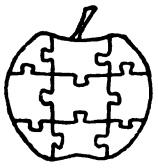

Apple Assembly Line

Volume 1 -- Issue 6 March, 1981

The Apple Assembly Line is still growing! I now am sending out over 300 copies per month! It is also growing in size, as you can see: this is the first 20 page issue.

In This Issue...

Second "Disk of the Quarter"

The second AALDQ is ready! If you would like to have the source code on disk in S-C Assembler II Version 4.0 format for all the programs which have appeared in AAL issues 4, 5, and 6, then send me $15. I will send you the disk, and you already have the documentation. DQ#1, covering issues 1, 2, and 3, is also still available at the same price.

Some New Books about the 6502

Apple Machine Language, by Don Inman and Kurt Inman, published by Reston (a Prentice-Hall Company). Hard cover, 296 pages, $14.95. If you are an absolute beginner, this is the book for you. You start by typing in an Applesoft program which helps you POKE in machine language code, and CALL it. Most of the examples involve lo-res graphics and sound. One chapter describes the Apple Mini-Assembler (which resides in the Integer BASIC ROMs). They never get around to a real assembler.

Practical Microcomputer Programming: the 6502, by W. J. Weller, published by Northern Technology Books. Hard cover, 459 pages, $32.95. Over 110 pages of the book are devoted to a listing of an assembler and a debugging package. A coupon inside the back cover can be redeemed for a tape copy which will run on the Apple II. By adding $7.50 to the coupon, you can get a disk version. The package can be loaded from the disk, but there is no capability for keeping source or object files on disk.

A Beautiful Dump . . . . . Robert H. Bernard

The old saying, "You can't tell the players without a scorecard," is certainly true for program debugging, and sometimes the only way is to look into memory and see what is there. The Apple II Monitor has a memory dump command, but I found it inadequate: it's formatted for a 40-column screen, it doesn't show ASCII codes, and getting output on a printer is a hassle.

So I sat down and wrote a quick assembly language memory dump modeled after a System/360 core dump (remember when computer memory was called "core"?), with both hex and ASCII. My first attempt took up more than one page of memory and was trapped where I assembled it by absolute internal references. I massaged it until it fit in less than a page and made it relocatable ("run anywhere") by making all internal jumps into relative branches. (A "page" in 6502 jargon is 256 bytes, with addresses running from xx00 through xxFF.)

Next I decided to add a printer feature; while I was at it I made it use 80 columns on the printer, 40 on the screen.

Next I made it print the bytes in groups of four, with a space between every four bytes. Sixteen bytes are printed per line on the screen, 32 on an 80-column printer. Spacing in groups of four makes it easier to spot certain address locations. If a byte value is a printable ASCII code, I print the character above the hexadecimal value. (Values $00-$1F and $80-$9F do not print.)

Then I wanted options to browze one screenful at a time, and backup when I passed the place I wanted to look at.

You probably think that by now the program is at least two, and maybe more, pages long. Not so! All the while I was able to keep it in only one page (which doesn't say much for my original code).

The end result (after 21 versions!) is listed here for your examination and pleasure.

Operating Instructions: BRUN the program anywhere in memory that you have a free page (256 bytes). When the "?" prompt appears, enter the address of the memory you want to dump in any of the following ways. After the address or address range, type the return key.

