Apple Assembly Line

Volume 1 -- Issue 8 May, 1981

In This Issue...

Save Your Fingers, Save Your Eyes

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), and QD#2 covers AAL issues 4-6 (January thru March 1981). QD#3 will be out at the end of May, covering issues 7-9. Some AAL subscribers have chosen to set up a standing order for the Quarterly Disks, so they get them as soon as they are ready.

Not only does it save you a lot of typing time. You also are saved the hours you might spend looking for the inadvertant changes you made while you typed!

Another Utility from RAK-WARE

Bob Kovacs is sure keeping busy! Last month he announced the Cross Reference Utility which works with your S-C ASSEMBLER II source programs. This month he has a Global Search & Replace Utility ready (see his ad on page 4). It is a nice companion to his disassembler, because it gives you a fast way to change all the labels made up by the disassembler into meaningful names.

If You Need Disks...

For a limited time, I am able to offer you a good price on Verbatim DataLife disks. These are bulk packaged, 20 to a pack, with no labels and with white sleeves. They are the same ones I use myself. I will send you a package of 20 for only $50.

Hi-Res SCRN Function for Applesoft

Apple's Lo-Res graphics capability includes a SCRN(X,Y) function, to determine the color currently on the screen at the given X,Y point. For some reason they did not provide the corresponding HSCRN(X,Y) function for Hi-Res graphics.

The following program implements the HSCRN function using the "&" character. If you write the statement "& HSCRN (A=X,Y)", this program will store either a 1 or a 0 into the variable A. The value 0 will be stored in A if there is not a spot plotted at X,Y; the value 1 will be stored if there is a spot.

Note that HPLOT(X,Y) may not result in a spot being plotted at X,Y; it depends on the HCOLOR you have set. If the HCOLOR is white, a spot will always be plotted; if it is black, a spot will always be erased; the other four colors may or may not plot a spot, depending on position and color.

The &HSCRN statement does not return the actual color, because that is MUCH more difficult to determine. The actual color depends on: whether the adjacent spots are on or off; whether X,Y is in an even or odd byte; whether X,Y is in an even or odd bit; and whether the sign bit of the byte is on or off. If you decide to add the capability to return a color value (0-7), send me a copy for this newsletter!

```
                  1000 *--------------------------------
                  1010 *       HI-RES SCRN FUNCTION
                  1020 *
                  1030 *       & HSCRN( A=X,Y )
                  1040 *       X,Y DEFINES THE SPOT
                  1050 *       A RECEIVES 0 OR 1
                  1060 *--------------------------------
                  1070         .OR $300
                  1080         .TF B.HIRES SCRN
                  1090 *--------------------------------
03F5-             1100 AMPERSAND.VECTOR     .EQ $3F5
                  1110 *--------------------------------
00B1-             1120 CHRGET               .EQ $00B1
00B7-             1130 CHRGOT               .EQ $00B7
DEC0-             1140 SYNCHR               .EQ $DEC0
DEC9-             1150 SYNTAX.ERROR         .EQ $DEC9
DFE3-             1160 PTRGET               .EQ $DFE3
E301-             1170 SNGFLT               .EQ $E301
F411-             1180 HPOSN                .EQ $F411
F6B9-             1190 HFNS                 .EQ $F6B9
                  1200 *--------------------------------
0011-             1210 VALUE.TYPE           .EQ $11
0026-             1220 HPNTR                .EQ $26
0030-             1230 HMASK                .EQ $30
0085-             1240 FORMULA.PNTR         .EQ $85
                  1250 *--------------------------------
00D0-             1260 TOKEN.EQUALS         .EQ $D0
00D7-             1270 TOKEN.SCRN           .EQ $D7
                  1280 *--------------------------------
                  1290 *       SETUP AMPERSAND VECTOR
                  1300 *--------------------------------
0300- A9 4C       1310 SETUP   LDA #$4C     JMP OPCODE
0302- 8D F5 03    1320         STA AMPERSAND.VECTOR
0305- A9 10       1330         LDA #HSCRN
0307- 8D F6 03    1340         STA AMPERSAND.VECTOR+1
030A- A9 03       1350         LDA /HSCRN
030C- 8D F7 03    1360         STA AMPERSAND.VECTOR+2
030F- 60          1370         RTS
                  1380 *--------------------------------
                  1390 *       HSCRN FUNCTION
                  1400 *--------------------------------
0310- A9 48       1410 HSCRN   LDA #'H       TEST FOR "HSCRN("
0312- 20 C0 DE    1420         JSR SYNCHR        FIRST LETTER "H"
0315- A9 D7       1430         LDA #TOKEN.SCRN   AND THEN TOKEN "SCRN("
0317- 20 C0 DE    1440         JSR SYNCHR
031A- 20 E3 DF    1450         JSR PTRGET   SCAN THE VARIABLE NAME
031D- 85 85       1460         STA FORMULA.PNTR  SAVE ITS POINTER ADDRESS
031F- 84 86       1470         STY FORMULA.PNTR+1
0321- A9 D0       1480         LDA #TOKEN.EQUALS CHECK FOR "="
0323- 20 C0 DE    1490         JSR SYNCHR
0326- A5 12       1500         LDA VALUE.TYPE+1  SAVE VARIABLE TYPE ON STACK
0328- 48          1510         PHA
0329- A5 11       1520         LDA VALUE.TYPE
032B- 48          1530         PHA
032C- 20 B9 F6    1540         JSR HFNS      SCAN "X,Y" EXPRESSIONS
032F- 20 11 F4    1550         JSR HPOSN     SET UP BASE, Y-REG, AND MASK
0332- 20 B7 00    1560         JSR CHRGOT    CHECK FOR FINAL ")"
0335- C9 29       1570         CMP #')
0337- D0 12       1580         BNE .2        SYNTAX ERROR IF NOT THERE!
0339- 20 B1 00    1590         JSR CHRGET    POSITION FOR NEXT STATEMENT
033C- A5 30       1600         LDA HMASK     ISOLATE SPOT AT X,Y
033E- 31 26       1610         AND (HPNTR),Y
0340- F0 02       1620         BEQ .1        SPOT IS OFF, RETURN ZERO
0342- A9 01       1630         LDA #1        SPOT IS ON, RETURN 1
0344- A8          1640 .1      TAY
0345- 20 01 E3    1650         JSR SNGFLT    CONVERT BYTE TO REAL VALUE
0348- 4C 5B DA    1660         JMP $DA5B     STORE IN VARIABLE, AND KEEP GOING!
                  1665 *--------------------------------
034B- 4C C9 DE    1670 .2      JMP SYNTAX.ERROR
```

```
5  PRINT  CHR$ (4)"BLOAD B.HIRES SCRN": CALL 768
10 HGR : HCOLOR= 3: HPLOT 0,0
20 FOR I = 1 TO 10
30 X =  RND (1) * 40:Y =  RND (1) * 40
40 HPLOT  TO X,Y: NEXT
60 FOR X = 0 TO 39: FOR Y = 0 TO 39
70 & H SCRN( A = X,Y): COLOR= 15 * A
80 PLOT X,Y: NEXT : NEXT
90 POKE  - 16298,0
100 GET A$: POKE  - 16297,0: GET A$: GOTO 90
```

Conquering Paddle Jitter.....Brooke Boering

A well-known problem with the paddles supplied with the Apple (at least they USED to be supplied!) concerns their tendency to rock back and forth between two adjacent values. "Jittering" like this can cause problems unless accuracy is unimportant, or unless the effect is somehow pleasing.

