Apple Assembly Line

Volume 1 -- Issue 9 June, 1981

In This Issue...

New Quarterly Disk Ready

Remember that all the source programs which appear in the Apple Assembly Line are available on disk, ready to assembly with the S-C Assembler II Version 4.0. Every three months I collect it all on a Quarterly Disk, and you can get it for only $15.

QD#1 covers AAL issues 1-3 (October thru December 1980), QD#2 covers AAL issues 4-6 (January thru March 1981), QD#3 covers issues 7-9 (April thru June 1981). Copies of all back issues of the AAL newsletter are available for $1.20 each.

Another Way to Get 80-Columns

Those unpredictable Apple Parallel Interface ROMs! I wonder if even Apple knows how many different versions they have made, and why!

Anyway, as you know if you have one, some of them make it very difficult to get 80-column printout when you are using the S-C Assembler II. You should be able to type control-I and "80N", but the assembler sees control-I and does a tab. Plus you get a syntax error, and the printer is un-hooked.

You can type "$I80N" (where "I" means control-I). Or you can type "$579:50" (assuming slot 1).

Or, you can make the first line of your program do it. Type in this line so it will be the first line in your program:

```
0000 *I80N
```

Then type the "MEM" command. It will tell you the memory address where your source program starts. Using monitor commands, display about 8 bytes at the beginning of the source program. Look for the pattern "49 38 30 4E". Change the "49" to "09", which is ASCII for control-I. When your program is LISTed or ASMed, the control-I will be caught by Apple's interface and put you into 80-column mode.

So, now you have at least three ways to make it work. Don't you wish you had the ROM version which is in my Apple Parallel card? It works right without ANY of the above! Now if I could only make it work with my screen printing program....

Two Fancy Tone Generators.....Mark Kriegsman

I was not quite satisfied with the sound from Bob Sander-Cederlof's "Touch-Tone Simulator" (AAL February 1981, page 5.6). His method for making two simulataneous tones was to play one tone for a while and then the other one for a while, letting your ear put it all together. I have written the following DUAL.TONES program which mixes the two tones together in a more realistic way. I also wrote SINGLE.TONE which plays a given tone at 16 different volume levels. All out of the standard Apple speaker! Really!

The programs are accessed from Applesoft with the "&". (See lines 1510 and 1830.) SINGLE.TONE is called with &T followed by three expressions separated by commas. The three expressions are for the tone, duration, and volume, respectively. Tone is a value from 0 to 255, duration a value from 0 to 65535, and volume a value from 0 to 15. Experiment with different settings and you will see how it works. By making loops which change both pitch and volume, you can simulate the sound of a falling bomb or a passing car.

DUAL.TONES also needs three parameters: tone#1, duration, and tone#2, respectively. The two tone values must be between 0 and 255; duration is again a value from 0 to 65535. It is interesting to try two tone values very close together, to hear the beating effect, and two tones at harmonic intervals to hear the chords. I think &D 254,28000,255 sounds a little like a light saber. Again, a loop which varies both tone values can make some exciting sound effects!

Lines 1340-1400 are executed when you BRUN B.AMPERTONE; they set up the ampersand vector for Applesoft. Once this is done, an ampersand in your program or typed in as a direct command will start executing the AMPERTONE subroutine.

Lines 1440-1490 determine which & routine you are calling. The character following the "&" is in the A-register. If it is "T", SINGLE.TONE is called; if "D", DUAL.TONE is called; if neither, you get SYNTAX ERR.

Subroutines in the Applesoft ROMs are used to read the parameter expressions (lines 2190-2230). GTBYTC advances to the next character, and then evaluates the expression that starts there. If the value is between 0 and 255 it is returned in the X-register. (If not, you get RANGE ERR.) CHKCOM makes sure the next character is a comma; if it isn't, you get SYNTAX ERR. GETNUM is used in executing the POKE statement. It looks for an expression giving a value between 0 and 65535, then a comma, and then another expression giving a value between 0 and 255. The first value is stored at $50 and $51, and the second is returned in the X-register.