S.E	To dump memory from S to E on the screen.
S-E	To dump memory from S to E on the printer.
S,E	To dump memory from S to E on the screen, but pauses after each screenful; press space bar to continue, or press control-C to stop.
S	To dump from S, pausing after each line; press space bar to dump next line, press letter "B" to back up one line, or press control-C to stop.

```
                    1000 *-------------------------------
                    1010 *
                    1020 * APPLE II RELOCATABLE MEMORY DUMP PROGRAM
                    1030 *       BY ROBERT H. BERNARD
                    1040 *          35 DOGWOOD LANE
                    1050 *          WESTPORT, CT 06880
                    1060 *
                    1070 *          JANUARY 17, 1981
                    1080 *
                    1090 *       COMMERCIAL RIGHTS RESERVED
                    1100 *
                    1110 *-------------------------------
                    1120 *       MONITOR ROM ROUTINES
                    1130 *-------------------------------
FDED-               1140 MON.COUT   .EQ $FDED
FD0C-               1150 MON.RDKEY  .EQ $FD0C
FD67-               1160 MON.GTLNZ  .EQ $FD67
FFC7-               1170 MON.ZMODE  .EQ $FFC7
FFA7-               1180 MON.GETNUM .EQ $FFA7
FD8E-               1190 MON.CROUT  .EQ $FD8E
F940-               1200 MON.PRNTYX .EQ $F940
F94A-               1210 MON.PRBL2  .EQ $F94A
FDDA-               1220 MON.PRBYTE .EQ $FDDA
FF65-               1230 MON.MON    .EQ $FF65
FC58-               1240 MON.HOME   .EQ $FC58
FE18-               1250 MON.SETMOD .EQ $FE18
FE95-               1260 MON.OUTPOR .EQ $FE95     SET OUTPUT PORT TO SLOT (A)
FE93-               1270 MON.SETVID .EQ $FE93     SET VIDEO
                    1280 *-------------------------------
                    1290 *       I/O ADDRESSES
                    1300 *-------------------------------
C000-               1310 KBD    .EQ $C000     KEYBOARD
C010-               1320 KBSTRB .EQ $C010     KBD RESET STROBE
                    1330 *-------------------------------
                    1340 *       PAGE-ZERO VARIABLES
                    1350 *-------------------------------
002E-               1360 PGCNT  .EQ $2E       LINES LEFT THIS PAGE
0030-               1370 ITEMCT .EQ $30       ITEMS PER LINE
0031-               1380 OPTION .EQ $31       SAME AS MON "MODE"
0033-               1390 PROMPT .EQ $33       LOC OF GETLN PROMPT CHAR
0034-               1400 YSAV   .EQ $34       POINTER TO IN BUFFER
003C-               1410 FRADRL .EQ $3C       STARTING ADR LO ORDER
003D-               1420 FRADRH .EQ $3D       ..HI ORDER
003E-               1430 TOADRL .EQ $3E       ENDING ADR LO ORDER
003F-               1440 TOADRH .EQ $3F       ..HI ORDER
                    1450 *-------------------------------
                    1460 *       USER-CHANGEABLE PARAMETERS
                    1470 *-------------------------------
0010-               1480 SCITMS .EQ 16        BYTES PER LINE SCREEN
0020-               1490 PRITMS .EQ 32        BYTES PER LINE PRINTER
0008-               1500 ITMSPG .EQ 8         ITEMS PER PAGE
0001-               1510 PRSLOT .EQ 1         PRINTER SLOT
                    1520 *-------------------------------
                    1530         .OR $0800
                    1540 *-------------------------------
0800- 20 93 FE      1550 MEMDMP JSR MON.SETVID   SET PR#0
0803- A9 BF         1560        LDA #$BF         '?' FOR BOUNDS
0805- 85 33         1570        STA PROMPT       SET PROMPT CHAR
0807- 20 67 FD      1580        JSR MON.GTLNZ    CR, THEN GET INPUT
080A- 20 C7 FF      1590        JSR MON.ZMODE    SET HEX DECODE MODE
080D- 20 A7 FF      1600        JSR MON.GETNUM
0810- 84 34         1610        STY YSAV         REMEMBER SCAN POS.
0812- E0 00         1620        CPX #0           ANY ADR SCANNED?
0814- D0 03         1630        BNE .3           YES
0816- 60            1640        RTS              NO. TERMINATE
0817- 65 FF         1650        .DA MON.MON      MONITOR ENTRY (IN CASE YOU WANT
                    1660 *                       TO CHANGE RETURN TO "JMP MON.MON")
                    1670 *
0819- A9 F0         1680 .3     LDA #-SCITMS  BYTES PER SCREEN LINE
081B- 85 30         1690        STA ITEMCT    ITEMS PER LINE
081D- 20 18 FE      1700        JSR MON.SETMOD   SET TO SCAN 2ND ARG
0820- C9 AD         1710        CMP #$AD         IS OPTION = '-' ?
0822- D0 0D         1720        BNE .2           NO. CHECK OTHERS
0824- E6 31         1730        INC OPTION       MAKE '.'
0826- A9 01         1740        LDA #PRSLOT   PRINTER SLOT NO
0828- 20 95 FE      1750        JSR MON.OUTPOR   SET OUTPUT PORT
082B- A9 E0         1760        LDA #-PRITMS  BYTES PER PRINTER LINE
082D- 85 30         1770        STA ITEMCT    ITEMS PER LINE
082F- D0 08         1780        BNE .1           GO GET 2ND ARG
```

```
0831- C9 AE      1790 .2     CMP #$AE      '.' ?
0833- F0 04      1800        BEQ .1        YES. 2 ARGS
0835- C9 AC      1810        CMP #$AC      ','?
0837- D0 07      1820        BNE SETPGL    ONLY ONE ARG
0839- A4 34      1830 .1     LDY YSAV      PTR TO IN BUFFER
083B- 20 A7 FF   1840        JSR MON.GETNUM   SCAN 2ND ARG
083E- 84 34      1850        STY YSAV      PTR TO IN BUFFER
0840- A9 08      1860 SETPGL LDA #ITMSPG   ITEMS PER PAGE
0842- 85 2E      1870        STA PGCNT
                 1880 *
0844- 20 8E FD   1890 NEXTLN JSR MON.CROUT    SKIP A LINE
0847- A5 30      1900        LDA ITEMCT    -ITEMS PER LINE
0849- 25 3C      1910        AND FRADRL    STARTING ADR 0 MOD ITEMCT
084B- 85 3C      1920        STA FRADRL
084D- AA         1930        TAX
084E- A4 3D      1940        LDY FRADRH    ..TO PRINT
0850- 20 40 F9   1950        JSR MON.PRNTYX   PRINT IT IN HEX
0853- A6 30      1960        LDX ITEMCT    NO OF BYTES THIS LINE
0855- A0 00      1970        LDY #0        POINTER
0857- F0 15      1980        BEQ NOBLNK    DON'T SPACE FIRST TIME
                 1990 *
0859- AD 00 C0   2000 CHKKEY LDA KBD       KEY DOWN?
085C- 10 40      2010        BPL CKDONE    NO
085E- AD 10 C0   2020        LDA KBSTRB    YES. CLEAR KEYBOARD
0861- 38         2030        SEC           PREPARE FOR
0862- B0 9C      2040 MDMP2  BCS MEMDMP    JMP TO START
                 2050 *
0864- 98         2060 NXTCHR TYA           TEST FOR
0865- 29 03      2070        AND #$03      0 MOD 4
0867- D0 05      2080        BNE NOBLNK
0869- A9 A0      2090        LDA #$A0
086B- 20 ED FD   2100        JSR MON.COUT PRINT A BLANK
086E- A9 A0      2110 NOBLNK LDA #$A0
0870- 20 ED FD   2120        JSR MON.COUT PRINT A BLANK
0873- B1 3C      2130        LDA (FRADRL),Y
0875- C9 20      2140        CMP #$20      CNTRL CHAR?
0877- 90 08      2150        BCC .1        YES. SUBSTITUTE BLANK
0879- C9 80      2160        CMP #$80      CNTRL CHAR?
087B- 90 06      2170        BCC .2        NO. OK TO PRINT
087D- C9 A0      2180        CMP #$A0      CNTRL CHAR?
087F- B0 02      2190        BCS .2        NO. OK TO PRINT
0881- A9 A0      2200 .1     LDA #$A0      SUBSTITUTE BLANK
0883- 20 ED FD   2210 .2     JSR MON.COUT
0886- C8         2220        INY           POINT AT NEXT
0887- E8         2230        INX           DONE ON THIS LINE?
0888- D0 DA      2240        BNE NXTCHR    NO
088A- 20 8E FD   2250        JSR MON.CROUT    YES. CR
                 2260 * PREPARE TO PRINT SAME ITEMS IN HEX
088D- A2 03      2270        LDX #3
088F- 20 4A F9   2280        JSR MON.PRBL2    OUTPUT (X) BLANKS
0892- A6 30      2290        LDX ITEMCT    ITEMS PER LINE
0894- A0 00      2300        LDY #0        POINTER
0896- F0 12      2310        BEQ NXTHEX    (JMP)
                 2320 *
0898- B0 A6      2330 SETPL1 BCS SETPGL    JUMP TO SET PG LENGTH
089A- C9 AC      2340 CKOPT  CMP #$AC      NO. OPTION=',' ?
089C- D0 A6      2350 NXTLN1 BNE NEXTLN    NO. JUMP TO PRINT
089E- A5 3C      2360 CKDONE LDA FRADRL    TEST IF DONE
08A0- C5 3E      2370        CMP TOADRL
08A2- A5 3D      2380        LDA FRADRH
08A4- E5 3F      2390        SBC TOADRH
08A6- 90 9C      2400        BCC NEXTLN    FROM < TO
08A8- B0 B8      2410 MDMP1  BCS MDMP2     JMP TO START
                 2420 *
08AA- 98         2430 NXTHEX TYA           TEST FOR
08AB- 29 03      2440        AND #$03      0 MOD 4
08AD- D0 05      2450        BNE .1        IF NOT, SKIP BLANK
08AF- A9 A0      2460        LDA #$A0
08B1- 20 ED FD   2470        JSR MON.COUT PRINT A BLANK
08B4- B1 3C      2480 .1     LDA (FRADRL),Y BYTE TO OUTPUT
08B6- 20 DA FD   2490        JSR MON.PRBYTE   OUTPUT IN HEX
08B9- C8         2500        INY           NEXT
08BA- E8         2510        INX           DONE ON THIS LINE?
08BB- D0 ED      2520        BNE NXTHEX    NO
08BD- 20 8E FD   2530        JSR MON.CROUT    YES. CR
```

```
                      2540 *       ADVANCE DUMP ADDRESS
08C0- 38              2550         SEC            PREPARE FOR SUBTRACT
08C1- A5 3C           2560         LDA FRADRL     INCREMENT ADDRESS
08C3- E5 30           2570         SBC ITEMCT     -ITEMS PER LINE
08C5- 85 3C           2580         STA FRADRL
08C7- 90 04           2590         BCC .2         NO CARRY
08C9- E6 3D           2600         INC FRADRH     PAGE BOUNDARY
08CB- F0 DB           2610         BEQ MDMP1      END OF MEMORY
08CD- A5 31           2620 .2      LDA OPTION
08CF- C9 AE           2630         CMP #$AE       '.'? (OPTION 1)
08D1- F0 86           2640         BEQ CHKKEY     NO. CHECK IF KEY DOWN
08D3- C6 2E           2650 CHKPAG  DEC PGCNT      PAGE END?
08D5- D0 C3           2660         BNE CKOPT      NO. CHECK OPTION
08D7- 20 0C FD        2670 PAUSE   JSR MON.RDKEY     GET A CHAR
08DA- C9 83           2680         CMP #$83       CNTRL-C?
08DC- F0 CA           2690         BEQ MDMP1      YES. START OVER
08DE- C9 C2           2700         CMP #$C2       WAS CHAR READ A 'B'?
08E0- F0 0A           2710         BEQ BACKUP     YES
08E2- A5 31           2720         LDA OPTION
08E4- C9 AC           2730         CMP #$AC       OPTION=',' ?
08E6- F0 B0           2740         BEQ SETPL1     YES
08E8- E6 2E           2750 ADVNCE  INC PGCNT      ONE MORE TIME
08EA- D0 B0           2760         BNE NXTLN1     JMP TO NXTLN
                      2770 *
08EC- A5 3C           2780 BACKUP  LDA FRADRL     CARRY IS SET
08EE- E9 90           2790         SBC #144       BACKUP SCITMS*(ITMSPG+1) BYTES
08F0- 85 3C           2800         STA FRADRL     SAVE LO ORDER
08F2- B0 02           2810         BCS .1         NO CARRY
08F4- C6 3D           2820         DEC FRADRH     PROPOGATE CARRY
08F6- 20 58 FC        2830 .1      JSR MON.HOME      CLEAR SCREEN
08F9- 38              2840         SEC            SIMULATE JMP
08FA- B0 9C           2850         BCS SETPL1     ..TO SETPGL
                      2860 *
00FC-                 2870 ZZSIZE  .EQ *-MEMDMP   PROGRAM SIZE
```

