DOS 3.3

DOUBLE HI-RES GRAPHICS V

The Graphics Workshop continues its exploration of Double Hi-Res animation by adding two routines that allow horizontal shifting to the Double Hi-Res driver. A technique for using pre-shifted animation on the DHR screen is also presented.

by Robert R. Devine
Computers and You
Mellor Park Mall
1855 North West Ave.
El Dorado, AR 71730

In the last installment of "Double Hi-Res Graphics," we developed the DHR Palette program, and in previous articles we developed some drawing routines to animate shapes on the Double Hi-Res screen.

In Part V we deal strictly with horizontal animation. We'll develop some new routines, and look at a few different animation methods. One routine is best suited to black and white shapes, and the other is best suited to shapes that contain some of the sixteen colors that are now available.

You may recall that our last attempt at moving shapes sideways was less than ideal, since we needed to move the shapes a full fourteen dots (one screen address) each time. The routines in Part V will allow you to move your shapes sideways, one horizontal dot each move.

Double Hi-Res Shift Animation

To minimize typing, we'll add just two new routines to the driver this month. The names of these new routines are SHIFTR (CALL 37444) and SHIFTL (CALL 37374). SHIFTR will move your shape one dot to the right, and SHIFTL will move your shape one dot to the left. The parameters that you'll need to set for these routines are VT, VB, HR and HL.

These routines do not make use of Shape Tables, so it is not necessary to specify SHNUM. SHIFTR and SHIFTL can be used to animate (move) any graphics that are *presently on the screen* — shapes, parts of shapes, or even background graphics.

These shifting routines have both advantages and disadvantages. On the plus side, they will create the smoothest horizontal movement possible, without the slightest flicker. No erasing of any kind will ever be required as the routines are self-erasing. Also, you will be able to move anything that is present on the screen, even if Shape Tables do not exist for the parts that you wish to move.

On the minus side, the routines are only practical for black and white shapes. A colored shape will change color with each shift as the bits and color blocks change alignment. (We will look at how to move colored shapes later in this article.) And, while very smooth, the shift routines are slower than those that move a shape of the same size on the regular Hi-Res screen. This is because:

1. A Double Hi-Res shape has twice as many data bytes that need to be processed for each shift as a comparably sized regular Hi-Res shape.
2. Since a Double Hi-Res dot is only half as wide, your shape will only move half as far each shift, thus requiring twice as many shifts to go the same distance.
3. The routines need to take care of soft-switch flipping to put the drawing in the proper bank of memory, which is not required in regular Hi-Res shift animation.

Entering the Shift Routines

At this point, load your old DHR driver routines into memory, then add the new SHIFTR and SHIFTL routines (**Listings 1 and 2**). Once they're in memory, you should have the DHR driver shown in **Listing 3**, DHR.DRIVER $91FE. Save this program on disk with the command:

```
BSAVE DHR.DRIVER $91FE,A$91FE,
      L$402
```

(For help in entering *Nibble* listings, see "A Welcome to New *Nibble* Readers" in the beginning of this issue.)

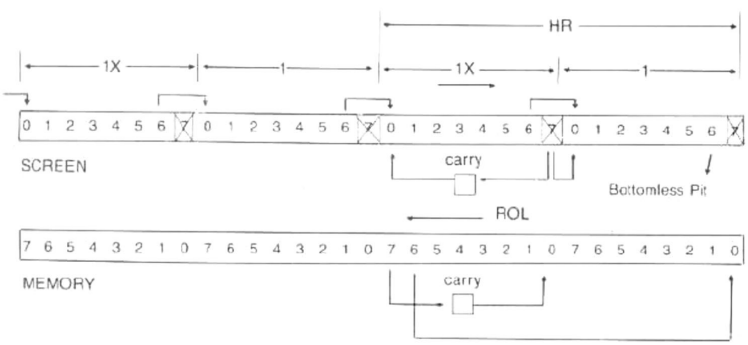
How the Shift Routines Work

Since the new routines are heavily documented in the listings, we won't go over them in detail. However, it would probably be a good idea to see just how Double Hi-Res shift animation works.