[Mark Kriegsman is a 15-year-old Apple expert living in Summit, New Jersey. I wrote the article above based on two letters and a program he sent. (Bob Sander-Cederlof)]

```
                        1000 *-------------------------------
                        1010 *       DUAL TONE, AND TONE WITH VOLUME CONTROL
                        1020 *-------------------------------
                        1030 *       WRITTEN BY MARK KRIEGSMAN.......5-22-81
                        1040 *       REVISED BY BOB SANDER-CEDERLOF..5-29-81
                        1050 *-------------------------------
                        1060         .OR $300
                        1070         .TF B.AMPERTONES
                        1080 *-------------------------------
                        1090 *       ROM SUBROUTINES USED
                        1100 *-------------------------------
DEBE-                   1110 CHKCOM .EQ $DEBE     MUST SEE COMMA
DEC9-                   1120 SYNERR .EQ $DEC9     SYNTAX ERROR
E6F5-                   1130 GTBYTC .EQ $E6F5     EAT CHAR, GET BYTE IN X
E746-                   1140 GETNUM .EQ $E746     GET TWO-BYTE VALUE IN $50,51
                        1150 *                    THEN COMMA AND ONE-BYTE VALUE IN X
                        1160 *-------------------------------
                        1170 *       PAGE-ZERO VARIABLES
                        1180 *-------------------------------
0050-                   1190 DURATION    .EQ $50 AND $51
00FB-                   1200 TONE1.CNT   .EQ $FB
00FC-                   1210 TONE2.CNT   .EQ $FC
00FD-                   1220 TONE1       .EQ $FD
00FE-                   1230 TONE2       .EQ $FE
00FF-                   1240 VOLUME      .EQ $FF
                        1250 *-------------------------------
                        1260 *       I/O ADDRESSES
                        1270 *-------------------------------
C030-                   1280 SPKR        .EQ $C030
                        1290 *-------------------------------
03F5-                   1300 AMPERSAND.VECTOR    .EQ $3F5 THRU $3F7
                        1310 *-------------------------------
                        1320 *       INITIALIZE AMPERSAND VECTOR
                        1330 *-------------------------------
0300- A9 4C             1340 INIT    LDA #$4C     JMP OPCODE
0302- 8D F5 03          1350         STA AMPERSAND.VECTOR
0305- A9 10             1360         LDA #AMPERTONE
0307- 8D F6 03          1370         STA AMPERSAND.VECTOR+1
030A- A9 03             1380         LDA /AMPERTONE
030C- 8D F7 03          1390         STA AMPERSAND.VECTOR+2
030F- 60                1400         RTS
                        1410 *-------------------------------
                        1420 *       AMPERSAND ENTRY POINT
                        1430 *-------------------------------
                        1440 AMPERTONE
0310- C9 54             1450         CMP #'T      IS IT TONE?
0312- F0 07             1460         BEQ SINGLE.TONE
0314- C9 44             1470         CMP #'D      IS IT DUAL?
0316- F0 37             1480         BEQ DUAL.TONES
0318- 4C C9 DE          1490         JMP SYNERR   NEITHER, SO SYNTAX ERROR
                        1500 *-------------------------------
                        1510 *       &T <TONE>,<DURATION>,<VOLUME>
                        1520 *-------------------------------
                        1530 SINGLE.TONE
031B- 20 84 03          1540         JSR GET.PARAMS
031E- 8A                1550         TXA          LIMIT VOLUME
031F- 29 0F             1560         AND #15      TO 0-15
0321- 85 FF             1570         STA VOLUME
0323- A5 FD             1580         LDA TONE1
0325- 85 FB             1590         STA TONE1.CNT
0327- C6 FB             1600 .1      DEC TONE1.CNT
0329- D0 19             1610         BNE .5
032B- AD 30 C0          1620         LDA SPKR     TOGGLE SPEAKER
032E- A5 FD             1630         LDA TONE1    RESET COUNTER
0330- 85 FB             1640         STA TONE1.CNT
0332- A4 FF             1650         LDY VOLUME
0334- EA                1660 .3      NOP
0335- EA                1670         NOP
0336- 88                1680         DEY
0337- 10 FB             1690         BPL .3
0339- AD 30 C0          1700         LDA SPKR     TOGGLE SPEAKER AGAIN
033C- A4 FF             1710         LDY VOLUME   EQUALIZE VOLUME DELAY
033E- EA                1720 .4      NOP
033F- C8                1730         INY
0340- C0 10             1740         CPY #16
0342- 90 FA             1750         BCC .4
```

```
0344- A0 0A     1760 .5     LDY #10       SHORT ADDITIONAL DELAY
0346- 88        1770 .6     DEY
0347- D0 FD     1780        BNE .6
0349- 20 8F 03  1790        JSR DECREMENT.DURATION
034C- 90 D9     1800        BCC .1
034E- 60        1810        RTS
                1820 *-------------------------------
                1830 *      &D <TONE1>,<DURATION>,<TONE2>
                1840 *-------------------------------
                1850 DUAL.TONES
034F- 20 84 03  1860        JSR GET.PARAMS
0352- 86 FE     1870        STX TONE2
0354- A5 FD     1880        LDA TONE1
0356- 85 FB     1890        STA TONE1.CNT
0358- A5 FE     1900        LDA TONE2
035A- 85 FC     1910        STA TONE2.CNT
035C- C6 FB     1920 .1     DEC TONE1.CNT
035E- F0 06     1930        BEQ .2        TIME TO TOGGLE
0360- 46 FF     1940        LSR VOLUME    TO EQUALIZE TIME
0362- A5 FF     1950        LDA VOLUME    TO EQUALIZE TIME
0364- 10 07     1960        BPL .3        ...ALWAYS
0366- AD 30 C0  1970 .2     LDA SPKR      TOGGLE SPEAKER
0369- A5 FD     1980        LDA TONE1     RESET COUNTER
036B- 85 FB     1990        STA TONE1.CNT
036D- C6 FC     2000 .3     DEC TONE2.CNT
036F- F0 06     2010        BEQ .4
0371- 46 FF     2020        LSR VOLUME    TO EQUALIZE TIME
0373- A5 FF     2030        LDA VOLUME    TO EQUALIZE TIME
0375- 10 07     2040        BPL .5        ...ALWAYS
0377- AD 30 C0  2050 .4     LDA SPKR      TOGGLE SPEAKER
037A- A5 FE     2060        LDA TONE2     RESET COUNTER
037C- 85 FC     2070        STA TONE2.CNT
037E- 20 8F 03  2080 .5     JSR DECREMENT.DURATION
0381- 90 D9     2090        BCC .1
0383- 60        2100        RTS
                2110 *-------------------------------
                2120 *      GET THREE PARAMETERS AFTER &T OR &D
                2130 *      1. 8-BIT VALUE, STORE IN TONE1
                2140 *      2. COMMA
                2150 *      3. 16-BIT VALUE, STORE IN DURATION
                2160 *      4. COMMA
                2170 *      5. 8-BIT VALUE, RETURN IN X-REGISTER
                2180 *-------------------------------
                2190 GET.PARAMS
0384- 20 F5 E6  2200        JSR GTBYTC    GET TONE
0387- 86 FD     2210        STX TONE1
0389- 20 BE DE  2220        JSR CHKCOM
038C- 4C 46 E7  2230        JMP GETNUM    GET DURATION AND VOLUME
                2240 *-------------------------------
                2250 *      DECREMENT DURATION
                2260 *      RETURN CARRY CLEAR IF NOT FINISHED
                2270 *-------------------------------
                2280 DECREMENT.DURATION
038F- A5 50     2290        LDA DURATION FINISHED YET?
0391- D0 08     2300        BNE .2
0393- A5 51     2310        LDA DURATION+1
0395- D0 02     2320        BNE .1
0397- 38        2330        SEC
0398- 60        2340        RTS           FINISHED
0399- C6 51     2350 .1     DEC DURATION+1
039B- C6 50     2360 .2     DEC DURATION
039D- 18        2370        CLC
039E- 60        2380        RTS
```

Correction

When I typed Rick Hatcher's code for "Hiding Things Under DOS", AAL April. 1981, page 10, I goofed. Change line 110 of the little Applesoft code from "110 POKE 40194,211" to "110 POKE 40192.211". Better yet, to reserve NP pages between the current bottom of DOS and DOS's buffers, use this code before any files are opened:

```
100 POKE 40192.PEEK(40192)-NP
110 CALL 42964
```

More About Multiplying on the 6502

You will remember Brooke Boering's article on this subject in MICRO last December; I mentioned it in AAL#5, and printed his 16x16 multiply subroutine. Now Leo J. Scanlon, author of 6502 Software Design, published an eight-page article "Multiplying by 1's and 0's" in Kilobaud Microcomputing, June 1981, pages 110-120.

If you are serious and really want to learn, this article gets down to the nuts and bolts level. Work your way through it, and you will have learned not only how to multiply, but also a lot about machine language in general. Subroutines are listed for 8x8, 16x16, and NxM multiplication, for both signed and unsigned operands.

Not to be outdone, I have written my own subroutine to multiply an M-byte multiplicand by a N-byte multiplier (both unsigned), producing a product of M+N bytes. It is written for clarity, not for size or speed (nevertheless, it is two bytes shorter than Scanlon's subroutine!).

The basic idea is to examine the bits of the multiplier one-by-one, starting on the right. If the multiplier bit = 1, the multiplicand is added in to the product, at the left end of the product register. In either case, the product register is then shifted right one bit position. The process is repeated until the multiplier is used up.

I wrote subroutines to shift the product register right one bit position, to shift the multiplier right one bit position returning the bit shifted out in the CARRY status bit, and to add the multiplicand to the product register. There is no reason these have to be subroutines; they could be coded in line, because they are only called from one place. I did it to make the overall program easier for you to follow.

The multiplication loop is coded as two loops: an outer loop for the number of bytes in the multiplier, and an inner loop for the number of bits in a byte. This allows me to have up to 255 bytes in the multiplier, just so the product (M+N bytes) is not more than 256 bytes. (Of course, if you want variables that long, you will have to move them out of page zero.)