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)

So-Called Unused Opcodes

The 6502 has 104 so-called unused opcodes. The various charts and reference manuals I have checked either leave them blank or call them "unused", "no-operation", or "future expansion". The 6502 has been around since 1976; I think we have waited long enough to know there will be no "expansion". But are they really unused? Do they have any effect if we try to execute them? Are they really no-ops? If so, how many bytes does the processor assume for each one?

These questions had never bothered me until I was looking through some disassembled memory and thought I found evidence of someone USING the "unused". It turned out they were not, but my curiosity was aroused. Just for fun, I built a little test routine and tried out the $FF opcode. Lo and behold! The 6502 thinks it is a 3-byte instruction, and it changes the A-register and some status bits!

About 45 minutes later I pinned it down: FFxxyy performs exactly the same as the two instructions FExxyy and FDxxyy. It is just as though I had executed one and then the other. In other words, anywhere in a program I find:

```
     INC VARIABLE,X
     SBC VARIABLE,X
```

I can substitute:

```
     .HS FF
     .DA VARIABLE
```

!!!!

You might wonder if I will ever find that sequence. I did try writing a program to demonstrate its use. It has the advantage of saving 3 bytes, and 4 clock cycles. (The SBC instruction is executed DURING the 7 cycles of the INC instruction!)

```
     TEST LDX INDEX
          LDA #10        FOR COUNTER(X)=10 TO 39
          STA COUNTER,X
     .1   LDA COUNTER,X  GET COUNTER(X)
          JSR $FDDA      PRINT IT OUT (OR WHATEVER)
          LDA #39        LIMIT
          .HS FF         DO INC AND SBC
          .DA COUNTER    ON COUNTER,X
          BCS .1         NEXT
          RTS
```

Are there any more? Before I could rest my curiosity, I had spent at least ten more hours, and had figured out what all 104 "unused opcodes" really do!

The center-fold chart shows the fruit of my detective work. The shaded opcodes are the "unused" ones. I don't know if every 6502 behaves the same as mine or not. Mine appears to be made by Synertek, and has a date code of 7720 (20th week of 1977). It could be that later versions or chips from other sources (MOS Technology or Rockwell) are different. If you find yours to be different, please let me know!

Twelve of the opcodes, all in column "x2", hang up the 6502; the only way to get out is to hit RESET or turn off the machine.

There are 27 opcodes which appear to have no effect on any registers or on memory. These could be called "NOP", but some of them are considered by the 6502 to have 2 or 3 bytes. I have labeled them "nop", "nop2", and "nop3" to distinguish how many bytes the 6502 thinks it is using. You could call nop2 "always skip one byte" and nop3 "always skip two bytes".

The action most of the rest perform can be deduced by looking at the other opcodes in the same row. For example, all of the xF column (except 8F and 9F) perform two instructions together: first the corresponding xE opcode, and then the corresponding xD opcode. In the same way, most of the opcodes in column x7 combine the x6 and x5 opcodes. The x3 column mirrors the x7 and xF columns, but with different addressing modes. And finally, the xB column mimics the other three columns, but with more exceptions. Most of the exceptions are in the 8x and 9x rows.

A few of the opcodes seem especially interesting and potentially useful. For example, A3xx performs three steps: first it loads xx into the X-register; then using this new value of X, it moves the byte addressed by (xx,X) into both the A- and X- registers. Another way of looking at this one is to say that whatever value xx has is doubled; then the two pagezero bytes at 2*xx and 2*xx+1 are used as the address for loading the A- and X-registers. You could use this for something, couldn't you?

There are five instructions which form the logical product of the A- and X-registers (without disturbing either register) and store the result in memory. If we call this new instruction "SAX", for "Store A&X", we have:

```
83   SAX (z,X)          8F   SAX a
87   SAX z              9F   SAX a,X
97   SAX z,Y
```

We get seven forms of the combination which shift a memory location using ASL, and then inclusive OR the results into A with an ORA instruction. If we call this new instruction ALO, we have:

```
03   ALO (z,X)          1B   ALO a,Y
13   ALO (z),Y          0F   ALO a
07   ALO z              1F   ALO a,X
17   ALO z,X
```

The same seven forms occur for the combinations ROL-AND, LSR-EOR, and ROR-ADC. Note that if you don't care what happens to the A-register, and the status register, these 28 instructions make two extra addressing modes available to the shift instructions: (z,X) and (z),Y.

Opcodes 4B and 6B might also be useful. You can do an AND-immediate followed by LSR or ROR on the A-register.

Opcodes 93, 9B, and 9E are really weird! It took a lot of head-scratching to figure out what they do.

93 Forms the logical product of the A-register and the byte at z+1 (which I call "hea") and stores it at (z),Y.

9B Forms the logical product of the A- and X-registers, and stores the result in the S-register (stack pointer)! Ouch! Then it takes up the third byte of the instruction (yy from 9B xx yy) and adds one to it (I call it "hea+1"). Then it forms the logical product of the new S-register and "hea+1" and stores the result at "a,Y". Whew!

9E Forms the logical product of the X-register and "hea+1" and stores the result at "a,Y".

We get six forms of the new "LAX" instruction, which loads the same value into both the A- and X-registers:

B3	LAX (z),Y	AB	LAX #v
A7	LAX z	AF	LAX a
B7	LAX z,Y	BF	LAX a,Y

I skipped over BB, because it is another extremely weird one. It forms the logical product of the byte at "a,Y" and S-register, and stores the result in the A-, X-, and S-registers. No wonder they didn't tell us about it!

Right under that one is the CB instruction. Well, good buddy (please excuse the CB talk!), it forms the logical product of the A- and X-registers, subtracts the immediate value (second byte of CB xx), and puts the result into the X-register.

The Cx and Dx rows provide us with seven forms that do a DEC on a memory byte, and then CMP the result with the A-register. Likewise, the Ex and Fx rows give us seven forms that perform INC followed by SBC.

It is a good thing to be aware that the so-called "unused" opcodes can be quite dangerous if they are accidentally executed. If your program goes momentarily wild and executes some data, chances are something somewhere will get strangely clobbered.