One solution to the jitter problem is to force the new paddle reading to move at least two increments from the prior reading. This words, but at the price of lower resolution. Also, it can have subtle side-effects.

A better solution is to keep track of the previous direction of movement, and enforcing the "rule of two" only if the direction is reversed.

The following program demonstrates my solution. It is set up to work with Applesoft, but it would be rather simple to make it directly callable from your own assembly language routines. To use from Applesoft, POKE the paddle number (0-3) at 768, CALL 770, and read the paddle value with PEEK(769).

I set up the following Applesoft program to test the routine, and to compare it with normal paddle readings:

```
10 POKE 768,0:CALL 770:PRINT PEEK(769):GOTO10
20 PRINT PDL(0):GOTO20
```

I typed RUN 20 and set the paddle to a jittery position. Then I typed control-C and RUN 10 to test the smoothing subroutine. The program really works!

```
                  1000 *--------------------------------
                  1010 *      PADDLE JITTER SMOOTHER
                  1020 *
                  1030 *      POKE 768,<PADDLE NUMBER>   0, 1, 2, OR 3
                  1040 *      CALL 770
                  1050 *      P=PEEK(769)   PADDLE VALUE 0-255
                  1060 *--------------------------------
 FB1E-            1070 MON.PREAD  .EQ $FB1E SUBROUTINE TO READ PADDLE
                  1080 *--------------------------------
                  1090        .OR $300
                  1100 *--------------------------------
 0300-            1110 PADDLE.NUMBER       .BS 1
 0301-            1120 PADDLE.VALUE        .BS 1
                  1130 *--------------------------------
                  1140 PADDLE.JITTER.SMOOTHER
 0302- AD 00 03   1150        LDA PADDLE.NUMBER
 0305- 29 03      1160        AND #3       BE CERTAIN 0>=PDL#>=3
 0307- AA         1170        TAX
 0308- 20 1E FB   1180        JSR MON.PREAD READ PADDLE VALUE
 030B- 98         1190        TYA          SAVE IN A-REG TOO
 030C- CC 39 03   1200        CPY PADDLE.VALUE.1
 030F- F0 24      1210        BEQ .8       SAME, RETURN THIS VALUE
 0311- AE 39 03   1220        LDX PADDLE.VALUE.1  DETERMINE PREVIOUS DIRECTION
 0314- EC 3A 03   1230        CPX PADDLE.VALUE.2
 0317- B0 08      1240        BCS .2       IT WAS INCREASING
                  1250 *--------------------------------
                  1260 *   IT WAS DECREASING...
                  1270 *--------------------------------
 0319- CC 39 03   1280        CPY PADDLE.VALUE.1  WHAT IS CURRENT DIRECTION?
 031C- 90 11      1290        BCC .6       STILL DECREASING, SO ACCEPT IT
 031E- 88         1300        DEY          SEE IF ONLY 1 STEP
 031F- B0 06      1310        BCS .5       ...ALWAYS
                  1320 *--------------------------------
                  1330 *   IT WAS INCREASING...
 0321- CC 39 03   1340 .2     CPY PADDLE.VALUE.1  DETERMINE CURRENT DIRECTION
 0324- B0 09      1350        BCS .6       STILL INCREASING, SO ACCEPT IT
 0326- C8         1360        INY          SEE IF ONLY 1 STEP
                  1370 *--------------------------------
                  1380 *   REVERSED DIRECTION
                  1390 *--------------------------------
 0327- CC 39 03   1400 .5     CPY PADDLE.VALUE.1  IF SAME NOW, IGNORE IT
 032A- D0 03      1410        BNE .6       USE NEW VALUE
 032C- 8A         1420        TXA          USE PREVIOUS VALUE
 032D- B0 06      1430        BCS .8       ...ALWAYS
                  1440 *--------------------------------
                  1450 *   ACCEPT NEW READING
                  1460 *--------------------------------
 032F- 8E 3A 03   1470 .6     STX PADDLE.VALUE.2  OLDEST READING
 0332- 8D 39 03   1480        STA PADDLE.VALUE.1  PREVIOUS READING
                  1490 *--------------------------------
 0335- 8D 01 03   1500 .8     STA PADDLE.VALUE    CURRENT READING
 0338- 60         1510        RTS
                  1520 *--------------------------------
 0339- 00         1530 PADDLE.VALUE.1  .DA #0
 033A- 00         1540 PADDLE.VALUE.2  .DA #0
                  1550 *--------------------------------
```

SYMBOL TABLE

```
FB1E- MON.PREAD
0302- PADDLE.JITTER.SMOOTHER
.02=0321, .05=0327, .06=032F, .08=0335
0300- PADDLE.NUMBER
0301- PADDLE.VALUE
0339- PADDLE.VALUE.1
033A- PADDLE.VALUE.2
```

Don't Be Shiftless

Now for another article aimed at that half of you who are really new to 6502 assembly language!

Sliding the bits in a byte back and forth, to the left or the right, is one of the traditional things computers like to do. Big computers have fancy instructions for doing it in many different ways, with special effects along the way. The 6502 only has four "shift" opcodes, so we have to work harder to get all the types of shifting our programs need.

Why shift anything? For various reasons, to suit your fancy. Since data in a byte is normally construed as a binary number, a shift left one bit-position will double the value and a shift right one bit-position will halve the value. If it is important to isolate a particular bit field out of a byte, and then to left or right justify the value which was stored in that field so that testing or arithmetic can be performed, you need shifting instructions. In order to implement multiply and divide on the 6502 you need shifting instructions. To position data for insertion into a bit field within a byte you need to shift. And more.

Show me a picture of a shift. Well, the 6502 makes that easy, because it is limited to shifting a byte to the left or the right, one bit-position at a time.

First let's look at the LSR instruction, which shifts right one bit-position. "LSR" stands for "Logical Shift Right". LSR will shift the contents of a byte one bit-position to the right, like this:

Old value: 1 0 0 1 1 1 0 1

<Do LSR>

New value: 0 1 0 0 1 1 1 0

LSR shifts in a zero-bit on the left end; the bit that is shifted out the right end goes into the CARRY status bit.

In the sample above the binary value of the old byte is $9D in hex, or 157 decimal. After shifting, the value is $4E hex or 78 decimal (157/2 = 78.5).

The fact the the bit shifted "out" goes into the CARRY status bit makes it possible to test what that bit was. For example, if you need to test a byte to see if it is even or odd, you can LSR it once and then do BCC or BCS to test the carry bit. If carry is set, the number was odd; if clear, it was even. The bit stored in CARRY can have other uses we will discover later.

Now let's see the ASL ("Arithmetic Shift Left") do its thing. It will shift a byte one bit-position to the left, with a zero coming in the right end. The bit shifted out the left end goes into the CARRY status bit. See the similarity to the LSR instruction?

```
                    CARRY
Old value:   0 0 0 1 1 1 0 1
<Do ASL>                          0
New value:   0 0 1 1 1 0 1 0
```

Note that the value is doubled; $1D (29) became $3A (58). This will not always be true; if the bit shifted out was a 1-bit, it will be doubled modulo 256. Integer BASIC users will know what that means, because they have the MOD function. For Applesoft-only people, it will mean here that the result is 256 less than the doubled value should be. Let's see an example: shifting 10011101 with ASL produces 00111010; $9D (157) becomes $3A (58), which is 256 less than 2*157.