There is one little trick you might not notice. After ACCUMULATE.PARTIAL.PRODUCT, carry will be set if the sum overflows. Then SHIFT.PRODUCT.RIGHT shifts the carry bit back into the product register, maintaining the right answer.

```
                    1000 *-------------------------------
                    1010 *      M-BYTE BY N-BYTE MULTIPLY
                    1020 *-------------------------------
0000-               1030 M      .EQ $00       # BYTES IN MULTIPLICAND
0001-               1040 N      .EQ $01       # BYTES IN MULTIPLIER
0002-               1050 PSIZE  .EQ $02       # BYTES IN PRODUCT
0003-               1060 I      .EQ $03       LOOP COUNTER
0004-               1070 J      .EQ $04       LOOP COUNTER
0090-               1080 MULTIPLICAND .EQ $90 THRU ...
00A0-               1090 MULTIPLIER   .EQ $A0 THRU ...
00B0-               1100 PRODUCT      .EQ $B0 THRU ...
                    1110 *-------------------------------
                    1120 MXN.MPY
                    1130 *-------------------------------
                    1140 *      CLEAR THE PRODUCT REGISTER
                    1150 *-------------------------------
0800- A4 00         1160        LDY M         # BYTES IN MULTIPLICAND
0802- 84 02         1170        STY PSIZE
0804- A9 00         1180        LDA #0
0806- 99 B0 00      1190 .1     STA PRODUCT,Y
0809- 88            1200        DEY
080A- 10 FA         1210        BPL .1
                    1220 *-------------------------------
                    1230 *      FOR I=M TO 1 STEP -1
                    1240 *      PSIZE = PSIZE + 1
                    1250 *      FOR J=8 TO 1 STEP -1
                    1260 *-------------------------------
080C- A5 01         1270        LDA N         # BYTES IN MULTIPLIER
080E- 85 03         1280        STA I
0810- E6 02         1290 .2     INC PSIZE
0812- A9 08         1300        LDA #8
0814- 85 04         1310        STA J
                    1320 *-------------------------------
                    1330 *      ACCUMULATE PARTIAL PRODUCT FOR NEXT BIT
                    1340 *-------------------------------
0816- 20 2A 08      1350 .3     JSR SHIFT.MULTIPLIER.RIGHT
0819- 90 03         1360        BCC .4        ZERO-BIT
081B- 20 40 08      1370        JSR ACCUMULATE.PARTIAL.PRODUCT
081E- 20 35 08      1380 .4     JSR SHIFT.PRODUCT.RIGHT
                    1390 *-------------------------------
                    1400 *      NEXT J : NEXT I
                    1410 *-------------------------------
0821- C6 04         1420        DEC J
0823- D0 F1         1430        BNE .3
0825- C6 03         1440        DEC I
0827- D0 E7         1450        BNE .2
0829- 60            1460        RTS
                    1470 *-------------------------------
                    1480 *      SHIFT MULTIPLIER RIGHT
                    1490 *-------------------------------
                    1500 SHIFT.MULTIPLIER.RIGHT
082A- A4 01         1510        LDY N         # BYTES IN MULTIPLIER
082C- A2 00         1520        LDX #0
082E- 76 A0         1530 .1     ROR MULTIPLIER,X
0830- E8            1540        INX
0831- 88            1550        DEY
0832- D0 FA         1560        BNE .1
0834- 60            1570        RTS
                    1580 *-------------------------------
                    1590 *      SHIFT PRODUCT RIGHT
                    1600 *-------------------------------
                    1610 SHIFT.PRODUCT.RIGHT
0835- A4 02         1620        LDY PSIZE     # BYTES IN PRODUCT
0837- A2 00         1630        LDX #0
0839- 76 B0         1640 .1     ROR PRODUCT,X
083B- E8            1650        INX
083C- 88            1660        DEY
083D- 10 FA         1670        BPL .1
083F- 60            1680        RTS
                    1690 *-------------------------------
                    1700 *      ACCUMULATE PARTIAL PRODUCT
                    1710 *-------------------------------
                    1720 ACCUMULATE.PARTIAL.PRODUCT
0840- A4 00         1730        LDY M
0842- 88            1740        DEY
0843- 18            1750        CLC
0844- B9 90 00      1760 .1     LDA MULTIPLICAND,Y
0847- 79 B0 00      1770        ADC PRODUCT,Y
084A- 99 B0 00      1780        STA PRODUCT,Y
084D- 88            1790        DEY
084E- 10 F4         1800        BPL .1
0850- 60            1810        RTS
```

Specialized Multiplications

Sometimes you need a multiplication routine that is not general at all. For example, when you are converting from decimal to binary, you need a routine that will multiply be ten. When you are computing the memory address of a character at a particular position on a particular line on the Apple Screen, you need to be able to multiply by 40 and 128. Other cases may come to your mind.

The subroutine BASCALC in the Apple Monitor computes the address in screen memory. Here is what it is really doing, written in Integer BASIC:

```
100 ADDR = 1024 + (LINE MOD 8)*128 + (LINE/8)*40
```

To do all that using a generalized multiply routine would take hundreds of microseconds; BASCALC takes only 40 microseconds. Here is Woz's code, with a few extra comments:

```
                1000 *-------------------------------
                1010 *      BASCALC FROM APPLE MONITOR
                1020 *-------------------------------
0028-           1030 BASL   .EQ $28
0029-           1040 BASH   .EQ $29
                1050 *-------------------------------
                1060 BASCALC
0800- 48        1070        PHA           ARG = 000ABCDE
0801- 4A        1080        LSR           (A) = 0000ABCD, E IN CARRY
0802- 29 03     1090        AND #3        (A) = 000000CD
0804- 09 04     1100        ORA #4        (A) = 000001CD
0806- 85 29     1110        STA BASH      HI-BYTE OF ADDRESS
0808- 68        1120        PLA           (A) = 000ABCDE
0809- 29 18     1130        AND #$18      (A) = 000AB000
080B- 90 02     1140        BCC .1        MERGE IN E FROM CARRY
080D- 69 7F     1150        ADC #$7F      (A) = E00AB000
080F- 85 28     1160 .1     STA BASL      BASL = E00AB000
0811- 0A        1170        ASL           (A) = 00AB0000, E IN CARRY AGAIN
0812- 0A        1180        ASL           (A) = 0AB00000, CARRY CLEAR
0813- 05 28     1190        ORA BASL      (A) = EABAB000
0815- 85 28     1200        STA BASL      LO-BYTE OF ADDRESS
0817- 60        1210        RTS
```

A subroutine to multiply by ten usually takes advantage of the fact that ten in binary is "1010". That is, 10*X is the same as 8*X + 2*X, or 2*(4*X+X). In fact, even in machines that have hardware multiply instructions, it is usually faster to multiply by ten using "shift-twice-and-add" than using the built in MPY opcode!

Here is a short piece of code which multiplies a two-byte value by ten, storing the result back in the same bytes.

```
                        1000 *-------------------------------
                        1010 *      MULTIPLY TWO BYTES BY TEN
                        1020 *-------------------------------
0000-                   1030 B0     .EQ $00
0001-                   1040 B1     .EQ $01
0800- A5 01             1050 BY.TEN LDA B1      SAVE HI-BYTE ON STACK
0802- 48                1060        PHA
0803- A5 00             1070        LDA B0      GET LO-BYTE IN A
0805- 06 00             1080        ASL B0      DOUBLE THE TWO-BYTE VALUE
0807- 26 01             1090        ROL B1
0809- 06 00             1100        ASL B0      DOUBLE IT AGAIN
080B- 26 01             1110        ROL B1
080D- 18                1120        CLC         ADD IN THE ORIGINAL VALUE
080E- 65 00             1130        ADC B0
0810- 85 00             1140        STA B0      LO-BYTE
0812- 68                1150        PLA         HI-BYTE
0813- 65 01             1160        ADC B1
0815- 85 01             1170        STA B1
0817- 06 00             1180        ASL B0      DOUBLE 5*B TO GET 10*B
0819- 26 01             1190        ROL B1
081B- 60                1200        RTS         RETURN TO CALLER
```

Another way, much less sophisticated, to multiply by ten is to simply add the number to itself nine times. If you have the S-C ASSEMBLER II Version 4.0, disassemble from $114A through $117A. You will find my subroutine for converting line numbers to binary. It is not elegant, but it does the job reasonably fast in a small amount of memory. A counter is initialized to 10; the next digit is read from the input line and converted from ASCII to binary; the number accumulator is added to the digit ten times, and the sum placed back into the number accumulator. The counter is in $52, and the number accumulator is in $50,51.

When you are converting from binary to decimal, you need to divide by ten. Or multiply by one-tenth. One-tenth written as a binary fraction is ".0001100110011001100....". Does the repetitive pattern here suggest to you a short-cut way to multiply by one-tenth? Maybe it would become even easier if we write one-tenth as 4/30 - 1/30. In decimal, to 8 places, that looks like .13333333 - .03333333 = .10000000. In binary, to 18 bits, it looks like .001000100010001000 - .000010001000100010 = .000110011001100110. See what you can come up with for a fast way to multiply a 16-bit number by one-tenth. I'll print the best version in AAL!