Shifting Shapes to the Right

Let us start out with SHIFTR (refer to **Figure 1**). When using SHIFTR you must always add one to the value of HR, which will mean that there is one additional address

Figure 1: Shift Right



(fourteen dots) to the right (ahead) of your shape. This provides additional non-shape bits in front of the shape into which the shape bits can be shifted.

After you have shifted the shape forward (to the right) fourteen times, the extra fourteen dots (one address) will be behind the shape. At that time you should use the MOVERT routine (CALL 37548), which will INCrement HR and HL to replace the fourteen shifting dots ahead of the shape in preparation for the next fourteen shifts forward.

"...a colored shape will change color with each shift as the bits and color blocks change alignment."

The key to shifting shapes rightward is the ROL (ROtate Left) statement in machine code. If that seems confusing, just remember that Hi-Res screen bytes are displayed in reverse of the way they appear in memory. If you look at **Figure 1**, you will see the bits as they appear on the screen and as they appear in memory. To move the screen dots rightward, you actually shift the bits leftward in memory.

SHIFTR processes the data bytes in the same order as REVDIR, entering the block at VB/HL (vertical bottom/horizontal left) and ending at VT/HR (vertical top/horizontal right). Thus, we *push* the shape rather than *pull* it forward. The first byte processed in each line is at HL/page 1X and the last byte processed is at HR/page 1.

First the Carry (a special bit in the 6502's Status Register) is conditioned to 0 or 1 based on the status of the B0FLAG (bit zero flag). This is really the pre-shifted status of bit 6 of the next byte to the left. Next, the byte is ROLed (ROtated Left), which shifts the contents of bits 0-6 into bits 1-7, and shifts the Carry into bit 0.

Now the rotated byte is placed on the screen to replace the existing byte on page 1X. Next, the byte in memory (in the Accumulator, not on the screen) is ROLed again, pushing bit 7 (which is the original bit 6 before the shift) into the Carry.

We then move to the next byte to the right (page 1) and ROL it, again moving the contents of bits 0-6 into bits 1-7 and setting bit 0 (from the Carry) to the status of bit 6 of the last byte that we shifted. The shifted byte is then replaced on the screen on page 1.

Finally, we again ROL bit 7 into the Carry to find the status of bit 6 before the shift, and set the B0FLAG before moving on to the next address to see if we need to shift the two bytes at that address.

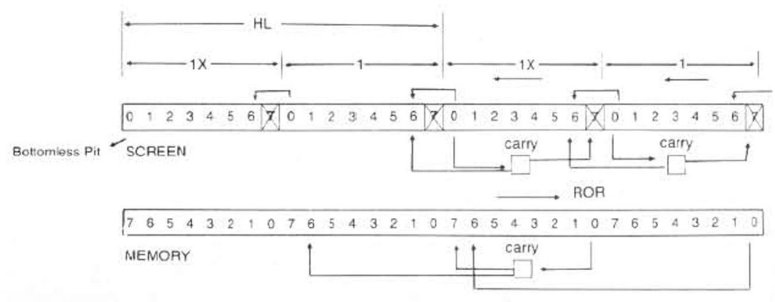
You will notice that when we begin each new line, we begin with our B0FLAG set to 0, which automatically sets the first 1 bit it encounters to 0, thus taking care of the erase as the shape moves forward. You will also notice that whatever the status of bit 6 as it is shifted out of the HR/page 1 byte, that value is dropped and does not carry forward to the next byte. If you ever watch your shape slowly disappearing from the screen because you shifted too many times (more than fourteen) without INCrementing HR and HL to add shifting bytes ahead, this is where your shape is going. The bit being shifted out of bit 6 HR/page 1 simply drops off and is lost forever.

Shifting Shapes to the Left

Now let us look at SHIFTL (refer to **Figure 2**). When using SHIFTL you must always subtract one from the value of HL, which means that there are always fourteen shifting dots to the left (ahead) of your shape. As with SHIFTR, this provides additional non-shape bits in front of the shape into which we can shift the shape bits. After you have shifted fourteen times to the left, the shifting bits will be behind the shape. You'll need to use the MOVELF routine (CALL 37559) which will DECrement HR and HL, replacing the shifting bits ahead of the shape in readiness for the next fourteen shifts.