<u>More about the carry bit.</u> Suppose I want to see if the third bit in a byte is 1 or 0. If the bit positions are numbered left to right from 7 down to 0 (like this: 7 6 5 4 3 2 1 0), I want to test bit 5. If I do three ASL's in a row, bit 5 will be in the CARRY status bit, and I can test it. Or, I could do two ASL's in a row, and look at the MINUS status bit. After a shift, the MINUS status bit is set if the new bit 7 is a 1-bit, or cleared if bit 7 is a 0-bit. The BPL and BMI instructions test the MINUS status bit.

<u>There are two more shift instructions to look at: ROL and ROR.</u> "ROL" is pronounced like a type of bread you eat at dinner, and "ROR" like the noise those giant cats at the zoo make. "ROL" stands for "Rotate One Left"; "ROR" means "Rotate One Right". They work just like LSR and ASL, except for what is shifted in to the byte. LSR shifts a zero-bit in the left end, and ASL shifts a zero-bit in the right end. ROL and ROR shift the old CARRY status bit in, just before the shifted-out bit comes into the CARRY bit.

```
                     Byte          Carry
Old value:   1 0 0 1 1 1 0 1        1
<Do ROL>
New value:   0 0 1 1 1 0 1 1        1
<Do ROL>
New value:   0 1 1 1 0 1 1 1        0
<Do ROL>
New value:   1 1 1 0 1 1 1 0        0
<Do ROR>
New value:   0 1 1 1 0 1 1 1        0
<Do ROR>
New value:   0 0 1 1 1 0 1 1        1
```

```
<Do ROR>
                 ___________________
                /                   \
New value:      1 0 0 1 1 1 0 1     1
<Do ROR>
                 ___________________
                /                   \
New value:      1 1 0 0 1 1 1 0     1
```

What about shifting values which take two bytes? We can do it using combinations of the four opcodes. Suppose you want to shift a 16-bit value stored at $1234 and $1235 left one bit-position. You want a zero to enter the least significant bit position, which is bit 0 of $1234. You want the most significant bit, bit 7 of $1235, to be in CARRY when you are through. Here is the program:

```
ASL $1234     0 --> bit 0, bit 7 --> CARRY
ROL $1235     CARRY --> bit 0, then bit 7 into CARRY
```

Simple, isn't it!

Addressing Modes. The four shift instructions all have the same five addressing modes. There is a one-byte form which shifts the A-register. Some assemblers write this as "ASL A", and don't allow "A" to be used as a label elsewhere. The S-C ASSEMBLER II writes it as just "ASL", so you can use "A" as a label elsewhere if you wish. The other addressing modes are: zero page direct; zero page,X; absolute; and absolute,X. No indirect modes, or indexing by Y modes are available.

[If you remember the article a few months ago about the "secret" opcodes, you will also remember that the two indirect-indexed modes and the absolute,Y mode are available if you don't mind what happens to the A-register after the shift. Or, if what does happen is something you also wanted. You might look up the article.]

Some real examples. The Apple Monitor ROM has some good examples in it. Disassemble (or look in the Monitor listing in the Reference Manual) at $FBC1 (the BASCALC subroutine. If you have the old Monitor ROMs, the multiply and divide subroutines at $FB60 and $FB81 are good examples. The PRBYTE subroutine at $FDDA uses four LSR's to get at the first hex digit. The subroutine DIG at $FF8A is used to convert ascii hex numbers to binary. Let's look at that one here:

```
FF8A: A2 03     DIG     LDX #$03    LOOP 4 TIMES
FF8C: 0A                ASL         LEFT JUSTIFY DIGIT VALUE
FF8D: 0A                ASL
FF8E: 0A                ASL
FF8F: 0A                ASL
FF90: 0A        NXTBIT  ASL         SHIFT DIGIT INTO A2L,A2H
FF91: 26 3E             ROL A2L
FF93: 26 3F             ROL A2H
FF95: CA                DEX
FF96: 10 F8             BPL NXTBIT
```

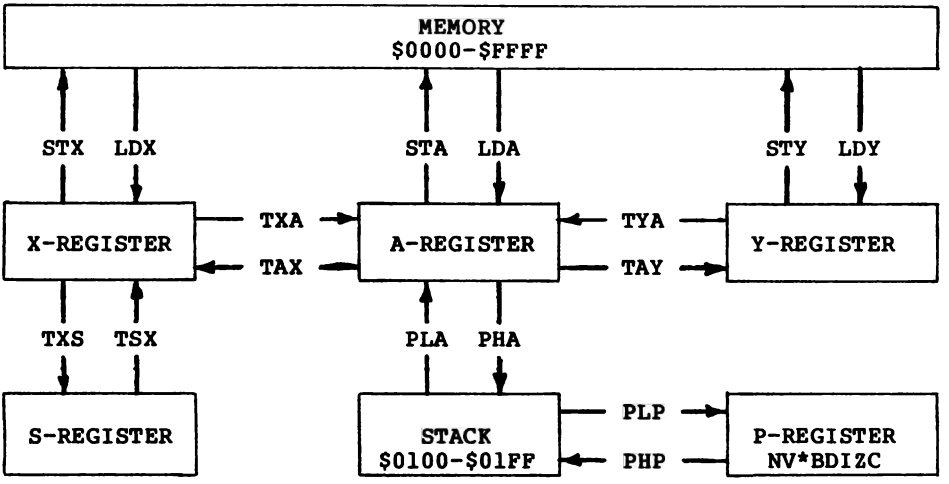
The ASCII value of the hex digit has already been modified so that the digit's value is in bits 3-0. The first four ASL's shift those 4 bits up to bits 7-4. The next ASL shifts the top bit into CARRY, and then the two ROL's shift that bit into the 16-bit value at A2L and A2H. The ASL-ROL-ROL loop is done four times, so all four bits are shifted into A2L,A2H.

In the Applesoft ROMs there is a subroutine which shifts a 32-bit value right any number of bit-positions. The subroutine is used in the floating point arithmetic package to adjust mantissas. It has the interesting feature (for speed's sake) of shifting 8 bits at a time until the shift count is less than 8. This is done by moving bytes with LDY-STY pairs. The code is at $E8DC thru $E912. The normal entry point is at $E8F0, with the number of bit-positions to be shifted in the A-register as a negative number, and with CARRY clear. The code above $E8F0 shifts right by bytes, and the code after $E8F0 shifts right by bits. The data to be shifted is in page zero, offset by the value in the X-register.

A somewhat similar subroutine is used to normalize the mantissa after a calculation. "Normalize" means to shift the mantissa left until the most significant bit is a one-bit. This code is at $E82E-E854 and $E874-E880. The first portion shifts left by bytes until the leading byte is non-zero (or until it has been determined that the whole value is zero). Once the leading byte is found to be non-zero, the second portion of code shifts left by bits until the leading bit is 1. The number of bit-positions shifted is counted as the subroutine moves along, and that value is subtracted from the exponent value of the floating point number ($E882-E88B).

Disassemble the routines I have pointed out in the various ROMs, and study them a while. Then try writing some of your own examples. Here is an assignment: write a subroutine that will shift a 16-bit value left or right from 0-15 bit positions. The value to be shifted is in page zero at $9D and $9E. The shift count is in the A-register. If the value in A is zero, return without doing anything. If A is negative, it indicates a shift right. If A is positive, it means to shift left. Okay? Give it a try!