Commented Listing of DOS 3.3 $B800-BCFF

As I promised last month, here are the innermost routines of DOS 3.3. These are the ones which actually read and write the hardware, and are the most significantly different routines between DOS 3.2.1 and DOS 3.3.

The major difference between the two versions of DOS is the way in which data bytes are coded on the disk. DOS 3.2.1 maps 256 8-bit bytes into 410 5-bit "nybbles". DOS 3.3 maps 256 8-bit bytes into 342 6-bit "nybbles". (The term "nybble" usually means 4 bits, but Apple uses nybble to mean 5- and 6-bits also.)

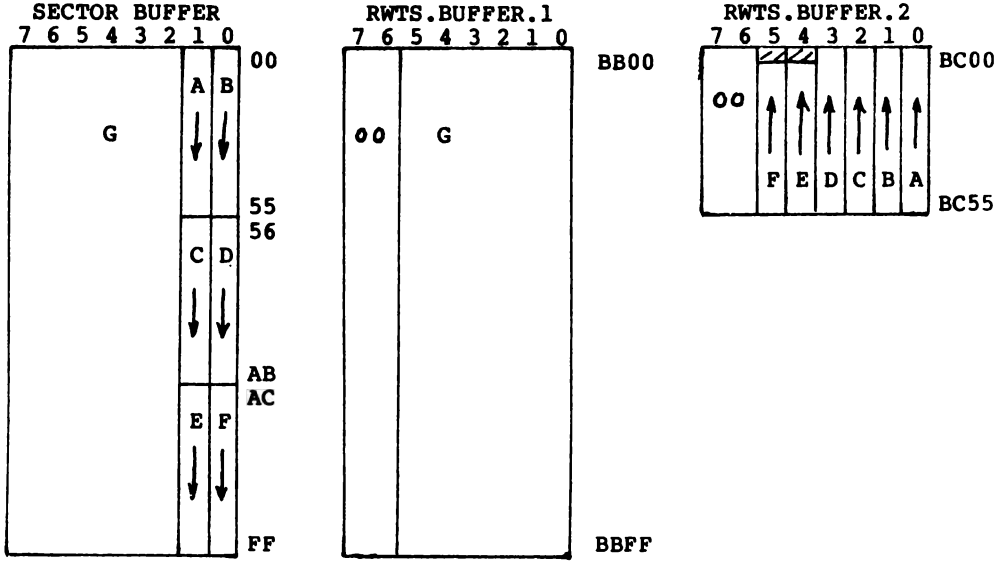
The two routines PRE.NYBBLE and POST.NYBBLE convert between memory format and disk format. The DOS 3.3 versions are much shorter and simpler than those of DOS 3.2.1, but they are still hard to visualize and explain.

To write a sector on the disk, RWTS calls PRE.NYBBLE and WRITE.SECTOR. Here is what happens:

1. The most significant 6 bits of each byte in the buffer are copied into $BB00-BBFF and right-justified with two zero-bits on the left.

2. The least significant 2 bits of each buffer byte are mapped into $BC00-BC55.

3. Each 6-bit nybble is used as an index into the NYBBLE.TABLE to pick up a corresponding 8-bit disk code. (The codes in NYBBLE.TABLE always have the first bit = 1, and never have more than two zero-bits in a row.)



To read a sector from the disk, RWTS calls READ.SECTOR and POST.NYBBLE. Here is what happens:

1. Each disk byte is converted to a 6-bit nybble and copied into the buffer from $BB00 through $BC55.

2. The nybbles in $BB00-BBFF become the most significant 6-bits of the buffer bytes.

3. The nybbles in $BC00-BC55 supply the least significant 2-bits for each buffer byte. This is the reverse of the process above.

WRITE.ADDRESS is called from FORMAT, when you are initializing a 16-sector disk. This subroutine was embedded inside FORMAT in DOS 3.2.1. READ.ADDRESS, READ.SECTOR, and WRITE.SECTOR are almost identical to the DOS 3.2.1 versions.