The key to shifting leftward is the ROR (ROtate Right) statement in machine code. To move the screen dots of your shape leftward, you must shift the bits in memory rightward.

Figure 2: Shift Left



SHIFTL processes the data bytes in the same order as SCAN and DRAW, beginning at VB/HR and finishing at VT/HL. The first byte processed in each line is at HR/page 1 and the last byte is at HL/page 1X.

First the B6FLAG (bit 6 flag) is tested to determine if the pre-shifted status of bit 0 of the next byte to the right was 0 or 1. If it was 1, then bit 7 is set to 1. Next, the Carry is set to 0, after which the byte is RORed. This moves the contents of bits 1-7 into bits 0-6, while the Carry (0) is moved into bit 7. We always leave bit 7 set to 0 after every shift.

Now the rotated byte is placed on the screen to replace the existing byte on page 1.

Next we switch to the byte on page 1X and test the Carry to determine the pre-shifted status of bit 0 of the page 1 byte that we just finished. This tells us if bit 6 of this byte needs to be set to 0 or 1. If it should be 1, then we again set bit 7 to 1. As always, the Carry is set to 0 and the byte is RORed by moving the contents of bits 1-7 into bits 0-6, with the 0 from the Carry setting bit 7 to 0. During the ROR, bit 0 falls into the Carry.

This rotated byte is then placed on the screen to replace the present byte on page 1X.

Finally, we test the Carry to see if the pre-shifted value of bit 0 was 0 or 1, and set the B6FLAG appropriately for use in the next address. Here we begin each line with the B6FLAG set to 0. This automatically sets the first 1 bit it encounters to 0, which takes care of our erasing needs.

In the SHIFTL routine, whatever value shifts out of bit 0 HL/page 1X drops off and is disregarded. Therefore, if you shift more than fourteen times without DECrementing HR and HL to add shifting bits, your shape will begin to disappear from the screen.

Testing the Shift Routines

We've examined the mechanics of shift animation in depth because the concepts behind it can be difficult and confusing. Now let's try our first test of shift animation on the Double Hi-Res screen. If you worked through the shift routines for normal Hi-Res from the Graphics Workshop series, you'll remember how easy they were to use. You will be glad to know that Double Hi-Res shift animation is just as easy. All the idiosyncrasies of the Double Hi-Res screen are handled by the driver routines.

To try out the first test you should enter the program shown in **Listing 4**. Save it on disk with the command:

SAVE SHIFT.TEST1

You will also need to have the DHR driver and our spaceship shape, SHAPE#144 (**Listing 5**), on the same disk. If you do not already have this shape file on disk, use the Monitor to enter the code and save it with the command:

BSAVE SHAPE#144,A$9000,L$54

When you run SHIFT.TEST1, you'll see your shape move smoothly back and forth across the screen. Let's see how it works.

You should be familiar with **lines 80-120**, which load the driver and the shape, then initialize full-screen Double Hi-Res graphics.

Line 130 draws the shape on the screen. This is the last time that the shape will be DRAWn in the program. Now that the shape is on the screen, the shifting routines take over.

Line 140 removes the extra erasing lines above and below the shape. We added these when the shape was originally created. There is no sense shifting unnecessary bytes.

Line 150 adds an extra address (14 dots) ahead of the shape.

Line 160 moves the shape from the left side to the right side of the screen. The **FOR HR=3 TO 39** simply provides a movement loop and shows the different values that HR will have as the shape moves. First, we shift the shape rightward fourteen times using SHIFTR (CALL 37444), then we INCrement HR and HL using MOVERT (CALL 37548) before jumping back to shift another fourteen times.

Line 170 moves the shape from the right side back to its starting point on the left side of the screen. It was not necessary for us to add shifting bits ahead of the shape because once we reached the right side and changed direction, the extra fourteen bits that were behind the shape at the end of our last rightward movement were ahead of the shape after we changed direction. (What this really means is that the last CALL to the MOVERT routine failed to execute because MOVERT won't allow HR to be INCremented past 39.) To move leftward we simply CALLed the SHIFTL routine (CALL 37374) fourteen times, then used MOVELF (CALL 37559) to DECrement HR and HL.