TRANSFER OPERATIONS



OTHER OPERATIONS

	A-REGISTER	X-REGISTER	Y-REGISTER	MEMORY
Arithmetic:	ADC	INX	INY	INC
	SBC	DEX	DEY	DEC
Logical:	AND	---	---	BIT
	ORA	---	---	---
	EOR	---	---	---
Shift:	ASL	---	---	ASL
	LSR	---	---	LSR
	ROL	---	---	ROL
	ROR	---	---	ROR
Compare:	CMP	CPX	CPY	---

Status:	SET	CLEAR	BRANCH
CARRY	SEC	CLC	BCC, BCS
OVERFLOW	---	CLV	BVC, BVS
DECIMAL	SED	CLD	--------
INTERRUPT	SEI	CLI	--------
ZERO	---	---	BEQ, BNE
MINUS	---	---	BPL, BMI

Jump: JMP, JSR

Return: RTS, RTI

Other: NOP, BRK

Commented Listing of DOS 3.2.1 $B800-BCFF

Here is the third installment of DOS disassembly, covering the routines called by RWTS.

There are six major subroutines between $B800 and BCFF. PRE.NYBBLE and POST.NYBBLE convert between memory format and disk format. READ.ADDRESS reads the next address header. READ.SECTOR reads a sector, and WRITE.SECTOR writes one. SEEK.TRACK.ABSOLUTE moves the head in or out to the desired track. With the sole exception of initializing a disk, all actual disk I/O is done by these six subroutines.

The bits that are written on the disk are considerably different from those in memory. Some computer systems make the transformation with expensive hardware controllers, but Wozniak's unique system does most of the work in software. The 13-sector controller cannot read accurately data which has two or more consecutive zero-bits. Of course, almost every byte you want to write has two or more zero-bits in a row! Therefore the software must encode the bytes you want to write.

One way to encode the bytes is to take four bits at a time, and interleave them with "clock" bits. In fact, the data in the address headers is recorded this way. For example, to record the byte "xyxyxyxy" in an address header, the two bytes "1x1x1x1x" and "1y1y1y1y" will be written. This means a 256-byte sector will take 512 bytes on the disk surface (plus header and trailer).

DOS 3.2.1 (and previous versions) use a more elaborate scheme. Each 256-byte sector is recorded as 410 bytes on the disk surface. The subroutine PRE.NYBBLE converts the 256-byte buffer to 410 bytes of 5-bits each. then the 5-bit values are converted to 8-bit values from NYBBLE.TABLE. These 8-bit values are chosen carefully; they have the following properties: 1) the first bit is "1"; 2) no consecutive zero-bits; and 3) the values $AA and $D5 are not used. As a sector is read back into memory, BYTE.TABLE is used to convert the 8-bit codes back to 5-bit values. POST.NYBBLE converts the 410 5-bit values back to 256 8-bit bytes.

In case you are curious, PRE.NYBBLE moves the bits from 256-bytes to 410 bytes like this:

1. The first 5 bytes are rearranged into 8 bytes:

5 input bytes								8 output bytes							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
A	A	A	A	A	B	B	B	0	0	0	A	A	A	A	A
C	C	C	C	C	D	D	D	0	0	0	C	C	C	C	C
E	E	E	E	E	F	F	F	0	0	0	E	E	E	E	E
G	G	G	G	G	H	I	J	0	0	0	G	G	G	G	G
K	K	K	K	K	L	M	N	0	0	0	K	K	K	K	K
								0	0	0	B	B	B	H	L
								0	0	0	D	D	D	I	M
								0	0	0	F	F	F	J	N

2. The 8 bytes are stored at the end of the 8 sections (at BB32, BB65, BB98, BBCB, BC32, BC65, AND BC98).

3. The second group of 5 bytes is rearranged into 8 bytes, and stored right before the first 8 (at BB31, BB64, ..., BC97).

4. The next 49 groups of 5 bytes are treated in the same way, with the last group being stored at BB00, BB33, BB66, BB99, BBCC, BC00, BC33, AND BC66.

5. The top 5 bits of the last byte are stored at BBFF, and the bottom 3 bits of the last byte are stored at BC99.

DOS 3.3 uses an even better scheme, but it requires a change in the controller ROMs. The change to one ROM gives you a different boot program; the other ROM makes the controller able to read two consecutive zero-bits accurately. (Note that SOME controller-drive combinations may be able to read two zero-bits in a row accurately WITHOUT the new ROM. Anyway, mine works!) DOS 3.3 converts the 256 bytes to 342 6-bit values; since each sector is shorter, more sectors can be written in each track. I may publish the disassembly of these same subroutines in the DOS 3.3 version next month.