Short as they are, I noticed that both PRE. and POST.NYBBLE can be written more efficiently. Can you see how to save three bytes in PRE.NYBBLE, and two bytes in POST.NYBBLE?

```
                        1000 *
                        1010 *--------------------------------------
                        1020 *      DOS 3.3 DISASSEMBLY         $B800 - $BCFF
                        1030 *      COMMENTS BY BOB SANDER-CEDERLOF  5-25-81
                        1040 *--------------------------------------
                        1050        .OR $B800
                        1060        .TA $0800
                        1070 *--------------------------------------
003E-                   1080 BUF.PNTR           .EQ $3E,3F
003E-                   1090 CONST.AA           .EQ $3E
003F-                   1100 FMT.SECTOR         .EQ $3F
0041-                   1110 VOLUME             .EQ $41
0044-                   1120 TRACK.CNTR         .EQ $44
0478-                   1130 CURRENT.TRACK      .EQ $0478
                        1140 *--------------------------------------
                        1150 *      DISK CONTROLLER ADDRESSES
                        1160 *--------------------------------------
C080-                   1170 PHOFF  .EQ $C080   PHASE-OFF
C081-                   1180 PHON   .EQ $C081   PHASE-ON
C088-                   1190 MTROFF .EQ $C088   MOTOR OFF
C089-                   1200 MTRON  .EQ $C089   MOTOR ON
C08A-                   1210 DRV0EN .EQ $C08A   DRIVE 0 ENABLE
C08B-                   1220 DRV1EN .EQ $C08B   DRIVE 1 ENABLE
C08C-                   1230 Q6L    .EQ $C08C   SET Q6 LOW
C08D-                   1240 Q6H    .EQ $C08D   SET Q6 HIGH
C08E-                   1250 Q7L    .EQ $C08E   SET Q7 LOW
C08F-                   1260 Q7H    .EQ $C08F   SET Q7 HIGH
                        1270 *
                        1280 *      Q6    Q7    USE OF Q6 AND Q7 LINES
                        1290 *      ---   ---   ----------------------
                        1300 *      LOW   LOW   READ (DISK TO SHIFT REGISTER)
                        1310 *      LOW   HIGH  WRITE (SHIFT REGISTER TO DISK)
                        1320 *      HIGH  LOW   SENSE WRITE PROTECT
                        1330 *      HIGH  HIGH  LOAD SHIFT REGISTER FROM DATA BUS
                        1340 *--------------------------------------
                        1350 *
                        1360 *--------------------------------------
                        1370 *      CONVERT 256 BYTES TO 342 6-BIT NYBBLES
                        1380 *--------------------------------------
                        1390 PRE.NYBBLE
B800- A2 00             1400        LDX #0
B802- A0 02             1410        LDY #2
B804- 88                1420 .1     DEY
B805- B1 3E             1430        LDA (BUF.PNTR),Y  NEXT REAL BYTE FROM BUFFER
B807- 4A                1440        LSR
B808- 3E 00 BC          1450        ROL RWTS.BUFFER.2,X
B80B- 4A                1460        LSR
B80C- 3E 00 BC          1470        ROL RWTS.BUFFER.2,X
B80F- 99 00 BB          1480        STA RWTS.BUFFER.1,Y
B812- E8                1490        INX
B813- E0 56             1500        CPX #86
B815- 90 ED             1510        BCC .1
B817- A2 00             1520        LDX #0
B819- 98                1530        TYA
B81A- D0 E8             1540        BNE .1
B81C- A2 55             1550        LDX #85     CLEAR TOP BITS OUT OF BUFFER
B81E- BD 00 BC          1560 .2     LDA RWTS.BUFFER.2,X
B821- 29 3F             1570        AND #$3F
B823- 9D 00 BC          1580        STA RWTS.BUFFER.2,X
B826- CA                1590        DEX
B827- 10 F5             1600        BPL .2
B829- 60                1610        RTS
```

```
                        1630 *-------------------------------
                        1640 *      WRITE A SECTOR ON THE DISK FROM RWTS.BUFFER
                        1650 *-------------------------------
                        1660 WRITE.SECTOR
B82A- 38                1670         SEC             SET IN CASE OF ERROR RETURN
B82B- 86 27             1680         STX $27         SAVE SLOT #
B82D- 8E 78 06          1690         STX $0678       HERE, TOO
B830- BD 8D C0          1700         LDA Q6H,X       Q6 HIGH, Q7 LOW,
B833- BD 8E C0          1710         LDA Q7L,X       TO READ WRITE PROTECT STATUS
B836- 30 7C             1720         BMI .5          DISK IS WRITE PROTECTED
B838- AD 00 BC          1730         LDA RWTS.BUFFER.2  FIRST NYBBLE OF DATA
B83B- 85 26             1740         STA $26         SAVE IT
B83D- A9 FF             1750         LDA #$FF        SYNC BYTE
B83F- 9D 8F C0          1760         STA Q7H,X       Q6H,Q7H: (A) TO SHIFT REGISTER
B842- 1D 8C C0          1770         ORA Q6L,X       Q6L,Q7H: WRITE ON DISK
B845- 48                1780         PHA             TIME DELAYS
B846- 68                1790         PLA
B847- EA                1800         NOP
B848- A0 04             1810         LDY #4          WRITE FOUR MORE SYNC BYTES
B84A- 48                1820 .1      PHA             WASTE TIME
B84B- 68                1830         PLA
B84C- 20 B9 B8          1840         JSR WRT2        WRITE (A) ON DISK
B84F- 88                1850         DEY
B850- D0 F8             1860         BNE .1          UNTIL 4 OF THEM
B852- A9 D5             1870         LDA #$D5        WRITE DATA HEADER
B854- 20 B8 B8          1880         JSR WRT1
B857- A9 AA             1890         LDA #$AA
B859- 20 B8 B8          1900         JSR WRT1
B85C- A9 AD             1910         LDA #$AD
B85E- 20 B8 B8          1920         JSR WRT1
B861- 98                1930         TYA             A=0
B862- A0 56             1940         LDY #86         WRITE 86 NYBBLES
B864- D0 03             1950         BNE .3          ...ALWAYS
B866- B9 00 BC          1960 .2      LDA RWTS.BUFFER.2,Y  GET CURRENT NYBBLE AND
B869- 59 FF BB          1970 .3      EOR RWTS.BUFFER.2-1,Y  EOR WITH PREVIOUS NYBBLE
B86C- AA                1980         TAX             USE AS OFFSET INTO TABLE
B86D- BD 29 BA          1990         LDA NYBBLE.TABLE,X  MAP 6-BITS TO 8-BITS
B870- A6 27             2000         LDX $27         GET SLOT AGAIN
B872- 9D 8D C0          2010         STA Q6H,X       Q6H,Q7H: (A) TO SHIFT REGISTER
B875- BD 8C C0          2020         LDA Q6L,X       Q6L,Q7H: WRITE ON DISK
B878- 88                2030         DEY
B879- D0 EB             2040         BNE .2          UNTIL ALL BYTES FROM THIS BLOCK DONE
B87B- A5 26             2050         LDA $26         GET FIRST NYBBLE
B87D- EA                2060         NOP
B87E- 59 00 BB          2070 .4      EOR RWTS.BUFFER.1,Y  EOR WITH CURRENT NYBBLE
B881- AA                2080         TAX             INDEX INTO TABLE
B882- BD 29 BA          2090         LDA NYBBLE.TABLE,X  MAP TO 8-BIT VALUE
B885- AE 78 06          2100         LDX $0678       SLOT # AGAIN
B888- 9D 8D C0          2110         STA Q6H,X       Q6H,Q7L: (A) TO SHIFT REGISTER
B88B- BD 8C C0          2120         LDA Q6L,X       Q6L,Q7H: WRITE ON DISK
B88E- B9 00 BB          2130         LDA RWTS.BUFFER.1,Y  GET NYBBLE
B891- C8                2140         INY
B892- D0 EA             2150         BNE .4          MORE TO DO
B894- AA                2160         TAX             LAST NYBBLE
B895- BD 29 BA          2170         LDA NYBBLE.TABLE,X  MAP TO 8 BITS
B898- A6 27             2180         LDX $27         SLOT # AGAIN
B89A- 20 BB B8          2190         JSR WRT3        WRITE CHECK SUM ON DISK
B89D- A9 DE             2200         LDA #$DE        WRITE TRAILER
B89F- 20 B8 B8          2210         JSR WRT1
B8A2- A9 AA             2220         LDA #$AA
B8A4- 20 B8 B8          2230         JSR WRT1
B8A7- A9 EB             2240         LDA #$EB
B8A9- 20 B8 B8          2250         JSR WRT1
B8AC- A9 FF             2260         LDA #$FF
B8AE- 20 B8 B8          2270         JSR WRT1
B8B1- BD 8E C0          2280         LDA Q7L,X       Q7L
B8B4- BD 8C C0          2290 .5      LDA Q6L,X       Q6L
B8B7- 60                2300         RTS
                        2310 *-------------------------------
B8B8- 18                2320 WRT1    CLC             WAIT 2 CYCLES
B8B9- 48                2330 WRT2    PHA             WAIT 3 CYCLES
B8BA- 68                2340         PLA             WAIT 4 CYCLES
B8BB- 9D 8D C0          2350 WRT3    STA Q6H,X       Q6H,Q7H: (A) TO SHIFT REGISTER
B8BE- 1D 8C C0          2360         ORA Q6L,X       Q6L,Q7H: WRITE ON DISK
B8C1- 60                2370         RTS
```

```
                        2390 *-------------------------------
                        2400 *       CONVERT 342 6-BIT NYBBLES TO 256 BYTES
                        2410 *       (THEY ARE NOW RIGHT-JUSTIFIED IN RWTS.BUFFER)
                        2420 *-------------------------------
                        2430 POST.NYBBLE
B8C2- A0 00             2440         LDY #0
B8C4- A2 56             2450 .1      LDX #86
B8C6- CA                2460 .2      DEX