Since all of the above information was deduced by testing and observation, I cannot be certain that I am 100% correct. I may have overlooked or mis-interpreted some results, or even made a clerical error. Furthermore, as I said before, my 6502 may be different from yours. You can test your own, to see if it works like mine.

And if the whole exercise seems academic to you, you can at least enjoy the first legible and complete hexadecimal opcode chart for the 6502.

	x0	x1	x2	x3	x4	x5	x6	x7
0x	BRK	ORA (z,X)	hang	ASL (z,X) ORA (z,X)	nop2	ORA z	ASL z	ASL z ORA z
1x	BPL r	ORA (z),Y	hang	ASL (z),Y ORA (z),Y	nop2	ORA z,X	ASL z,X	ASL z,X ORA z,X
2x	JSR a	AND (z,X)	hang	ROL (z,X) AND (z,X)	BIT z	AND z	ROL z	ROL z AND z
3x	BMI r	AND (z),Y	hang	ROL (z),Y AND (z),Y	nop2	AND z,X	ROL z,X	ROL z,X AND z,X
4x	RTI	EOR (z,X)	hang	LSR (z,X) EOR (z,X)	nop2	EOR z	LSR z	LSR z EOR z
5x	BVC r	EOR (z),Y	hang	LSR (z),Y EOR (z),Y	nop2	EOR z,X	LSR z,X	LSR z,X EOR z,X
6x	RTS	ADC (z,X)	hang	ROR (z,X) ADC (z,X)	nop2	ADC z	ROR z	ROR z ADC z
7x	BVS r	ADC (z),Y	hang	ROR (z),Y ADC (z),Y	nop2	ADC z,X	ROR z,X	ROR z,X ADC z,X
8x	nop2	STA (z,X)	nop2	A&X --> (z,X)	STY z	STA z	STX z	A&X --> z
9x	BCC r	STA (z),Y	hang	A&hea --> (z),Y	STY z,X	STA z,X	STX z,Y	A&X --> z,Y

Ax	LDY #v	LDA (z,X)	LDX #v	LDX #v
				LDA (z,X)
				LDX (z,X)
					LDY z	LDA z	LDX z	LDX z
								LDA z

Bx	BCS r	LDA (z),Y	hang	LDA (z),Y LDX (z),Y	LDY z,X	LDA z,X	LDX z,Y	LDX z,Y LDA z,Y
Cx	CPY #v	CMP (z,X)	nop2	DEC (z,X) CMP (z,X)	CPY z	CMP z	DEC z	DEC z CMP z
Dx	BNE r	CMP (z),Y	hang	DEC (z),Y CMP (z),Y	nop2	CMP z,X	DEC z,X	DEC z,X CMP z,X
Ex	CPX #v	SBC (z,X)	nop2	INC (z,X) SBC (z,X)	CPX z	SBC z	INC z	INC z SBC z
Fx	BEQ r	SBC (z),Y	hang	INC (z),Y SBC (z),Y	nop2	SBC z,X	INC z,X	INC z,X SBC z,X

A	A-register (Accumulator)
S	S-register (Stack Pointer)
X	X-register
Y	Y-register
a	2-byte absolute address
r	1-byte relative address
v	1-byte immediate value
z	1-byte pagezero address
-->	"result is stored in"

x8	x9	xA	xB	xC	xD	xE	xF
PHP	ORA #v	ASL	AND #v	nop3	ORA a	ASL a	ASL a ORA a
CLC	ORA a,Y	nop	ASL a,Y ORA a,Y	nop3	ORA a,X	ASL a,X	ASL a,X ORA a,X
PLP	AND #v	ROL	AND #v	BIT a	AND a	ROL a	ROL a AND a
SEC	AND a,Y	nop	ROL a,Y AND a,Y	hop3	AND a,X	ROL a,X	ROL a,X AND a,X
PHA	EOR #v	LSR	AND #v LSR	JMP a	EOR a	LSR a	LSR a EOR a
CLI	EOR a,Y	nop	LSR a,Y EOR a,Y	nop3	EOR a,X	LSR a,X	LSR a,X EOR a,X
PLA	ADC #v	ROR	AND #v ROR	JMP (a)	ADC a	ROR a	ROR a ADC a
SEI	ADC a,Y	nop	ROR a,Y ADC a,Y	nop3	ADC a,X	ROR a,X	ROR a,X ADC a,X
DEY	nop2	TXA	#v&X --> A	STY a	STA a	STX a	A&X --> a
TYA	STA a,Y	TXS	A&X-->S S&hea+1 --> a,Y	nop3	STA a,X	X&hea+1 --> a,Y	A&X --> a,X
TAY	LDA #v	TAX	LDA #v TAX	LDY a	LDA a	LDX a	LDX a LDA a
CLV	LDA a,Y	TSX	a,Y & S -->A&S	LDY a,X	LDA a,X	LDX a,Y	LDX a,Y LDA a,Y
INY	CMP #v	DEX	A&X-#v --> X	CPY a	CMP a	DEC a	DEC a CMP a
CLD	CMP a,Y	nop	DEC a,Y CMP a,Y	nop3	CMP a,X	DEC a,X	DEC a,X CMP a,X
INX	SBC #v	NOP	SBC #v	CPX a	SBC a	INC a	INC a SBC a
SED	SBC a,Y	nop	INC a,Y SBC a,Y	nop3	SBC a,X	INC a,X	INC a,X SBC a,X

hea high-byte of effective address
     93: the byte at z+1
     9B: 3rd byte of instruction
     9E: 3rd byte of instruction

&   and-function (logical product)

hang computer hangs up, only way to
      regain control is to hit RESET

nop   1-byte instruction, no operation
nop2 2-byte instruction, no operation
nop3 3-byte instruction, no operation

Assembler Modifications:

1) Address $101C,which originally held a JSR $1F80, now does a JSR $24CC, where Bank #1 of the language card is turned on with a "dummy" LDY $C088, then the JSR $1F80 is performed and finally a RTS returns control to $101F where the initialization routine continues.

2) Address $1063 (modified from "B.EDIT2") does a JMP $24D3, where the subroutines "NEW.NML" & "MY.NML" from "B.EDIT2" now reside.

3) Address $1079 does a JMP now instead of a JSR (no change from "B.EDIT2").

4) Address $1125 is now a RTS (no change from "B.EDIT2"). I added the two NOPs at $1126 & $1127, because I don't like to leave "strange" code dangling.

5) Address $1246 now contains COPy instead of LOAd as a command and its associated address directs it to $D000, 1 byte before the COPY routine at $D001. The reason that COPY doesn't start at $D000 is obscure and warrants an explanation. If it were to start at $D000, the command table address would be $CFFF, the mere referencing of which is a signal to the APPLE II to turn OFF its peripheral cards! If an 80-column card were in use, for example, it could be turned off each time you tried to do a COPY!

6) Address $126E (incorrectly identified as $2746 in Lee Meador's article, AAL 1/81), now contains EDIt instead of SAVe. It points to $D13B, 1 byte before the EDITASM routine at $D13C. Thanks, Lee, for the terrific idea!

7) Addresses $24D3-$24E3 now contain the code which resided at $0824-$0834 in "B.EDIT2".

8) Addresses $24E4-$24EA now contain the code which resided at $0835-$083B in "B.EDIT2".

9) I changed Mike Laumer's <ctrl E> to <ctrl N>, to match Neil Konzen's PLE convention.