Remember that DOS 3.2.1 puts 13-sectors on each track, with each sector having this format: sync bytes, address header, sync bytes, data block. Sync bytes are written to automatically synchronize the reading process, so that we can be sure we are not splitting bytes. Each sync byte is 8 one-bits followed by 1 zero-bit. The address header is 14-bytes long on the disk surface, and looks like this (in hex): D5 AA B5 vv vv ss ss tt tt cc cc DE AA EB. "vv vv" stands for the two bytes used to record the volume number; "ss ss" is the sector number; "tt tt" is the track number; and "cc cc" is the checksum of the volume, track, and sector. The data block is like this: D5 AA AD <410 bytes of data> <checksum> DE AA EB.

```
                1010 *--------------------------------
                1020 *      DOS 3.2.1 DISASSEMBLY      $B800 - $BCFF
                1025 *      COMMENTS BY BOB SANDER-CEDERLOF  4-28-81
                1030 *--------------------------------
                1040        .OR $B800
                1050        .TA $0800
                1060 *--------------------------------
003E-           1070 BUF.PNTR   .EQ $3E,3F
0478-           1080 CURRENT.TRACK        .EQ $0478
                1090 *--------------------------------
                1100 *      DISK CONTROLLER ADDRESSES
                1110 *--------------------------------
C080-           1120 PHOFF  .EQ $C080     PHASE-OFF
C081-           1130 PHON   .EQ $C081     PHASE-ON
C088-           1140 MTROFF .EQ $C088     MOTOR OFF
C089-           1150 MTRON  .EQ $C089     MOTOR ON
C08A-           1160 DRV0EN .EQ $C08A     DRIVE 0 ENABLE
C08B-           1170 DRV1EN .EQ $C08B     DRIVE 1 ENABLE
C08C-           1180 Q6L    .EQ $C08C     SET Q6 LOW
C08D-           1190 Q6H    .EQ $C08D     SET Q6 HIGH
C08E-           1200 Q7L    .EQ $C08E     SET Q7 LOW
C08F-           1210 Q7H    .EQ $C08F     SET Q7 HIGH
                1220 *
                1230 *      Q6    Q7      USE OF Q6 AND Q7 LINES
                1240 *      ---   ---     ----------------------
                1250 *      LOW   LOW     READ (DISK TO SHIFT REGISTER)
                1260 *      LOW   HIGH    WRITE (SHIFT REGISTER TO DISK)
                1270 *      HIGH  LOW     SENSE WRITE PROTECT
                1280 *      HIGH  HIGH    LOAD SHIFT REGISTER FROM DATA BUS
```

```
                1290 *
                1300 *--------------------------------
                1310 *      CONVERT 256 BYTES TO 410 5-BIT NYBBLES
                1320 *--------------------------------
                1330 PRE.NYBBLE
B800- A2 32     1340        LDX #50       51 BYTES PER SECTION
B802- A0 00     1350        LDY #0        INDEX INTO 256-BYTE BUFFER
                1360 *--BUFFER PART 1, SECTION 1-----
B804- B1 3E     1370 .1     LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
B806- 85 26     1380        STA $26       SAVE HERE FOR LOWER 3 BITS
B808- 4A        1390        LSR           USE TOP 5 BITS
B809- 4A        1400        LSR
B80A- 4A        1410        LSR
B80B- 9D 00 BB  1420        STA RWTS.BUFFER.1.1,X
                1430 *--BUFFER PART 1, SECTION 2-----
B80E- C8        1440        INY           NEXT REAL BYTE
B80F- B1 3E     1450        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
B811- 85 27     1460        STA $27       SAVE HERE FOR LOWER 3 BITS
B813- 4A        1470        LSR           USE TOP 5 BITS
B814- 4A        1480        LSR
B815- 4A        1490        LSR
B816- 9D 33 BB  1500        STA RWTS.BUFFER.1.2,X
                1510 *--BUFFER PART 1, SECTION 3-----
B819- C8        1520        INY           NEXT REAL BYTE
B81A- B1 3E     1530        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
B81C- 85 2A     1540        STA $2A       SAVE FOR LOWER 3 BITS
B81E- 4A        1550        LSR
B81F- 4A        1560        LSR           USE TOP 5 BITS
B820- 4A        1570        LSR
B821- 9D 66 BB  1580        STA RWTS.BUFFER.1.3,X
                1590 *--BUFFER PART 1, SECTION 4-----
B824- C8        1600        INY           NEXT REAL BYTE
B825- B1 3E     1610        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
B827- 4A        1620        LSR           USE TOP 5 BITS
B828- 26 2A     1630        ROL $2A       BIT 0 INTO $2A
B82A- 4A        1640        LSR
B82B- 26 27     1650        ROL $27       BIT 1 INTO $27
B82D- 4A        1660        LSR
B82E- 26 26     1670        ROL $26       BIT 2 INTO $26
B830- 9D 99 BB  1680        STA RWTS.BUFFER.1.4,X
                1690 *--BUFFER PART 1, SECTION 5-----
B833- C8        1700        INY           NEXT REAL BYTE
B834- B1 3E     1710        LDA (BUF.PNTR),Y  GET BYTE FROM BUFFER
B836- 4A        1720        LSR           USE TOP 5 BITS
B837- 26 2A     1730        ROL $2A       BIT 0 INTO $2A
B839- 4A        1740        LSR
B83A- 26 27     1750        ROL $27       BIT 1 INTO $27
B83C- 4A        1760        LSR           HOLD BIT 2 IN CARRY-BIT
B83D- 9D CC BB  1770        STA RWTS.BUFFER.1.5,X
                1780 *--BUFFER PART 2, SECTION 0-----
B840- A5 26     1790        LDA $26       APPEND BIT 2 TO $26
B842- 2A        1800        ROL
B843- 29 1F     1810        AND #$1F      5-BIT MASK
B845- 9D 00 BC  1820        STA RWTS.BUFFER.2.1,X
                1830 *--BUFFER PART 2, SECTION 1-----
B848- A5 27     1840        LDA $27
B84A- 29 1F     1850        AND #$1F
B84C- 9D 33 BC  1860        STA RWTS.BUFFER.2.2,X
                1870 *--BUFFER PART 2, SECTION 2-----
B84F- A5 2A     1880        LDA $2A
B851- 29 1F     1890        AND #$1F
B853- 9D 66 BC  1900        STA RWTS.BUFFER.2.3,X
                1910 *--------------------------------
B856- C8        1920        INY           NEXT REAL BYTE
B857- CA        1930        DEX           NEXT BYTE IN EACH SECTION
B858- 10 AA     1940        BPL .1        LOOP UNTIL EACH SECTION FULL
                1950 *--------------------------------
B85A- B1 3E     1960        LDA (BUF.PNTR),Y  GET LAST REAL BYTE
B85C- AA        1970        TAX
B85D- 29 07     1980        AND #7        USE LOWER 3 BITS
B85F- 8D 99 BC  1990        STA RWTS.BUFFER.2.4
B862- 8A        2000        TXA           NOW GET 5 UPPER BITS
B863- 4A        2010        LSR
B864- 4A        2020        LSR
B865- 4A        2030        LSR
B866- 8D FF BB  2040        STA RWTS.BUFFER.1.6
B869- 60        2050        RTS
```

```
                2060 *
