17- A0 0E         3190         LDY #14         STORE ACTUAL VOLUME NUMBER FOUND
BE19- 91 48         3200         STA (IOB.PNTR),Y
BE1B- 68            3210         PLA             GET DESIRED VOLUME# AGAIN
BE1C- F0 08         3220         BEQ .11         IF =0, DON'T CARE
BE1E- C5 2F         3230         CMP VOLUME      SEE IF RIGHT VOLUME
BE20- F0 04         3240         BEQ .11         YES
BE22- A9 20         3250         LDA #ERR.WRONG.VOLUME
BE24- D0 E1         3260         BNE .7          UH OH!
                    3270 *-------------------------------
BE26- A0 05         3280 .11     LDY #5          GET SECTOR# WANTED
BE28- B1 48         3290         LDA (IOB.PNTR),Y   (LOGICAL SECTOR NUMBER)
BE2A- A8            3300         TAY             INDEX INTO PHYSICAL SECTOR VECTOR
BE2B- B9 B8 BF      3310         LDA PHYSICAL.SECTOR.VECTOR,Y
BE2E- C5 2D         3320         CMP SECTOR
BE30- D0 97         3330         BNE .21         NOT THE RIGHT SECTOR
BE32- 28            3340         PLP             GET COMMAND FLAG AGAIN
BE33- 90 1C         3350         BCC WRITE
BE35- 20 DC B8      3360         JSR READ.SECTOR
BE38- 08            3370         PHP             SAVE RESULT; IF BAD, WILL BE COMMAND
BE39- B0 8E         3380         BCS .21         BAD READ
BE3B- 28            3390         PLP             THROW AWAY
BE3C- A2 00         3400         LDX #0
BE3E- 86 26         3410         STX $26
BE40- 20 C2 B8      3420         JSR POST.NYBBLE
BE43- AE F8 05      3430         LDX SLOT
                    3440 RWTS.EXIT
BE46- 18            3450         CLC
BE47- 24            3460         .HS 24          "BIT" TO SKIP NEXT INSTRUCTION
                    3470 *-------------------------------
                    3480 ERROR.HANDLER
BE48- 38            3490         SEC             INDICATE AN ERROR
BE49- A0 0D         3500         LDY #13         STORE ERROR CODE
BE4B- 91 48         3510         STA (IOB.PNTR),Y
BE4D- BD 88 C0      3520         LDA MOTOR.OFF,X
BE50- 60            3530         RTS
                    3540 *-------------------------------
BE51- 20 2A B8      3550 WRITE   JSR WRITE.SECTOR
BE54- 90 F0         3560         BCC RWTS.EXIT
BE56- A9 10         3570         LDA #ERR.WRITE.PROTECT
BE58- B0 EE         3580         BCS ERROR.HANDLER    ...ALWAYS
                    3590 *-------------------------------
                    3600 *       SEEK TRACK SUBROUTINE
                    3610 *       (A) = TRACK# TO SEEK
                    3620 *       (DRIVE.NO) IS NEGATIVE IF DRIVE 1
                    3630 *                  AND POSITIVE IF DRIVE 2
                    3640 *-------------------------------
                    3650 SEEK.TRACK
BE5A- 48            3660         PHA             SAVE TRACK#
BE5B- A0 01         3670         LDY #1          CHECK DEVICE CHARACTERISTICS TABLE
BE5D- B1 3C         3680         LDA (DCT.PNTR),Y   FOR TYPE OF DISK
BE5F- 6A            3690         ROR             SET CARRY IF TWO PHASES PER TRACK
BE60- 68            3700         PLA             GET TRACK# AGAIN
BE61- 90 08         3710         BCC .1          ONE PHASE PER TRACK
BE63- 0A            3720         ASL             TWO PHASES PER TRACK, SO DOUBLE IT
BE64- 20 6B BE      3730         JSR .1          FIND THE TRACK
BE67- 4E 78 04      3740         LSR CURRENT.TRACK  DIVIDE IT BACK DOWN
BE6A- 60            3750         RTS
```

```
                        3760 *-------------------------------
BE6B- 85 2A             3770 .1     STA TRACK
BE6D- 20 8E BE          3780        JSR GET.SLOT.IN.Y
BE70- B9 78 04          3790        LDA DRIVE.1.TRACK,Y
BE73- 24 35             3800        BIT DRIVE.NO   WHICH DRIVE?