10) I also wanted an "instant recall" feature for the "last line edited". Typing EDIT without any argument works exactly as described by Mike, UNTIL a line has been edited. Then typing EDIT will cause that last edited line to be re-displayed. It doesn't "forget" that line # until a new line # is used as an EDIt argument.

SO...I have both COPY & EDIT capability, with all of the normal RAM space still available for assembled code! There's plenty of space left on the language card for future expansion (although there's only about $10 bytes unused in the $24EB area!). Well, aren't you excited? I sure was! - what are you waiting for? Get coding!!

B.EDIT2 Modifications:

1) Change the following lines:

```
1060            .OR $D13C
1070            .TF EDITASM A$D13C  Please use this name!
```

2) Delete lines 1360-1700

3) Add the following lines:

```
1360 NEW.NML    .EQ $24D3
1370 MY.NML     .EQ $24D9
1380 NEXT       .EQ $24E4,$24E5
1390 END        .EQ $24E6,$24E7
1400 CHAR       .EQ $24E8
1410 EDPTR      .EQ $24E9
1420 FKEY       .EQ $24EA
```

4) If you want EDIT to provide "instant recall", change line:

```
1740            BMI .4              YES, .4!
```

5) If you want <ctrl N> as "go to end of line" command, change line:

```
4330            .DA #$8E, E.END-1
```

6) SAVE EDITASM.SOURCE (or any name)

7) ASM the code

COPY Modifications:

1) Add the following lines:

```
1292            .OR $D001
1294            .TF BMC A$D001      Please use this name!
```

2) SAVE BLOCK MOVE/COPY.SOURCE (or any name)

3) ASM the code

Now...key in "Modifier EXEC Maker". If you have an APPLE II Plus, delete lines 40-50. SAVE it, then RUN it to create the textfile ASMDISK 4.0 MODIFIER. Now EXEC the file which will do all the hard work! I have my "Hello" program do the EXECuting! Good luck!

```
10 D$ =  CHR$ (4)
20  PRINT D$;"OPEN ASMDISK 4.0 MODIFIER"
30  PRINT D$;"WRITE ASMDISK 4.0 MODIFIER"
40  REM  GET INTO MAIN BOARD ROM - OTHERWISE ASSEMBLER BOMBS WHE
N CALLED
50  PRINT "INT"
60  REM TURN ON 'MON' TO SEE ACTION
70  PRINT "MON C,I,O"
80  REM BLOAD THE ASSEMBLER
90  PRINT "BLOAD ASMDISK 4.0"
100  REM GET INTO THE MONITOR
110  PRINT "CALL-151"
120  REM   UNLOCK BANK 1 OF LANG CARD
130  PRINT "C089"
140  PRINT "C089"
150  REM BLOAD BLOCK MOVE & COPY
160  PRINT "BLOAD BMC A$D001,A$D001"
170  REM BLOAD EDITASM
180  PRINT "BLOAD EDITASM A$D13C,A$D13C"
190  REM WRITE PROTECT LANG CARD
200  PRINT "C08A"
210  REM  ADD ROUTINE TO TURN ON BANK 1 OF LANGUAGE CARD EACH TI
ME $1010 IS CALLED
220  PRINT "101C:20 CC 24"
230  PRINT "24CC:AC 88 C0 20 80 1F 60"
240  REM MODIFY ASSEMBLER TO ADD 'NEW.NML'/'MY.NML' ROUTINES & L
OCAL VARIABLES FROM 'B.EDIT2' (NOT LOCATED THERE ANYMORE)
250  PRINT "24D3:20 D9 24 4C 26 10 A0 00 20 8D 12 20 4A 11"
260  PRINT "24E1:4C 66 10 00 00 00 00 00 00 00"
270  REM PATCH ASSEMBLER 'NML' SECTION
280  PRINT "1063:4C D3 24"
290  PRINT "1078:4C"
300  PRINT "1125:60 EA EA"
310  REM MODIFY ASSEMBLER COMMAND TABLE TO REPLACE 'LOAD' WITH C
OPY' & 'SAVE' WITH 'EDIT'
320  PRINT "1246:43 4F 50 00 D0"
330  PRINT "126E:45 44 49 3B D1"
340  REM MODIFY ASSEMBLER TO ADD '.DA WITH COMMA"
350  PRINT "20D4:4C B0 24"
360  PRINT "20D7:4C C7 24"
370  PRINT "20DA:4C B5 24"
380  PRINT "24B0:A5 DB 20 FA 19 A5 DC 20 FA 19 20 8B 12 C9 2C"
390  PRINT "24BF:F0 03 4C 8E 18 4C B5 20 A5 DB 18 90 EB"
400  REM PATCH 'PRT' VECTOR TO POINT TO SYMBOL TABLE AT $1E4E
410  PRINT "1009:4C 4E 1E"
420  REM  TURN OFF 'MON' NOW
430  PRINT "NOMON C,I,O"
440  REM JUMP TO ASSEMBLER
450  PRINT "1000G"
460  PRINT D$;"CLOSE ASMDISK 4.0 MODIFIER"
```

```
1  REM THIS PROGRAM MAKES A FILE
2  REM  'ASMDISK 4.0 MODIFIER'
3  REM WHICH IS THEN EXECUTABLE
4  REM     CREATED 1/10/81
5  REM           BY
6  REM     C. J. WELMAN
7  REM  SANTA ANA CA 92707
```

```
]LIST

10 D$ = "": REM     BLIND <CTRL D>
20  TEXT : CALL  - 936: PRINT "S-C ASSEMBLER II             VERSI
ON 4.0 ----------------------------------------"
30  PRINT "S-C SOFTWARE          COPYRIGHT  7-9-80"
40  VTAB 4
50  PRINT D$;"EXEC ASMDISK 4.0 MODIFIER"
60  END
```

Commented Listing of DOS 3.2.1 RWTS

I promised in the original AAL flyer that I would print dis-assemblies of things like DOS. Here is the first installment. RWTS is described in some detail in the DOS Reference Manual, pages 94-98.

There are not too many differences between the various versions of RWTS. Each one, from 3.1 to 3.2 to 3.2.1 to 3.3, seems mainly to clean up errors of the previous ones. I will probably print some DOS 3.3 listings in the future, as well as more of 3.2.1.