                2070 *--------------------------------
                2080 *       WRITE A SECTOR ON THE DISK FROM RWTS.BUFFER
                2090 *--------------------------------
                2100 WRITE.SECTOR
B86A- 38        2110         SEC              SET IN CASE OF ERROR RETURN
B86B- BD 8D C0  2120         LDA Q6H,X        Q6 HIGH, Q7 LOW,
B86E- BD 8E C0  2130         LDA Q7L,X        TO READ WRITE PROTECT STATUS
B871- 30 7C     2140         BMI .5           DISK IS WRITE PROTECTED
B873- 86 27     2150         STX $27          SAVE SLOT #
B875- 8E 78 06  2160         STX $0678        HERE, TOO
B878- AD 00 BC  2170         LDA RWTS.BUFFER.2.1  FIRST NYBBLE OF DATA
B87B- 85 26     2180         STA $26          SAVE IT
B87D- A9 FF     2190         LDA #$FF         SYNC BYTE
B87F- 9D 8F C0  2200         STA Q7H,X        Q6H,Q7H: (A) TO SHIFT REGISTER
B882- 1D 8C C0  2210         ORA Q6L,X        Q6L,Q7H: WRITE ON DISK
B885- 48        2220         PHA              TIME DELAYS
B886- 68        2230         PLA
B887- EA        2240         NOP
B888- A0 0A     2250         LDY #10          WRITE TEN MORE SYNC BYTES
B88A- 05 26     2260 .1      ORA $26          WASTE TIME
B88C- 20 F4 B8  2270         JSR WRT2         WRITE (A) ON DISK
B88F- 88        2280         DEY
B890- D0 F8     2290         BNE .1           UNTIL 10 OF THEM
B892- A9 D5     2300         LDA #$D5         WRITE DATA HEADER
B894- 20 F3 B8  2310         JSR WRT1
B897- A9 AA     2320         LDA #$AA
B899- 20 F3 B8  2330         JSR WRT1
B89C- A9 AD     2340         LDA #$AD
B89E- 20 F3 B8  2350         JSR WRT1
B8A1- 98        2360         TYA              A=0
B8A2- A0 9A     2370         LDY #154         WRITE 154 NYBBLES
B8A4- D0 03     2380         BNE .3           ...ALWAYS
B8A6- B9 00 BC  2390 .2      LDA RWTS.BUFFER.2.1,Y  GET CURRENT NYBBLE AND
B8A9- 59 FF BB  2400 .3      EOR RWTS.BUFFER.2.1-1,Y  EOR WITH PREVIOUS NYBBLE
B8AC- AA        2410         TAX              USE AS OFFSET INTO TABLE
B8AD- BD 9A BC  2420         LDA NYBBLE.TABLE,X  MAP 5-BITS TO 8-BITS
B8B0- A6 27     2430         LDX $27          GET SLOT AGAIN
B8B2- 9D 8D C0  2440         STA Q6H,X        Q6H,Q7H: (A) TO SHIFT REGISTER
B8B5- BD 8C C0  2450         LDA Q6L,X        Q6L,Q7H: WRITE ON DISK
B8B8- 88        2460         DEY
B8B9- D0 EB     2470         BNE .2           UNTIL ALL BYTES FROM THIS BLOCK DONE
B8BB- A5 26     2480         LDA $26          GET FIRST NYBBLE
B8BD- EA        2490         NOP
B8BE- 59 00 BB  2500 .4      EOR RWTS.BUFFER.1.1,Y  EOR WITH CURRENT NYBBLE
B8C1- AA        2510         TAX              INDEX INTO TABLE
B8C2- BD 9A BC  2520         LDA NYBBLE.TABLE,X  MAP TO 8-BIT VALUE
B8C5- AE 78 06  2530         LDX $0678        SLOT # AGAIN
B8C8- 9D 8D C0  2540         STA Q6H,X        Q6H,Q7L: (A) TO SHIFT REGISTER
B8CB- BD 8C C0  2550         LDA Q6L,X        Q6L,Q7H: WRITE ON DISK
B8CE- B9 00 BB  2560         LDA RWTS.BUFFER.1.1,Y  GET NYBBLE
B8D1- C8        2570         INY
B8D2- D0 EA     2580         BNE .4           MORE TO DO
B8D4- AA        2590         TAX              LAST NYBBLE
B8D5- BD 9A BC  2600         LDA NYBBLE.TABLE,X  MAP TO 8 BITS
B8D8- A6 27     2610         LDX $27          SLOT # AGAIN
B8DA- 20 F6 B8  2620         JSR WRT3         WRITE CHECK SUM ON DISK
B8DD- A9 DE     2630         LDA #$DE         WRITE TRAILER
B8DF- 20 F3 B8  2640         JSR WRT1
B8E2- A9 AA     2650         LDA #$AA
B8E4- 20 F3 B8  2660         JSR WRT1
B8E7- A9 EB     2670         LDA #$EB
B8E9- 20 F3 B8  2680         JSR WRT1
B8EC- BD 8E C0  2690         LDA Q7L,X        Q7L
B8EF- BD 8C C0  2700 .5      LDA Q6L,X        Q6L
B8F2- 60        2710         RTS
                2720 *--------------------------------
B8F3- 18        2730 WRT1    CLC              WAIT 2 CYCLES
B8F4- 48        2740 WRT2    PHA              WAIT 3 CYCLES
B8F5- 68        2750         PLA              WAIT 4 CYCLES
B8F6- 9D 8D C0  2760 WRT3    STA Q6H,X        Q6H,Q7H: (A) TO SHIFT REGISTER
B8F9- 1D 8C C0  2770         ORA Q6L,X        Q6L,Q7H: WRITE ON DISK
B8FC- 60        2780         RTS
```

```
                2790 *
                2800 *--------------------------------
                2810 *       READ SECTOR INTO RWTS.BUFFER
                2820 *--------------------------------
                2830 READ.SECTOR
B8FD- A0 20     2840         LDY #32       MUST FIND $D5 WITHIN 32 BYTES
B8FF- 88        2850 .1      DEY
B900- F0 61     2860         BEQ ERROR.RETURN
B902- BD 8C C0  2870 .2      LDA Q6L,X     READ SHIFT REGISTER
B905- 10 FB     2880         BPL .2        WAIT FOR FULL BYTE
B907- 49 D5     2890 .3      EOR #$D5      SEE IF FOUND $D5
B909- D0 F4     2900         BNE .1        NOT YET
B90B- EA        2910         NOP           DELAY BEFORE NEXT READ
B90C- BD 8C C0  2920 .4      LDA Q6L,X     READ SHIFT REGISTER
B90F- 10 FB     2930         BPL .4        WAIT FOR FULL BYTE
B911- C9 AA     2940         CMP #$AA      SEE IF $AA
B913- D0 F2     2950         BNE .3        NO
B915- A0 9A     2960         LDY #154      BYTE COUNT FOR LATER
B917- BD 8C C0  2970 .5      LDA Q6L,X     READ SHIFT REGISTER
B91A- 10 FB     2980         BPL .5        WAIT FOR FULL BYTE
B91C- C9 AD     2990         CMP #$AD      IS IT $AD?
B91E- D0 E7     3000         BNE .3        NO
                3010 *--------------------------------
B920- A9 00     3020         LDA #0        BEGIN CHECKSUM
B922- 88        3030 .6      DEY
B923- 84 26     3040         STY $26
B925- BC 8C C0  3050 .7      LDY Q6L,X     READ SHIFT REGISTER
B928- 10 FB     3060         BPL .7        WAIT FOR FULL BYTE
B92A- 59 00 BA  3070         EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
B92D- A4 26     3080         LDY $26       BUFFER INDEX
B92F- 99 00 BC  3090         STA RWTS.BUFFER.2.1,Y
B932- D0 EE     3100         BNE .6
B934- 84 26     3110 .8      STY $26
B936- BC 8C C0  3120 .9      LDY Q6L,X     READ SHIFT REGISTER
B939- 10 FB     3130         BPL .9        WAIT FOR FULL BYTE
B93B- 59 00 BA  3140         EOR BYTE.TABLE,Y  CONVERT TO NYBBLE
B93E- A4 26     3150         LDY $26
B940- 99 00 BB  3160         STA RWTS.BUFFER.1.1,Y
B943- C8        3170         INY
B944- D0 EE     3180         BNE .8
B946- BC 8C C0  3190 .10     LDY Q6L,X     READ CHECKSUM
B949- 10 FB     3200         BPL .10
B94B- D9 00 BA  3210         CMP BYTE.TABLE,Y
B94E- D0 13     3220         BNE ERROR.RETURN