B8C7- 30 FB             2470         BMI .1
B8C9- B9 00 BB          2480         LDA RWTS.BUFFER.1,Y
B8CC- 5E 00 BC          2490         LSR RWTS.BUFFER.2,X
B8CF- 2A                2500         ROL
B8D0- 5E 00 BC          2510         LSR RWTS.BUFFER.2,X
B8D3- 2A                2520         ROL
B8D4- 91 3E             2530         STA (BUF.PNTR),Y
B8D6- C8                2540         INY
B8D7- C4 26             2550         CPY $26       (RWTS PUT 0 IN $26)
B8D9- D0 EB             2560         BNE .2
B8DB- 60                2570         RTS
                        2580 *-------------------------------
                        2590 *       READ SECTOR INTO RWTS.BUFFER
                        2600 *-------------------------------
                        2610 READ.SECTOR
B8DC- A0 20             2620         LDY #32       MUST FIND $D5 WITHIN 32 BYTES
B8DE- 88                2630 .1      DEY
B8DF- F0 61             2640         BEQ ERROR.RETURN
B8E1- BD 8C C0          2650 .2      LDA Q6L,X     READ SHIFT REGISTER
B8E4- 10 FB             2660         BPL .2        WAIT FOR FULL BYTE
B8E6- 49 D5             2670 .3      EOR #$D5      SEE IF FOUND $D5
B8E8- D0 F4             2680         BNE .1        NOT YET
B8EA- EA                2690         NOP           DELAY BEFORE NEXT READ
B8EB- BD 8C C0          2700 .4      LDA Q6L,X     READ SHIFT REGISTER
B8EE- 10 FB             2710         BPL .4        WAIT FOR FULL BYTE
B8F0- C9 AA             2720         CMP #$AA      SEE IF $AA
B8F2- D0 F2             2730         BNE .3        NO
B8F4- A0 56             2740         LDY #86       BYTE COUNT FOR LATER
B8F6- BD 8C C0          2750 .5      LDA Q6L,X     READ SHIFT REGISTER
B8F9- 10 FB             2760         BPL .5        WAIT FOR FULL BYTE
B8FB- C9 AD             2770         CMP #$AD      IS IT $AD?
B8FD- D0 E7             2780         BNE .3        NO
                        2790 *-------------------------------
B8FF- A9 00             2800         LDA #0        BEGIN CHECKSUM
B901- 88                2810 .6      DEY
B902- 84 26             2820         STY $26
B904- BC 8C C0          2830 .7      LDY Q6L,X     READ SHIFT REGISTER
B907- 10 FB             2840         BPL .7        WAIT FOR FULL BYTE
B909- 59 00 BA          2850         EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
B90C- A4 26             2860         LDY $26       BUFFER INDEX
B90E- 99 00 BC          2870         STA RWTS.BUFFER.2,Y
B911- D0 EE             2880         BNE .6
B913- 84 26             2890 .8      STY $26
B915- BC 8C C0          2900 .9      LDY Q6L,X     READ SHIFT REGISTER
B918- 10 FB             2910         BPL .9        WAIT FOR FULL BYTE
B91A- 59 00 BA          2920         EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
B91D- A4 26             2930         LDY $26
B91F- 99 00 BB          2940         STA RWTS.BUFFER.1,Y
B922- C8                2950         INY
B923- D0 EE             2960         BNE .8
B925- BC 8C C0          2970 .10     LDY Q6L,X     READ CHECKSUM
B928- 10 FB             2980         BPL .10
B92A- D9 00 BA          2990         CMP BYTE.TABLE,Y
B92D- D0 13             3000         BNE ERROR.RETURN
B92F- BD 8C C0          3010 .11     LDA Q6L,X     READ TRAILER
B932- 10 FB             3020         BPL .11
B934- C9 DE             3030         CMP #$DE
B936- D0 0A             3040         BNE ERROR.RETURN
B938- EA                3050         NOP
B939- BD 8C C0          3060 .12     LDA Q6L,X
B93C- 10 FB             3070         BPL .12
B93E- C9 AA             3080         CMP #$AA
B940- F0 5C             3090         BEQ GOOD.RETURN
                        3100 ERROR.RETURN
B942- 38                3110         SEC
B943- 60                3120         RTS
```

```
                        3140 *-------------------------------
                        3150 *       READ ADDRESS
                        3160 *-------------------------------
                        3170 READ.ADDRESS
B944- A0 FC             3180         LDY #$FC      TRY 772 TIMES (FROM $FCFC TO $10000)
B946- 84 26             3190         STY $26
B948- C8                3200 .1      INY
B949- D0 04             3210         BNE .2
B94B- E6 26             3220         INC $26
B94D- F0 F3             3230         BEQ ERROR.RETURN
B94F- BD 8C C0          3240 .2      LDA Q6L,X     READ SHIFT REGISTER
B952- 10 FB             3250         BPL .2        WAIT FOR FULL BYTE
B954- C9 D5             3260 .3      CMP #$D5      SEE IF $D5
B956- D0 F0             3270         BNE .1        NO
B958- EA                3280         NOP           DELAY
B959- BD 8C C0          3290 .4      LDA Q6L,X     READ SHIFT REGISTER
B95C- 10 FB             3300         BPL .4        WAIT FOR FULL BYTE
B95E- C9 AA             3310         CMP #$AA      SEE IF $AA
B960- D0 F2             3320         BNE .3        NO
B962- A0 03             3330         LDY #3        READ 3 BYTES LATER
B964- BD 8C C0          3340 .5      LDA Q6L,X     READ SHIFT REGISTER
B967- 10 FB             3350         BPL .5
B969- C9 96             3360         CMP #$96      SEE IF $96
B96B- D0 E7             3370         BNE .3        NO
B96D- A9 00             3380         LDA #0        START CHECK SUM
B96F- 85 27             3390 .6      STA $27
B971- BD 8C C0          3400 .7      LDA Q6L,X     READ REGISTER
B974- 10 FB             3410         BPL .7
B976- 2A                3420         ROL
B977- 85 26             3430         STA $26
B979- BD 8C C0          3440 .8      LDA Q6L,X     READ REGISTER
B97C- 10 FB             3450         BPL .8        WAIT FOR FULL BYTE
B97E- 25 26             3460         AND $26       MERGE THE NYBBLES
B980- 99 2C 00          3470         STA $2C,Y     $2C — CHECK SUM
B983- 45 27             3480         EOR $27       $2D — SECTOR
B985- 88                3490         DEY           $2E — TRACK
B986- 10 E7             3500         BPL .6        $2F — VOLUME
B988- A8                3510         TAY           TEST CHECK SUM
B989- D0 B7             3520         BNE ERROR.RETURN
B98B- BD 8C C0          3530 .9      LDA Q6L,X     READ REGISTER
B98E- 10 FB             3540         BPL .9        WAIT FOR FULL BYTE
B990- C9 DE             3550         CMP #$DE      TEST FOR VALID TRAILER
B992- D0 AE             3560         BNE ERROR.RETURN
B994- EA                3570         NOP
B995- BD 8C C0          3580 .10     LDA Q6L,X     READ REGISTER
B998- 10 FB             3590         BPL .10
B99A- C9 AA             3600         CMP #$AA
B99C- D0 A4             3610         BNE ERROR.RETURN
                        3620 GOOD.RETURN
B99E- 18                3630         CLC
B99F- 60                3640         RTS
```

```
                        3660 *-------------------------------
                        3670 *       TRACK SEEK
                        3680 *-------------------------------
                        3690 SEEK.TRACK.ABSOLUTE
B9A0- 86 2B             3700         STX $2B       CURRENT SLOT*16
B9A2- 85 2A             3710         STA $2A       SAVE TRACK #
B9A4- CD 78 04          3720         CMP CURRENT.TRACK  COMPARE TO CURRENT TRACK
B9A7- F0 53             3730         BEQ .9        ALREADY THERE
B9A9- A9 00             3740         LDA #0
B9AB- 85 26             3750         STA $26       # OF STEPS SO FAR
B9AD- AD 78 04          3760 .1      LDA CURRENT.TRACK  CURRENT TRACK NUMBER
B9B0- 85 27             3770         STA $27
B9B2- 38                3780         SEC
B9B3- E5 2A             3790         SBC $2A       DESIRED TRACK
B9B5- F0 33             3800         BEQ .6        WE HAVE ARRIVED
B9B7- B0 07             3810         BCS .2        CURRENT > DESIRED
B9B9- 49 FF             3820         EOR #$FF      CURRENT < DESIRED
B9BB- EE 78 04          3830         INC CURRENT.TRACK  INCREMENT CURRENT
B9BE- 90 05             3840         BCC .3        ...ALWAYS
B9C0- 69 FE             3850 .2      ADC #$FE      CARRY SET, SO A=A-1
B9C2- CE 78 04          3860         DEC CURRENT.TRACK  DECREMENT CURRENT TRACK
B9C5- C5 26             3870 .3      CMP $26       GET MINIMUM OF:
B9C7- 90 02             3880         BCC .4           1. # OF TRACKS TO MOVE LESS 1
B9C9- A5 26             3890         LDA $26          2. # OF ITERATIONS SO FAR
B9CB- C9 0C             3900 .4      CMP #12          3. ELEVEN
B9CD- B0 01             3910         BCS .5
B9CF- A8                3920         TAY
B9D0- 38                3930 .5      SEC           TURN PHASE ON
B9D1- 20 EE B9          3940         JSR .7
B9D4- B9 11 BA          3950         LDA ONTBL,Y   GET DELAY TIME
B9D7- 20 00 BA          3960         JSR DLY100    DELAY 100*A MICROSECONDS
B9DA- A5 27             3970         LDA $27       TRACK NUMBER
B9DC- 18                3980         CLC           TURN PHASE OFF
B9DD- 20 F1 B9          3990         JSR .8
B9E0- B9 1D BA          4000         LDA OFFTBL,Y
B9E3- 20 00 BA          4010         JSR DLY100