There is a bug in the 3.2.1 version (a bad address), at line 2200. It works anyway, but it is sloppy. Another problem I have discovered the hard way: the "previous slot #" in the IOB should be a slot that has a disk controller in it. If not, RWTS may do strange things to whatever is in that slot. I put in "0", and it turned on my language card! Zap! No more Applesoft!

```
                    1000 *       .LIST OFF
                    1010 *-------------------------------
                    1020 *       DOS 3.2.1 DISASSEMBLY $BD00-BE9F
                    1030 *       BOB SANDER-CEDERLOF        3-3-81
                    1040 *-------------------------------
0478-               1050 CURRENT.TRACK          .EQ $478
0478-               1060 DRIVE.1.TRACK          .EQ $478 THRU 47F (INDEX BY SLOT)
04F8-               1070 DRIVE.2.TRACK          .EQ $4F8 THRU 4FF (INDEX BY SLOT)
04F8-               1080 SEARCH.COUNT           .EQ $4F8
0578-               1090 RETRY.COUNT            .EQ $578
05F8-               1100 SLOT                   .EQ $5F8
06F8-               1110 SEEK.COUNT             .EQ $6F8
                    1120 *-------------------------------
C080-               1130 PHASE.OFF              .EQ $C080
C081-               1140 PHASE.ON               .EQ $C081
C088-               1150 MOTOR.OFF              .EQ $C088
C089-               1160 MOTOR.ON               .EQ $C089
C08A-               1170 ENABLE.DRIVE.1         .EQ $C08A
C08B-               1180 ENABLE.DRIVE.2         .EQ $C08B
C08C-               1190 Q6L                    .EQ $C08C
C08D-               1200 Q6H                    .EQ $C08D
C08E-               1210 Q7L                    .EQ $C08E
C08F-               1220 Q7H                    .EQ $C08F
                    1230 *-------------------------------
002D-               1240 SECTOR      .EQ $2D
002E-               1250 TRACK       .EQ $2E
002F-               1260 VOLUME      .EQ $2F
0035-               1270 DRIVE.NO    .EQ $35
003C-               1280 DCT.PNTR    .EQ $3C,3D
003E-               1290 BUF.PNTR    .EQ $3E,3F
0046-               1300 MOTOR.TIME  .EQ $46,47
0048-               1310 IOB.PNTR    .EQ $48,49
                    1320 *-------------------------------
B800-               1330 PRE.NYBBLE             .EQ $B800
B86A-               1340 WRITE.SECTOR           .EQ $B86A
B8FD-               1350 READ.SECTOR            .EQ $B8FD
B965-               1360 READ.ADDRESS           .EQ $B965
B9C1-               1370 POST.NYBBLE            .EQ $B9C1
BA1E-               1380 SEEK.TRACK.ABSOLUTE    .EQ $BA1E
                    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- BD 8C C0 1810          LDA Q6L,X     GET CURRENT DATA
BD3D- 48       1820          PHA           7 CYCLE DELAY
BD3E- 68       1830          PLA
BD3F- 8E F8 05 1840          STX SLOT
BD42- DD 8C C0 1850          CMP Q6L,X     SEE IF DATA CHANGED
BD45- 08       1860          PHP           SAVE ANSWER ON STACK
BD46- BD 89 C0 1870          LDA MOTOR.ON,X   TURN ON MOTOR
BD49- A0 06    1880          LDY #6        COPY POINTERS INTO PAGE ZERO
BD4B- B1 48    1890 .4       LDA (IOB.PNTR),Y
BD4D- 99 36 00 1900          STA DCT.PNTR-6,Y
BD50- C8       1910          INY           DCT.PNTR .EQ $3C,3D
BD51- C0 0A    1920          CPY #10       BUF.PNTR .EQ $3E,3F
BD53- D0 F6    1930          BNE .4
BD55- A0 03    1940          LDY #3        GET MOTOR ON TIME FROM DCT
BD57- B1 3C    1950          LDA (DCT.PNTR),Y
BD59- 85 47    1960          STA MOTOR.TIME+1  HIGH BYTE ONLY
BD5B- A0 02    1970          LDY #2        GET DRIVE #
BD5D- B1 48    1980          LDA (IOB.PNTR),Y
BD5F- A0 10    1990          LDY #16       SEE IF SAME AS OLD DRIVE#
BD61- D1 48    2000          CMP (IOB.PNTR),Y
BD63- F0 06    2010          BEQ .5        YES
BD65- 91 48    2020          STA (IOB.PNTR),Y  UPDATE OLD DRIVE #
BD67- 28       2030          PLP           SET Z STATUS
BD68- A0 00    2040          LDY #0        TO FLAG MOTOR OFF
BD6A- 08       2050          PHP
BD6B- 6A       2060 .5       ROR           CHECK LSB OF DRIVE #
BD6C- 90 05    2070          BCC .6        DRIVE 2
BD6E- BD 8A C0 2080          LDA ENABLE.DRIVE.1,X
BD71- B0 03    2090          BCS .7           ...ALWAYS
BD73- BD 8B C0 2100 .6       LDA ENABLE.DRIVE.2,X
BD76- 66 35    2110 .7       ROR DRIVE.NO  SET SIGN BIT IF DRIVE 1
BD78- 28       2120          PLP           WAS MOTOR PROBABLY OFF?
BD79- 08       2130          PHP
BD7A- D0 0B    2140          BNE .9        NO, DEFINITELY ON
               2150 *-------------------------------
               2160 *       DELAY FROM 150 TO 180 MILLISECONDS,
               2170 *       DEPENDING ON WHAT GARBAGE IS IN A-REG
               2180 *-------------------------------
BD7C- A0 07    2190          LDY #7        YES, WAIT A WHILE
BD7E- 20 7F BA 2200 .8       JSR $BA7F     ***BUG!!!*** SHOULD BE $BA7B
BD81- 88       2210          DEY           BUT IT WORKS ANYWAY....
BD82- D0 FA    2220          BNE .8
BD84- AE F8 05 2230          LDX SLOT      RESTORE SLOT#
               2240 *-------------------------------
BD87- A0 04    2250 .9       LDY #4        GET TRACK #
BD89- B1 48    2260          LDA (IOB.PNTR),Y
BD8B- 20 4B BE 2270          JSR SEEK.TRACK
BD8E- 28       2280          PLP           WAS MOTOR DEFINITELY ON?
BD8F- D0 0D    2290          BNE PROCESS.COMMAND  YES, MOTOR ON
```

```
                        2300 *-------------------------------
                        2310 *      MOTOR WAS OFF, SO WAIT REST OF MOTOR ON TIME
                        2320 *      FOR APPLE DISK II, MOTOR ON TIME IS 1 SECOND.
                        2330 *      PART OF THIS TIME IS COUNTED DOWN WHILE SEEKING
                        2340 *      FOR THE TRACK.
                        2350 *-------------------------------
BD91- A0 12             2360 .10    LDY #18       ABOUT 100 MICROSECONDS PER TRIP
BD93- 88                2370 .11    DEY
BD94- D0 FD             2380        BNE .11
BD96- E6 46             2390        INC MOTOR.TIME
BD98- D0 F7             2400        BNE .10
BD9A- E6 47             2410        INC MOTOR.TIME+1
BD9C- D0 F3             2420        BNE .10
                        2430 *-------------------------------
                        2440 *      MOTOR ON AND UP TO SPEED, SO LET'S
                        2450 *      FIND OUT WHAT THE COMMAND IS AND DO IT!
                        2460 *-------------------------------
                        2470 PROCESS.COMMAND
BD9E- A0 0C             2480        LDY #12       GET COMMAND
BDA0- B1 48             2490        LDA (IOB.PNTR),Y
BDA2- F0 5A             2500        BEQ .8        NULL COMMAND, LET'S LEAVE
BDA4- C9 04             2510        CMP #4        FORMAT?
BDA6- F0 58             2520        BEQ .9        YES
BDA8- 6A                2530        ROR           SET CARRY=1 IF READ, =0 IF WRITE
BDA9- 08                2540        PHP           SAVE ON STACK
BDAA- B0 03             2550        BCS .1        READ
BDAC- 20 00 B8          2560        JSR PRE.NYBBLE  WRITE
BDAF- A0 30             2570 .1     LDY #48       UP TO 48 RETRIES
BDB1- 8C 78 05          2580        STY RETRY.COUNT
BDB4- AE F8 05          2590 .2     LDX SLOT      GET SLOT NUMBER AGAIN
BDB7- 20 65 B9          2600        JSR READ.ADDRESS
BDBA- 90 24             2610        BCC .5        GOOD ADDRESS READ
BDBC- CE 78 05          2620 .21    DEC RETRY.COUNT
BDBF- 10 F3             2630        BPL .2        KEEP TRYING
BDC1- AD 78 04          2640 .3     LDA CURRENT.TRACK  GET TRACK WE WANTED
BDC4- 48                2650        PHA           SAVE IT
BDC5- A9 60             2660        LDA #96       PRETEND TO BE ON TRACK 96
BDC7- 20 86 BE          2670        JSR SETUP.TRACK
BDCA- CE F8 06          2680        DEC SEEK.COUNT
BDCD- F0 28             2690        BEQ .6        NO MORE RE-CALIBRATES
BDCF- A9 04             2700        LDA #4
BDD1- 8D F8 04          2710        STA SEARCH.COUNT
BDD4- A9 00             2720        LDA #0        LOOK FOR TRACK 0
BDD6- 20 4B BE          2730        JSR SEEK.TRACK
BDD9- 68                2740        PLA           GET TRACK WE REALLY WANT
BDDA- 20 4B BE          2750 .4     JSR SEEK.TRACK
BDDD- 4C AF BD          2760        JMP .1
                        2770 *-------------------------------
BDE0- A4 2E             2780 .5     LDY $2E       TRACK# IN ADDRESS HEADER
BDE2- CC 78 04          2790        CPY CURRENT.TRACK
BDE5- F0 22             2800        BEQ .10       FOUND RIGHT TRACK
BDE7- AD 78 04          2810        LDA CURRENT.TRACK
BDEA- 48                2820        PHA           SAVE TRACK WE REALLY WANT
BDEB- 98                2830        TYA           SET UP TRACK WE ACTUALLY FOUNG
BDEC- 20 86 BE          2840        JSR SETUP.TRACK
BDEF- 68                2850        PLA           TRACK WE WANT
BDF0- CE F8 04          2860        DEC SEARCH.COUNT
BDF3- D0 E5             2870        BNE .4        TRY AGAIN
BDF5- F0 CA             2880        BEQ .3        TRY TO RE-CALIBRATE AGAIN
                        2890 *-------------------------------
                        2900 *      DRIVE ERROR, CANNOT FIND TRACK
                        2910 *-------------------------------
BDF7- 68                2920 .6     PLA           REMOVE CURRENT.TRACK
BDF8- A9 40             2930        LDA #ERR.BAD.DRIVE
BDFA- 28                2940 .7     PLP
BDFB- 4C 39 BE          2950        JMP ERROR.HANDLER
                        2960 *-------------------------------
                        2970 *      NULL COMMAND, ON THE WAY OUT....
                        2980 *-------------------------------
BDFE- F0 37             2990 .8     BEQ RWTS.EXIT
                        3000 *-------------------------------
                        3010 *      FORMAT COMMAND
                        3020 *-------------------------------
BE00- A0 03             3030 .9     LDY #3        GET VOLUME# WANTED
BE02- B1 48             3040        LDA (IOB.PNTR),Y
BE04- 85 2F             3050        STA VOLUME    SET IN PLACE AND GO FORMAT
BE06- 4C A0 BE          3060        JMP FORMAT
```

```
                        3070 *-------------------------------
                        3080 *       READ OR WRITE COMMAND
                        3090 *-------------------------------
BE09- A0 03             3100 .10     LDY #3          GET VOLUME# WANTED
BE0B- B1 48             3110         LDA (IOB.PNTR),Y
BE0D- 48                3120         PHA             SAVE DESIRED VOLUME# ON STACK
BE0E- A5 2F             3130         LDA VOLUME
BE10- A0 0E             3140         LDY #14         STORE ACTUAL VOLUME NUMBER FOUND
BE12- 91 48             3150         STA (IOB.PNTR),Y
BE14- 68                3160         PLA             GET DESIRED VOLUME# AGAIN
BE15- F0 08             3170         BEQ .11         IF =0, DON'T CARE
BE17- C5 2F             3180         CMP VOLUME      SEE IF RIGHT VOLUME
BE19- F0 04             3190         BEQ .11         YES
BE1B- A9 20             3200         LDA #ERR.WRONG.VOLUME
BE1D- D0 DB             3210         BNE .7          UH OH!
                        3220 *-------------------------------
BE1F- A0 05             3230 .11     LDY #5          GET SECTOR# WANTED
BE21- A5 2D             3240         LDA SECTOR      AND THE ONE WE FOUND
BE23- D1 48             3250         CMP (IOB.PNTR),Y   AND COMPARE THEM.
BE25- D0 95             3260         BNE .21         NOT THE RIGHT SECTOR
BE27- 28                3270         PLP             GET COMMAND FLAG AGAIN
BE28- 90 18             3280         BCC WRITE
BE2A- 20 FD B8          3290         JSR READ.SECTOR
BE2D- 08                3300         PHP             SAVE RESULT; IF BAD, WILL BE COMMAND
BE2E- B0 8C             3310         BCS .21         BAD READ
BE30- 28                3320         PLP             THROW AWAY
BE31- 20 C1 B9          3330         JSR POST.NYBBLE
BE34- AE F8 05          3340         LDX SLOT
                        3350 RWTS.EXIT
BE37- 18                3360         CLC
BE38- 24                3370         .HS 24          "BIT" TO SKIP NEXT INSTRUCTION
                        3380 *-------------------------------
                        3390 ERROR.HANDLER
BE39- 38                3400         SEC             INDICATE AN ERROR
BE3A- A0 0D             3410         LDY #13         STORE ERROR CODE
BE3C- 91 48             3420         STA (IOB.PNTR),Y
BE3E- BD 88 C0          3430         LDA MOTOR.OFF,X
BE41- 60                3440         RTS
                        3450 *-------------------------------
BE42- 20 6A B8          3460 WRITE   JSR WRITE.SECTOR
BE45- 90 F0             3470         BCC RWTS.EXIT
BE47- A9 10             3480         LDA #ERR.WRITE.PROTECT
BE49- B0 EE             3490         BCS ERROR.HANDLER   ...ALWAYS
                        3500 *-------------------------------
                        3510 *       SEEK TRACK SUBROUTINE
                        3520 *       (A) = TRACK# TO SEEK
                        3530 *       (DRIVE.NO) IS NEGATIVE IF DRIVE 1
                        3540 *                  AND POSITIVE IF DRIVE 2
                        3550 *-------------------------------
                        3560 SEEK.TRACK
BE4B- 48                3570         PHA             SAVE TRACK#
BE4C- A0 01             3580         LDY #1          CHECK DEVICE CHARACTERISTICS TABLE
BE4E- B1 3C             3590         LDA (DCT.PNTR),Y   FOR TYPE OF DISK
BE50- 6A                3600         ROR             SET CARRY IF TWO PHASES PER TRACK
BE51- 68                3610         PLA             GET TRACK# AGAIN
BE52- 90 08             3620         BCC .1          ONE PHASE PER TRACK
BE54- 0A                3630         ASL             TWO PHASES PER TRACK, SO DOUBLE IT
BE55- 20 5C BE          3640         JSR .1          FIND THE TRACK
BE58- 4E 78 04          3650         LSR CURRENT.TRACK  DIVIDE IT BACK DOWN
BE5B- 60                3660         RTS
                        3670 *-------------------------------
BE5C- 85 2E             3680 .1      STA TRACK
BE5E- 20 7F BE          3690         JSR GET.SLOT.IN.Y
BE61- B9 78 04          3700         LDA DRIVE.1.TRACK,Y
BE64- 24 35             3710         BIT DRIVE.NO   WHICH DRIVE?
BE66- 30 03             3720         BMI .2          DRIVE 1
BE68- B9 F8 04          3730         LDA DRIVE.2.TRACK,Y
BE6B- 8D 78 04          3740 .2      STA CURRENT.TRACK   WHERE WE ARE RIGHT NOW
BE6E- A5 2E             3750         LDA TRACK       WHERE WE WANT TO BE
BE70- 24 35             3760         BIT DRIVE.NO WHICH DRIVE?
BE72- 30 05             3770         BMI .3          DRIVE 1
BE74- 99 F8 04          3780         STA DRIVE.2.TRACK,Y   DRIVE 2
BE77- 10 03             3790         BPL .4          ...ALWAYS
BE79- 99 78 04          3800 .3      STA DRIVE.1.TRACK,Y
BE7C- 4C 1E BA          3810 .4      JMP SEEK.TRACK.ABSOLUTE
```

```
                        3820 *-------------------------------
                        3830 *      CONVERT SLOT*16 TO SLOT IN Y-REG
                        3840 *-------------------------------
                        3850 GET.SLOT.IN.Y
BE7F- 8A                3860         TXA            SLOT*16 FROM X-REG
BE80- 4A                3870         LSR
BE81- 4A                3880         LSR
BE82- 4A                3890         LSR
BE83- 4A                3900         LSR
BE84- A8                3910         TAY            SLOT INTO Y
BE85- 60                3920         RTS
                        3930 *-------------------------------
                        3940 *      SET UP CURRENT TRACK LOCATION
                        3950 *      IN DRIVE.1.TRACK OR DRIVE.2.TRACK VECTORS,
                        3960 *      INDEXED BY SLOT NUMBER.
                        3970 *
                        3980 *      (A) = TRACK# TO BE SET UP
                        3990 *-------------------------------
                        4000 SETUP.TRACK
BE86- 48                4010         PHA            SAVE TRACK # WE WANT TO SET UP
BE87- A0 02             4020         LDY #2         GET DRIVE NUMBER FROM IOB
BE89- B1 48             4030         LDA (IOB.PNTR),Y
BE8B- 6A                4040         ROR            SET CARRY IF DRIVE 1, CLEAR IF 2
BE8C- 66 35             4050         ROR DRIVE.NO MAKE NEGATIVE IF 1, POSITIVE IF 2
BE8E- 20 7F BE          4060         JSR GET.SLOT.IN.Y
BE91- 68                4070         PLA            GET TRACK #
BE92- 0A                4080         ASL            DOUBLE IT
BE93- 24 35             4090         BIT DRIVE.NO   WHICH DRIVE?
BE95- 30 05             4100         BMI .1         DRIVE 1
BE97- 99 F8 04          4110         STA DRIVE.2.TRACK,Y
BE9A- 10 03             4120         BPL .2         ...ALWAYS
BE9C- 99 78 04          4130 .1      STA DRIVE.1.TRACK,Y
BE9F- 60                4140 .2      RTS
                        4150 *-------------------------------
                        4160 FORMAT
```