B950- BD 8C C0  3230 .11     LDA Q6L,X     READ TRAILER
B953- 10 FB     3240         BPL .11
B955- C9 DE     3250         CMP #$DE
B957- D0 0A     3260         BNE ERROR.RETURN
B959- EA        3270         NOP
B95A- BD 8C C0  3280 .12     LDA Q6L,X
B95D- 10 FB     3290         BPL .12
B95F- C9 AA     3300         CMP #$AA
B961- F0 5C     3310         BEQ GOOD.RETURN
                3320 ERROR.RETURN
B963- 38        3330         SEC
B964- 60        3340         RTS
```

```
                3350 *
                3360 *--------------------------------
                3370 *       READ ADDRESS
                3380 *--------------------------------
                3390 READ.ADDRESS
B965- A0 F8     3400         LDY #$F8      TRY 1800 TIMES (FROM $F8F8 TO $10000)
B967- 84 26     3410         STY $26
B969- C8        3420 .1      INY
B96A- D0 04     3430         BNE .2
B96C- E6 26     3440         INC $26
B96E- F0 F3     3450         BEQ ERROR.RETURN
B970- BD 8C C0  3460 .2      LDA Q6L,X     READ SHIFT REGISTER
B973- 10 FB     3470         BPL .2        WAIT FOR FULL BYTE
B975- C9 D5     3480 .3      CMP #$D5      SEE IF $D5
B977- D0 F0     3490         BNE .1        NO
B979- EA        3500         NOP           DELAY
B97A- BD 8C C0  3510 .4      LDA Q6L,X     READ SHIFT REGISTER
B97D- 10 FB     3520         BPL .4        WAIT FOR FULL BYTE
B97F- C9 AA     3530         CMP #$AA      SEE IF $AA
B981- D0 F2     3540         BNE .3        NO
B983- A0 03     3550         LDY #3        READ 3 BYTES LATER
B985- BD 8C C0  3560 .5      LDA Q6L,X     READ SHIFT REGISTER
B988- 10 FB     3570         BPL .5
B98A- C9 B5     3580         CMP #$B5      SEE IF $B5
B98C- D0 E7     3590         BNE .3        NO
B98E- A9 00     3600         LDA #0        START CHECK SUM
B990- 85 27     3610 .6      STA $27
B992- BD 8C C0  3620 .7      LDA Q6L,X     READ REGISTER
B995- 10 FB     3630         BPL .7
B997- 2A        3640         ROL
B998- 85 26     3650         STA $26
B99A- BD 8C C0  3660 .8      LDA Q6L,X     READ REGISTER
B99D- 10 FB     3670         BPL .8        WAIT FOR FULL BYTE
B99F- 25 26     3680         AND $26       MERGE THE NYBBLES
B9A1- 99 2C 00  3690         STA $2C,Y     $2C -- CHECK SUM
B9A4- 45 27     3700         EOR $27       $2D -- SECTOR
B9A6- 88        3710         DEY           $2E -- TRACK
B9A7- 10 E7     3720         BPL .6        $2F -- VOLUME
B9A9- A8        3730         TAY           TEST CHECK SUM
B9AA- D0 B7     3740         BNE ERROR.RETURN
B9AC- BD 8C C0  3750 .9      LDA Q6L,X     READ REGISTER
B9AF- 10 FB     3760         BPL .9        WAIT FOR FULL BYTE
B9B1- C9 DE     3770         CMP #$DE      TEST FOR VALID TRAILER
B9B3- D0 AE     3780         BNE ERROR.RETURN
B9B5- EA        3790         NOP
B9B6- BD 8C C0  3800 .10     LDA Q6L,X     READ REGISTER
B9B9- 10 FB     3810         BPL .10
B9BB- C9 AA     3820         CMP #$AA
B9BD- D0 A4     3830         BNE ERROR.RETURN
                3840 GOOD.RETURN
B9BF- 18        3850         CLC
B9C0- 60        3860         RTS
```

```
                3870 *
                3880 *-----------------------------------
                3890 *      CONVERT 410 5-BIT NYBBLES TO 256 BYTES
                3900 *      (THEY ARE NOW LEFT-JUSTIFIED IN RWTS.BUFFER)
                3910 *-----------------------------------
                3920 POST.NYBBLE
B9C1- A2 32     3930        LDX #50       51 BYTES PER SECTION
B9C3- A0 00     3940        LDY #0
                3950 *--BUFFER PART 1, SECTION 1-----
B9C5- BD 00 BC  3960 .1     LDA RWTS.BUFFER.2.1,X
B9C8- 4A        3970        LSR
B9C9- 4A        3980        LSR           RIGHT-JUSTIFY THE NYBBLE
B9CA- 4A        3990        LSR
B9CB- 85 27     4000        STA $27       SAVE BIT 0
B9CD- 4A        4010        LSR
B9CE- 85 26     4020        STA $26       SAVE BIT 1
B9D0- 4A        4030        LSR           BITS 2-4
B9D1- 1D 00 BB  4040        ORA RWTS.BUFFER.1.1,X
B9D4- 91 3E     4050        STA (BUF.PNTR),Y  STORE IN BUFFER
                4060 *--BUFFER PART 1, SECTION 2-----
B9D6- C8        4070        INY           NEXT BYTE
B9D7- BD 33 BC  4080        LDA RWTS.BUFFER.2.2,X
B9DA- 4A        4090        LSR           RIGHT-JUSTIFY THE NYBBLE
B9DB- 4A        4100        LSR
B9DC- 4A        4110        LSR
B9DD- 4A        4120        LSR           BIT 0 INTO CARRY
B9DE- 26 27     4130        ROL $27       AND SAVE HERE
B9E0- 4A        4140        LSR           BIT 1 INTO CARRY
B9E1- 26 26     4150        ROL $26       AND SAVE HERE
B9E3- 1D 33 BB  4160        ORA RWTS.BUFFER.1.2,X
B9E6- 91 3E     4170        STA (BUF.PNTR),Y  STORE THE BYTE
                4180 *--BUFFER PART 1, SECTION 3-----
B9E8- C8        4190        INY           NEXT BYTE
B9E9- BD 66 BC  4200        LDA RWTS.BUFFER.2.3,X
B9EC- 4A        4210        LSR           RIGHT-JUSTIFY THE NYBBLE
B9ED- 4A        4220        LSR
B9EE- 4A        4230        LSR
B9EF- 4A        4240        LSR           BIT 0 INTO CARRY
B9F0- 26 27     4250        ROL $27       AND SAVE HERE
B9F2- 4A        4260        LSR           BIT 1 INTO CARRY
B9F3- 26 26     4270        ROL $26       AND SAVE HERE
B9F5- 1D 66 BB  4280        ORA RWTS.BUFFER.1.3,X
B9F8- 91 3E     4290        STA (BUF.PNTR),Y  STORE THE BYTE
                4300 *--BUFFER PART1, SECTION 4-----
B9FA- C8        4310        INY           NEXT BYTE
B9FB- A5 26     4320        LDA $26       USE THE 3 BITS SAVED HERE
B9FD- 29 07     4330        AND #7        MAKE SURE ONLY 3 BITS
B9FF- 1D 99 BB  4340        ORA RWTS.BUFFER.1.4,X
BA02- 91 3E     4350        STA (BUF.PNTR),Y  STORE THE BYTE
                4360 *--BUFFER PART1, SECTION 5-----
BA04- C8        4370        INY           NEXT BYTE
BA05- A5 27     4380        LDA $27       USE THE 3 BITS SAVED HERE
BA07- 29 07     4390        AND #7        MAKE SURE ONLY 3 BITS
BA09- 1D CC BB  4400        ORA RWTS.BUFFER.1.5,X
BA0C- 91 3E     4410        STA (BUF.PNTR),Y  STORE THE BYTE
                4420 *-----------------------------------
BA0E- C8        4430        INY           NEXT BYTE
BA0F- CA        4440        DEX
BA10- 10 B3     4450        BPL .1
                4460 *-----------------------------------
BA12- AD 99 BC  4470        LDA RWTS.BUFFER.2.4  GET THE LAST BYTE
BA15- 4A        4480        LSR           RIGHT JUSTIFY
BA16- 4A        4490        LSR
BA17- 4A        4500        LSR