B9E6- E6 26             4020         INC $26       # OF STEPS SO FAR
B9E8- D0 C3             4030         BNE .1        ...ALWAYS
                        4040 *-------------------------------
B9EA- 20 00 BA          4050 .6      JSR DLY100
B9ED- 18                4060         CLC           TURN PHASE OFF
B9EE- AD 78 04          4070 .7      LDA CURRENT.TRACK
B9F1- 29 03             4080 .8      AND #3        ONLY KEEP LOW-ORDER 2 BITS
B9F3- 2A                4090         ROL           (0000 0XX0)
B9F4- 05 2B             4100         ORA $2B       (0SSS 0XX0) MERGE SLOT
B9F6- AA                4110         TAX           USE AS INDEX FOR PHASE-OFF
B9F7- BD 80 C0          4120         LDA PHOFF,X   PHASE-OFF
B9FA- A6 2B             4130         LDX $2B
B9FC- 60                4140 .9      RTS
                        4150 *-------------------------------
B9FD- AA A0 A0          4160         .HS AAA0A0    FILLER:  NOT USED IN DOS 3.3
                        4170 *-------------------------------
                        4180 *       SHORT DELAY SUBROUTINE
                        4190 *-------------------------------
BA00- A2 11             4200 DLY100  LDX #17       100*A MICROSECONDS
BA02- CA                4210 .1      DEX
BA03- D0 FD             4220         BNE .1
BA05- E6 46             4230         INC $46
BA07- D0 02             4240         BNE .2
BA09- E6 47             4250         INC $47
BA0B- 38                4260 .2      SEC
BA0C- E9 01             4270         SBC #1
BA0E- D0 F0             4280         BNE DLY100
BA10- 60                4290         RTS
                        4300 *-------------------------------
                        4310 *       DELAY TIMES FOR STEPPING MOTOR
                        4320 *-------------------------------
BA11- 01 30 28
BA14- 24 20 1E
BA17- 1D 1C 1C
BA1A- 1C 1C 1C 4330 ONTBL    .HS 01302824201E1D1C1C1C1C1C
BA1D- 70 2C 26
BA20- 22 1F 1E
BA23- 1D 1C 1C
BA26- 1C 1C 1C 4340 OFFTBL   .HS 702C26221F1E1D1C1C1C1C1C
```

```
                        4360 *-------------------------------
                        4370 *       NYBBLE TABLE
                        4380 *-------------------------------
                        4390 NYBBLE.TABLE
BA29- 96 97 9A
BA2C- 9B 9D 9E
BA2F- 9F A6 A7
BA32- AB AC AD
BA35- AE AF             4400         .HS 96979A9B9D9E9FA6A7ABACADAEAF
BA37- B2 B3 B4
BA3A- B5 B6 B7
BA3D- B9 BA BB
BA40- BC BD BE
BA43- BF CB             4410         .HS B2B3B4B5B6B7B9BABBBCBDBEBFCB
BA45- CD CE CF
BA48- D3 D6 D7
BA4B- D9 DA DB
BA4E- DC DD DE
BA51- DF E5             4420         .HS CDCECFD3D6D7D9DADBDCDDDEDFE5
BA53- E6 E7 E9
BA56- EA EB EC
BA59- ED EE EF
BA5C- F2 F3 F4
BA5F- F5 F6             4430         .HS E6E7E9EAEBECEDEEEFF2F3F4F5F6
BA61- F7 F9 FA
BA64- FB FC FD
BA67- FE FF             4440         .HS F7F9FAFBFCFDFEFF
                        4450 *-------------------------------
                        4460 *       FILLER:  $BA69 THRU $BA95 NOT USED BY DOS 3.3
                        4470 *-------------------------------
BA69-                   4480         .BS 45
                        4490 *-------------------------------
                        4500 *       BYTE TABLE
                        4510 *-------------------------------
BA00-                   4520 BYTE.TABLE .EQ *-$96
BA96- 00 01 98
BA99- 99 02 03
BA9C- 9C 04 05
BA9F- 06 A0 A1
BAA2- A2 A3             4530         .HS 0001989902039C040506A0A1A2A3
BAA4- A4 A5 07
BAA7- 08 A8 A9
BAAA- AA 09 0A
BAAD- 0B 0C 0D
BAB0- B0 B1             4540         .HS A4A50708A8A9AA090A0B0C0DB0B1
BAB2- 0E 0F 10
BAB5- 11 12 13
BAB8- B8 14 15
BABB- 16 17 18
BABE- 19 1A             4550         .HS 0E0F10111213B814151617181 91A
BAC0- C0 C1 C2
BAC3- C3 C4 C5
BAC6- C6 C7 C8
BAC9- C9 CA 1B
BACC- CC 1C             4560         .HS C0C1C2C3C4C5C6C7C8C9CA1BCC1C
BACE- 1D 1E D0
BAD1- D1 D2 1F
BAD4- D4 D5 20
BAD7- 21 D8 22
BADA- 23 24             4570         .HS 1D1ED0D1D21FD4D52021D8222324
BADC- 25 26 27
BADF- 28 E0 E1
BAE2- E2 E3 E4
BAE5- 29 2A 2B
BAE8- E8 2C             4580         .HS 25262728E0E1E2E3E4292A2BE82C
BAEA- 2D 2E 2F
BAED- 30 31 32
BAF0- F0 F1 33
BAF3- 34 35 36
BAF6- 37 38 F8
BAF9- 39 3A 3B
BAFC- 3C 3D 3E
BAFF- 3F                4590         .HS 2D2E2F303132F0F1333435363738F8393A3B3C3D3E3F
                        4600 *-------------------------------
                        4610 *       342-BYTE BUFFER FOR NYBBLES
                        4620 *-------------------------------
BB00-                   4630 RWTS.BUFFER.1 .BS 256  $BB00 - BBFF
BC00-                   4640 RWTS.BUFFER.2 .BS 86   $BC00 - BC55
                        4650 *-------------------------------
```

```
                        4670 *-------------------------------
                        4680 *       WRITE ADDRESS HEADER (CALLED BY FORMAT)
                        4690 *-------------------------------
                        4700 WRITE.ADDRESS
BC56- 38                4710         SEC             SET IN CASE OF ERROR RETURN
BC57- BD 8D C0          4720         LDA Q6H,X       Q6 HIGH, Q7 LOW,
BC5A- BD 8E C0          4730         LDA Q7L,X       TO READ WRITE PROTECT STATUS
BC5D- 30 5E             4740         BMI .2          DISK IS WRITE PROTECTED
BC5F- A9 FF             4750         LDA #$FF        SYNC BYTE
BC61- 9D 8F C0          4760         STA Q7H,X       Q6H,Q7H: (A) TO SHIFT REGISTER
BC64- DD 8C C0          4770         CMP Q6L,X       Q6L,Q7H: WRITE ON DISK
BC67- 48                4780         PHA             TIME DELAYS
BC68- 68                4790         PLA
BC69- 20 C3 BC          4800 .1      JSR .3          12 CYCLE DELAY
BC6C- 20 C3 BC          4810         JSR .3          12 CYCLE DELAY
BC6F- 9D 8D C0          4820         STA Q6H,X       WRITE ON DISK
BC72- DD 8C C0          4830         CMP Q6L,X
BC75- EA                4840         NOP
BC76- 88                4850         DEY
BC77- D0 F0             4860         BNE .1
BC79- A9 D5             4870         LDA #$D5        WRITE D5 AA 96
BC7B- 20 D5 BC          4880         JSR WRITE.BYTE.3
BC7E- A9 AA             4890         LDA #$AA
BC80- 20 D5 BC          4900         JSR WRITE.BYTE.3
BC83- A9 96             4910         LDA #$96
BC85- 20 D5 BC          4920         JSR WRITE.BYTE.3
BC88- A5 41             4930         LDA VOLUME      WRITE VOLUME, TRACK, AND SECTOR
BC8A- 20 C4 BC          4940         JSR WRITE.BYTE.1
BC8D- A5 44             4950         LDA TRACK.CNTR
BC8F- 20 C4 BC          4960         JSR WRITE.BYTE.1
BC92- A5 3F             4970         LDA FMT.SECTOR
BC94- 20 C4 BC          4980         JSR WRITE.BYTE.1
BC97- A5 41             4990         LDA VOLUME      COMPUTE CHECKSUM
BC99- 45 44             5000         EOR TRACK.CNTR
BC9B- 45 3F             5010         EOR FMT.SECTOR
BC9D- 48                5020         PHA             WRITE CHECKSUM
BC9E- 4A                5030         LSR
BC9F- 05 3E             5040         ORA CONST.AA    #$AA, FOR TIMING
BCA1- 9D 8D C0          5050         STA Q6H,X
BCA4- BD 8C C0          5060         LDA Q6L,X
BCA7- 68                5070         PLA
BCA8- 09 AA             5080         ORA #$AA
BCAA- 20 D4 BC          5090         JSR WRITE.BYTE.2
BCAD- A9 DE             5100         LDA #$DE        WRITE DE AA EB
BCAF- 20 D5 BC          5110         JSR WRITE.BYTE.3
BCB2- A9 AA             5120         LDA #$AA
BCB4- 20 D5 BC          5130         JSR WRITE.BYTE.3
BCB7- A9 EB             5140         LDA #$EB
BCB9- 20 D5 BC          5150         JSR WRITE.BYTE.3
BCBC- 18                5160         CLC
BCBD- BD 8E C0          5170 .2      LDA Q7L,X
BCC0- BD 8C C0          5180         LDA Q6L,X
BCC3- 60                5190 .3      RTS
                        5200 *-------------------------------
                        5210 *       SUBROUTINES TO WRITE BYTE ON DISK
                        5220 *-------------------------------
                        5230 WRITE.BYTE.1
BCC4- 48                5240         PHA             ADDRESS BLOCK FORMAT
BCC5- 4A                5250         LSR
BCC6- 05 3E             5260         ORA CONST.AA
BCC8- 9D 8D C0          5270         STA Q6H,X
BCCB- DD 8C C0          5280         CMP Q6L,X
BCCE- 68                5290         PLA
BCCF- EA                5300         NOP
BCD0- EA                5310         NOP
BCD1- EA                5320         NOP
BCD2- 09 AA             5330         ORA #$AA
                        5340 WRITE.BYTE.2
BCD4- EA                5350         NOP
                        5360 WRITE.BYTE.3
BCD5- EA                5370         NOP
BCD6- 48                5380         PHA
BCD7- 68                5390         PLA
BCD8- 9D 8D C0          5400         STA Q6H,X
BCDB- DD 8C C0          5410         CMP Q6L,X
BCDE- 60                5420         RTS
                        5430 *-------------------------------
                        5440 *       $BCDF THRU $BCFF IS NOT USED BY DOS 3.3
                        5450 *-------------------------------
```