SYMBOL TABLE

```
003E- BUF.PNTR
0478- CURRENT.TRACK
003C- DCT.PNTR
0478- DRIVE.1.TRACK
04F8- DRIVE.2.TRACK
0035- DRIVE.NO
C08A- ENABLE.DRIVE.1
C08B- ENABLE.DRIVE.2
0040- ERR.BAD.DRIVE
0010- ERR.WRITE.PROTECT
0020- ERR.WRONG.VOLUME
BE39- ERROR.HANDLER
BEA0- FORMAT
BE7F- GET.SLOT.IN.Y
0048- IOB.PNTR
C088- MOTOR.OFF
C089- MOTOR.ON
0046- MOTOR.TIME
C080- PHASE.OFF
C081- PHASE.ON
B9C1- POST.NYBBLE
B800- PRE.NYBBLE
BD9E- PROCESS.COMMAND
.01=BDAF, .02=BDB4, .21=BDBC, .03=BDC1
.04=BDDA, .05=BDE0, .06=BDF7, .07=BDFA
.08=BDFE, .09=BE00, .10=BE09, .11=BE1F
C08D- Q6H
C08C- Q6L
C08F- Q7H
C08E- Q7L
B965- READ.ADDRESS
B8FD- READ.SECTOR
0578- RETRY.COUNT
BD00- RWTS
.01=BD25, .02=BD2A, .03=BD34, .04=BD4B
.05=BD6B, .06=BD73, .07=BD76, .08=BD7E
.09=BD87, .10=BD91, .11=BD93
BE37- RWTS.EXIT
04F8- SEARCH.COUNT
002D- SECTOR
06F8- SEEK.COUNT
BE4B- SEEK.TRACK
.01=BE5C, .02=BE6B, .03=BE79, .04=BE7C
BA1E- SEEK.TRACK.ABSOLUTE
BE86- SETUP.TRACK
.01=BE9C, .02=BE9F
05F8- SLOT
002E- TRACK
002F- VOLUME
BE42- WRITE
B86A- WRITE.SECTOR
```