BA18- 0D FF BB  4510        ORA RWTS.BUFFER.1.6
BA1B- 91 3E     4520        STA (BUF.PNTR),Y  STORE THE LAST BYTE
BA1D- 60        4530        RTS
```

```
                  4540 *
                  4550 *--------------------------------
                  4560 *       TRACK SEEK
                  4570 *--------------------------------
                  4580 SEEK.TRACK.ABSOLUTE
BA1E- 86 2B       4590         STX $2B       CURRENT SLOT*16
BA20- 85 2A       4600         STA $2A       SAVE TRACK #
BA22- CD 78 04    4610         CMP CURRENT.TRACK  COMPARE TO CURRENT TRACK
BA25- F0 53       4620         BEQ .9        ALREADY THERE
BA27- A9 00       4630         LDA #0
BA29- 85 26       4640         STA $26       # OF STEPS SO FAR
BA2B- AD 78 04    4650 .1      LDA CURRENT.TRACK  CURRENT TRACK NUMBER
BA2E- 85 27       4660         STA $27
BA30- 38          4670         SEC
BA31- E5 2A       4680         SBC $2A       DESIRED TRACK
BA33- F0 33       4690         BEQ .6        WE HAVE ARRIVED
BA35- B0 07       4700         BCS .2        CURRENT > DESIRED
BA37- 49 FF       4710         EOR #$FF      CURRENT < DESIRED
BA39- EE 78 04    4720         INC CURRENT.TRACK  INCREMENT CURRENT
BA3C- 90 05       4730         BCC .3        ...ALWAYS
BA3E- 69 FE       4740 .2      ADC #$FE      CARRY SET, SO A=A-1
BA40- CE 78 04    4750         DEC CURRENT.TRACK  DECREMENT CURRENT TRACK
BA43- C5 26       4760 .3      CMP $26       GET MINIMUM OF:
BA45- 90 02       4770         BCC .4           1. # OF TRACKS TO MOVE LESS 1
BA47- A5 26       4780         LDA $26          2. # OF ITERATIONS SO FAR
BA49- C9 0C       4790 .4      CMP #12          3. ELEVEN
BA4B- B0 01       4800         BCS .5
BA4D- A8          4810         TAY
BA4E- 38          4820 .5      SEC           TURN PHASE ON
BA4F- 20 6C BA    4830         JSR .7
BA52- B9 8C BA    4840         LDA ONTBL,Y   GET DELAY TIME
BA55- 20 7B BA    4850         JSR DLY100    DELAY 100*A MICROSECONDS
BA58- A5 27       4860         LDA $27       TRACK NUMBER
BA5A- 18          4870         CLC           TURN PHASE OFF
BA5B- 20 6F BA    4880         JSR .8
BA5E- B9 98 BA    4890         LDA OFFTBL,Y
BA61- 20 7B BA    4900         JSR DLY100
BA64- E6 26       4910         INC $26       # OF STEPS SO FAR
BA66- D0 C3       4920         BNE .1        ...ALWAYS
                  4930 *--------------------------------
BA68- 20 7B BA    4940 .6      JSR DLY100
BA6B- 18          4950         CLC           TURN PHASE OFF
BA6C- AD 78 04    4960 .7      LDA CURRENT.TRACK
BA6F- 29 03       4970 .8      AND #3        ONLY KEEP LOW-ORDER 2 BITS
BA71- 2A          4980         ROL           (0000 0XX0)
BA72- 05 2B       4990         ORA $2B       (0SSS 0XX0) MERGE SLOT
BA74- AA          5000         TAX           USE AS INDEX FOR PHASE-OFF
BA75- BD 80 C0    5010         LDA PHOFF,X   PHASE-OFF
BA78- A6 2B       5020         LDX $2B
BA7A- 60          5030 .9      RTS
                  5040 *--------------------------------
                  5050 *       SHORT DELAY SUBROUTINE
                  5060 *--------------------------------
BA7B- A2 11       5070 DLY100  LDX #17       100*A MICROSECONDS
BA7D- CA          5080 .1      DEX
BA7E- D0 FD       5090         BNE .1
BA80- E6 46       5100         INC $46
BA82- D0 02       5110         BNE .2
BA84- E6 47       5120         INC $47
BA86- 38          5130 .2      SEC
BA87- E9 01       5140         SBC #1
BA89- D0 F0       5150         BNE DLY100
BA8B- 60          5160         RTS
                  5170 *--------------------------------
                  5180 *       DELAY TIMES FOR STEPPING MOTOR
                  5190 *--------------------------------
BA8C- 01 30 28
BA8F- 24 20 1E
BA92- 1D 1C 1C
BA95- 1C 1C 1C    5200 ONTBL   .HS 01302824201E1D1C1C1C1C1C
BA98- 70 2C 26
BA9B- 22 1F 1E
BA9E- 1D 1C 1C
BAA1- 1C 1C 1C    5210 OFFTBL  .HS 702C26221F1E1D1C1C1C1C1C
BAA4- 1C 1C 1C
BAA7- 1C          5220         .HS 1C1C1C1C
```

```
                5230 *
                5240 *-----------------------------
                5250 *      BYTE TABLE
                5260 *-----------------------------
BA00-           5270 BYTE.TABLE .EQ *-$A8
BAA8- 00 00 00
BAAB- 00 01 08
BAAE- 10 18 02
BAB1- 03 04 05
BAB4- 06 20 28
BAB7- 30        5280        .HS 00000000010810180203040506202830
BAB8- 07 09 38
BABB- 40 0A 48
BABE- 50 58 0B
BAC1- 0C 0D 0E
BAC4- 0F 11 12
BAC7- 13        5290        .HS 070938400A4850580B0C0D0E0F111213
BAC8- 14 15 16
BACB- 17 19 1A
BACE- 1B 1C 1D
BAD1- 1E 21 22
BAD4- 23 24 60
BAD7- 68        5300        .HS 14151617191A1B1C1D1E212223246068
BAD8- 25 26 70
BADB- 78 27 80
BADE- 88 90 29
BAE1- 2A 2B 2C
BAE4- 2D 2E 2F
BAE7- 31        5310        .HS 252670782780889029 2A2B2C2D2E2F31
BAE8- 32 33 98
BAEB- A0 34 A8
BAEE- B0 B8 35
BAF1- 36 37 39
BAF4- 3A C0 C8
BAF7- D0        5320        .HS 323398A034A8B0B8353637393AC0C8D0
BAF8- 3B 3C D8
BAFB- E0 3E E8
BAFE- F0 F8     5330        .HS 3B3CD8E03EE8F0F8
                5340 *-----------------------------
                5350 *      410-BYTE BUFFER FOR NYBBLES
                5360 *-----------------------------
BB00-           5370 RWTS.BUFFER.1.1 .BS 51  $BB00 - BB32
BB33-           5380 RWTS.BUFFER.1.2 .BS 51  $BB33 - BB65
BB66-           5390 RWTS.BUFFER.1.3 .BS 51  $BB66 - BB98
BB99-           5400 RWTS.BUFFER.1.4 .BS 51  $BB99 - BBCB
BBCC-           5410 RWTS.BUFFER.1.5 .BS 51  $BBCC - BBFE
BBFF-           5420 RWTS.BUFFER.1.6 .BS 1   $BBFF
BC00-           5430 RWTS.BUFFER.2.1 .BS 51  $BC00 - BC32
BC33-           5440 RWTS.BUFFER.2.2 .BS 51  $BC33 - BC65
BC66-           5450 RWTS.BUFFER.2.3 .BS 51  $BC66 -BC98
BC99-           5460 RWTS.BUFFER.2.4 .BS 1   $BC99
                5470 *-----------------------------
                5480 *      NYBBLE TABLE
                5490 *-----------------------------
                5500 NYBBLE.TABLE
BC9A- AB AD AE
BC9D- AF B5 B6
BCA0- B7 BA BB
BCA3- BD BE BF  5510        .HS ABADAEAFB5B6B7BABBBDBEBF
BCA6- D6 D7 DA
BCA9- DB DD DE
BCAC- DF EA EB
BCAF- ED EE EF  5520        .HS D6D7DADBDDDEDFEAEBEDEEEF
BCB2- F5 F6 F7
BCB5- FA FB FD
BCB8- FE FF     5530        .HS F5F6F7FAFBFDFEFF
                5540 *-----------------------------
                5550 *      $BCBA THRU $BCFF IS NOT USED BY DOS 3.2.1
                5560 *-----------------------------
                5570 *
```

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).