Beneath Apple DOS -- A Review

If you have any interest whatsoever in DOS, be sure to buy this book! It costs $19.95 (plus shipping), from Quality Software, 6660 Reseda Blvd., Suite 105. Reseda, CA 91335. Call them up at (213) 344-6599 and give them your Master Charge or VISA number. Do it now!

Or better yet, send your check for $18 to S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. I'll mail you a copy postpaid right away! Saves you both time and money!

The authors of Beneath Apple DOS are Don Worth and Pieter Lechner. You may know Don from his adventure-like program. "Beneath Apple Manor", or from his LINKER program (both available from Quality Software).

The book is published with a plastic comb binding, and is about the same dimensions as the "Apple Assembly Line". There are 156 pages, organized into 8 chapters and 3 appendices. A comprehensive Quick Reference Card for DOS 3.3 is included. There are cartoon sketches throughout which both amuse and aid comprehension, as well as more traditional diagrams and charts and tables. A four page index helps you find whatever you need to know.

Though the book focuses on DOS 3.3, it covers all the major differences found in earlier versions. Chapter 2 is called "The Evolution of DOS", and traces features and differences from Versions 3, 3.1, 3.2, 3.2.1, and 3.3. At other points throughout the book, wherever the various versions differ, the details for each version are explained.

Chapter 3 covers diskette formatting, in much more detail than the Apple DOS manual: how bits are recorded, how 256 bytes are converted to 410 or 342 shorter bytes, how those shorter bytes are converted to encoded bytes ready to be written, how the checksum is computed and tested, how the sectors are identified around a track, all about self-sync bytes, and how sectors are interleaved.

Chapter 4 covers diskette organization: the DOS image, the Volume Table of Contents, the catalog, track/sector lists, and the format of each type of file. Some guidelines for repairing damaged diskettes are given.

Chapter 5 outlines the overall structure of DOS. The booting process is explained in a fair amount of detail. If you need more information on DOS internals, chapter 8 is for you.

Chapter 6 gives clear instructions for using RWTS from machine language programs. You may already be quite familiar with this, because: 1) it is fairly well explained in the DOS manual; 2) many articles have been published in magazines and newsletters telling you how; and 3) you have gone ahead and tried it yourself. But there is another way to get into DOS which treats files as files, but without the normal DOS overhead. Apple's FID utility uses this way in, through the so-called File Manager. Chapter 6

goes into great detail describing the File Manager, and some examples showing how to use it are given. This information has never been published before, and is well worth the price of the entire book. Chapter 6 also shows you how to talk to the disk drive directly. without any DOS at all.

Chapter 7 explains how to customize DOS, and gives the patches for four nice custom features: avoiding the language card reload, making space between DOS and its buffers, removing the pause during a long CATALOG, and changing the HELLO file start-up from RUN to BRUN or EXEC.

Chapter 8, 42 pages long, describes EVERY routine in DOS. It starts with the disk controller ROM (at C600 of your controller is in slot 6), and goes from 9D00 through BFFF subroutine by subroutine. The descriptions are in text form: no disassembled code, and no flowcharts. If you put the book beside a disassembled section of DOS, it is easily understood. Data sections are outlined also, so that you can tell what every byte is there for. The last page of chapter 8 lists all the zero-page variables used by DOS, and explains each use.

Appendix A contains five sample programs which can be used to examine and repair diskettes. They also illustrate the use of RWTS and the File Manager.

Appendix B briefly explains the philosophy of disk protection schemes. Someday someone will write a whole book on this subject. This Appendix is only four pages, so you won't find out how to create the uncrackable disk, or even how to crack it if you did.

Appendix C is an excellent glossary of terms used in the book. I estimate that about 160 words are defined.

The authors list five good reasons why they wrote Beneath Apple DOS; no, six:

1. To show direct assembly language access to DOS.
2. To help you fix clobbered diskettes.
3. To correct errors and ommissions in the Apple manuals.
4. To provide complete infomation on diskette formatting and DOS internal operation.
5. To allow you to customize DOS to fit your needs.
6. To make the authors a lot of money.

They have done an excellent job with the first five objectives, and I think number 6 will be met as well.

Apple Assembly Line is published monthly by S-C SOFTWARE, P. O. Box 5537, Richardson, TX 75080. 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).