Line 180 jumps back to **line 160** to start moving to the right again.

The Complexities of Shift Animation

I could write reams delving into the complexities of shift animation to impress you with my knowledge, but if you understand SHIFT.TEST1, you already know the whole story!

Colored Shapes and Horizontal Movement

To move colored shapes horizontally, you must move four dots per move. The best way to do this is to use a series of shapes, with each shape shifted four dots from the last one. As you draw the series of shapes, one atop the other, *at the same HR/HL*, the shape will appear to move forward. This type of animation is referred to as pre-shifted animation. When working with pre-shifted shapes on the Double Hi-Res screen, you will normally use a series of seven shapes — each shape shifted four dots from the next one, in a block shape that is two addresses (28 dots) wider than the actual size of the shape.

When using pre-shifted shapes you will always want to use the EOROFF routine (CALL 37517) so that the previous shape in the series is properly erased when the next shape is drawn. Another nice thing about pre-shifted shapes is that by slightly varying the shapes in the series, you can achieve the effect of animation (e.g., a man walking) while the shape moves forward. If we wanted to, we could easily put blinking lights on our spaceship by adding that effect to three or four of the shapes in the series.

To demonstrate pre-shifted animation, we will need to create a series of eight pre-shifted shapes. **Figure 3** shows how each of the shapes in the series will be oriented within the block shape.

As you will recall, the spaceship shape is only three addresses (six bytes) wide; however, in our pre-shifted series, each shape will be five addresses (ten bytes) wide. The first shape in the series (shape 137) will have two empty addresses to the right of it, with the actual shape running from Double Hi-Res X-coordinates 0-39, while the last shape in the series (shape 144) will have two empty addresses to the left of it, with the actual shape residing in X-coordinates 28-67. The other shapes in the series will reside somewhere between these two extremes.

You can see from **Figure 3** that if you were to draw each shape in the series, one atop the next, the shape would appear to move to the right (drawing shapes 137-144) or to the left (drawing shapes 144-137). As each new shape in the series is drawn, the exposed parts of the last shape are automatically erased.

CREATE.PRE-SHIFTS (**Listing 6**) is a short program that does all the work of creating our series of eight pre-shifted shapes. (We'll see shortly why we need eight rather than seven shapes.)

Lines 80-130 perform exactly the same functions as our shift animation test in **Listing 4**, setting things up and DRAWing the shape on the screen. **Lines 150-160** translate the X-coordinate (0-559) into the proper HPLOT X-coordinate (0-279).

Line 170 simply draws a series of vertical black lines through the shape, which has the effect of changing the color from white to yellow.

Line 180 removes the extra lines from above and below the shape (which we put there when the shape was created) and changes the width of the shape from three addresses to five addresses.

Lines 190-220 create the eight pre-shifted shapes. First we **POKE SHNUM** into location 251, then we SCAN the shape into a Shape Table. Next we move the shape right four dots and jump back to set the next shape number, continuing until all eight shapes are created. Finally, CALL 37966 exits Double Hi-Res.

Line 230 saves the shape to disk.

That was easy, wasn't it?

You should note at this point that we're being rather wasteful with memory here. Each of the shapes in the series is only 120 bytes long, but we use an entire 256-byte memory page for each shape. In a normal programming environment you would want to pack the shapes together, one immediately following the other. To do this, enter two POKEs for each shape. First enter **POKE 251,SHNUM** (the high byte of the address where the shape begins), and then enter **POKE 37781** (the low byte of the address where the shape begins). This changes the first instruction in DRAW from LDA #0 to LDA (low byte). When using DRAWDN with packed shapes, the second POKE would be **POKE 37709** (low byte), and when using REVDIR it would be **POKE 37625** (low byte).

Now that we've prepared a series of pre-shifted shapes, let's try them out in a program.