BE75- 30 03             3810        BMI .2         DRIVE 1
BE77- B9 F8 04          3820        LDA DRIVE.2.TRACK,Y
BE7A- 8D 78 04          3830 .2     STA CURRENT.TRACK   WHERE WE ARE RIGHT NOW
BE7D- A5 2A             3840        LDA TRACK      WHERE WE WANT TO BE
BE7F- 24 35             3850        BIT DRIVE.NO WHICH DRIVE?
BE81- 30 05             3860        BMI .3         DRIVE 1
BE83- 99 F8 04          3870        STA DRIVE.2.TRACK,Y   DRIVE 2
BE86- 10 03             3880        BPL .4              ...ALWAYS
BE88- 99 78 04          3890 .3     STA DRIVE.1.TRACK,Y
BE8B- 4C A0 B9          3900 .4     JMP SEEK.TRACK.ABSOLUTE
                        3910 *-------------------------------
                        3920 *      CONVERT SLOT*16 TO SLOT IN Y-REG
                        3930 *-------------------------------
                        3940 GET.SLOT.IN.Y
BE8E- 8A                3950        TXA            SLOT*16 FROM X-REG
BE8F- 4A                3960        LSR
BE90- 4A                3970        LSR
BE91- 4A                3980        LSR
BE92- 4A                3990        LSR
BE93- A8                4000        TAY            SLOT INTO Y
BE94- 60                4010        RTS
                        4020 *-------------------------------
                        4030 *      SET UP CURRENT TRACK LOCATION
                        4040 *      IN DRIVE.1.TRACK OR DRIVE.2.TRACK VECTORS,
                        4050 *      INDEXED BY SLOT NUMBER.
                        4060 *
                        4070 *      (A) = TRACK# TO BE SET UP
                        4080 *-------------------------------
                        4090 SETUP.TRACK
BE95- 48                4100        PHA            SAVE TRACK # WE WANT TO SET UP
BE96- A0 02             4110        LDY #2         GET DRIVE NUMBER FROM IOB
BE98- B1 48             4120        LDA (IOB.PNTR),Y
BE9A- 6A                4130        ROR            SET CARRY IF DRIVE 1, CLEAR IF 2
BE9B- 66 35             4140        ROR DRIVE.NO MAKE NEGATIVE IF 1, POSITIVE IF 2
BE9D- 20 8E BE          4150        JSR GET.SLOT.IN.Y
BEA0- 68                4160        PLA            GET TRACK #
BEA1- 0A                4170        ASL            DOUBLE IT
BEA2- 24 35             4180        BIT DRIVE.NO   WHICH DRIVE?
BEA4- 30 05             4190        BMI .1         DRIVE 1
BEA6- 99 F8 04          4200        STA DRIVE.2.TRACK,Y
BEA9- 10 03             4210        BPL .2         ...ALWAYS
BEAB- 99 78 04          4220 .1     STA DRIVE.1.TRACK,Y
BEAE- 60                4230 .2     RTS
                        4240 *-------------------------------
                        4250 FORMAT
                        4260 *-------------------------------
BEAF-                   4270        .BS $BFB8-*
                        4280 PHYSICAL.SECTOR.VECTOR
BFB8- 00 0D 0B
BFBB- 09 07 05
BFBE- 03 01 0E
BFC1- 0C 0A 08
BFC4- 06 04 02
BFC7- 0F                4290        .HS 000D0B09070503010E0C0A080604020F
```

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $12 per year in the U.S.A., Canada, and Mexico. Other countries add $12/year for extra postage. Back issues are available for $1.20 each (other countries add $1 per back issue for postage).