& Command Interface for S-C Assembler II

Here is yet another way to add new commands to Version 4.0. You are somewhat familiar with the use of the & in Applesoft. This little program patches the assembler so that you can add as many new commands as you wish.

I have shown as examples the EDIT, COPY, and SYM commands. You need to fill in the correct starting address in lines 1250 and 1260.

Use the .TF directive to direct the object code to a file. Then use BRUN to install the patch. Lines 1100-1120 patch the assembler to hook in the code at lines 3010-3100. After it is hooked in, make a new copy of the assembler by using BSAVE ASMDISK 4.0 WITH &,A$FD7,L$.... (Fill in the appropriate length, depending on what else you have added to the assembler in the past.)

```
                    1000 *---------------------------------
                    1010 *       & COMMAND INTERFACE
                    1020 *
                    1030 *       &<COMMAND STRING>
                    1040 *
                    1050 *---------------------------------
                    1060 *
                    1070 *       ORIGIN MUST BE SET SO THAT LAST BYTE
                    1080 *       IS AT $0FFF.
                    1085         .OR $FD1
                    1090 *---------------------------------
0FD1- A9 AF         1100         LDA #AMPERSAND.INTERFACE-$103D
0FD3- 8D 3C 10      1110         STA $103C
0FD6- 60            1120         RTS
                    1130 *---------------------------------
0FD7- 4C 00 10      1140         JMP $1000
                    1150 *---------------------------------
0FDA- 05 03         1160 AOPTBL  .HS 0503
0FDC- 45 44 49      1170         .AS /EDI/
0FDF- 0F 10         1180         .DA EDIT-1
0FE1- 43 4F 50      1190         .AS /COP/
0FE4- 0F 10         1200         .DA COPY-1
0FE6- 53 59 4D      1210         .AS /SYM/
0FE9- 4D 1E         1220         .DA STPRNT-1
0FEB- 00            1230         .HS 00        END OF TABLE
                    1240 *---------------------------------
1010-               1250 EDIT    .EQ $1010
1010-               1260 COPY    .EQ $1010
1E4E-               1270 STPRNT  .EQ $1E4E
                    3000 *---------------------------------
                    3010 AMPERSAND.INTERFACE
0FEC- C9 26         3020         CMP #'&
0FEE- F0 03         3030         BEQ .1
0FF0- 4C 63 10      3040         JMP $1063
0FF3- A9 DA         3050 .1      LDA #AOPTBL
0FF5- 85 02         3060         STA $02
0FF7- A9 0F         3070         LDA /AOPTBL
0FF9- 85 03         3080         STA $03
0FFB- A9 01         3090         LDA #1
0FFD- 4C 47 10      3100         JMP $1047
```


Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. Subscription rate is $12/year, in the U.S.A., Canada, and Mexico. Other countries add $6/year for extra postage.