SHIFT.TEST2 (**Listing 7**) does basically the same thing as SHIFT.TEST1 except that this time we move a colored shape across the screen using a series of pre-shifted shapes.

Lines 80-120 are again used to set up Double Hi-Res. **Line 130** turns off the EOR function of DRAW. This is always necessary when using pre-shifted shapes. **Line 140** sets up the location at which the shape will first appear on the screen.

Line 150 is a loop that indicates the values of HL as we move across the screen. Step 2 double increments HR and HL after every series of seven shapes.

Line 160 steps through the series of seven shapes and draws them in sequence, all at the current HR/HL. If you wonder why we use just shapes 138-144 instead of 137-144 (all eight shapes), refer to **Figure 3**. You'll note that shapes 137 and 144 are exactly the same except that they are shifted exactly two addresses apart. If we ran the series from 137 to 144, this is what would happen: after drawing shape 144 and incrementing HR and HL twice, when we drew shape 137 at the new HR/HL, we would be drawing at the location of the old shape 144. The only way to keep the shape constantly moving forward is to DRAW shape 138 following shape 144. The same reasoning applies to the use of shapes 143-137 when moving the shape leftward.

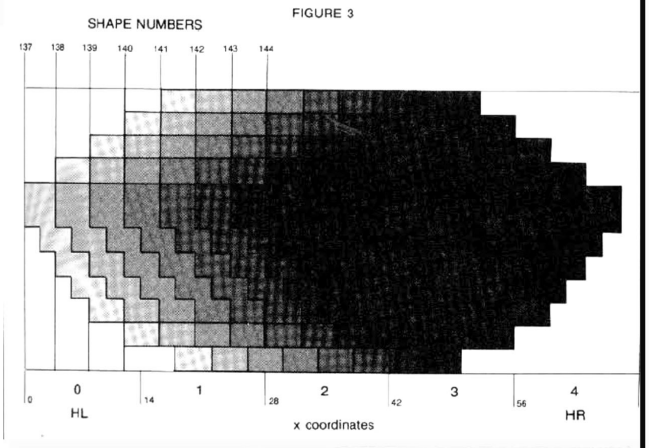
Line 170 double increments HR and HL by CALLing MOVERT twice. **Line 190** decrements HR and HL by one in preparation for the return trip.

Lines 200-230 move the shape back to the left side using the same methods that we used in moving rightward, but this time we step through shapes 143-137, and double decrement HR and HL to achieve leftward movement.

This animation method is almost as smooth as our shift routines although we're moving four dots each time. Since all of our animation activity is directed at this one shape, you may catch some strobing effects as the shape is DRAWn over and over again.

Now you have at your disposal two different methods of horizontal animation for Double Hi-Res. Next month we'll finish the Double Hi-Res driver by adding some routines that will allow you to do vertical shift animation, totally eliminating the need to use DRAW or DRAWDN for many types of animation. See you then!

Figure 3: Pre-shifted Shapes



LISTING 1: SHIFTR

```
                1000   .OR $9244        ** DHR-SHIFTR
                1010   .TA $800         ** BY ROBERT DEVINE
                1015 ** COPYRIGHT 1984 BY MICROSPARC, INC.
00FC-           1020 YT .EQ $FC         ** DECIMAL 252
00FD-           1030 YB .EQ $FD         ** DECIMAL 253
00FE-           1040 HR .EQ $FE         ** DECIMAL 254
00FF-           1050 HL .EQ $FF         ** DECIMAL 255
0026-           1060 HBASL .EQ $26      ** DECIMAL 38  (SCREEN BASE
0027-           1070 HBASH .EQ $27      ** DECIMAL 39  ADDRESS)
0006-           1080 YO .EQ $6          ** DECIMAL 6
9464-           1110 YADDR .EQ $9464    ** DECIMAL 37988 (READ YTABLE)
C054-           1120 PAGE1 .EQ $C054
C055-           1130 PAGE1X .EQ $C055
0008-           1140 B0FLAG .EQ $08
9244- A5 FD     1170 SHIFTR LDA YB      ** CALL 37444 TO ENTER
9246- 85 06     1180  STA YO            ** STORE IN $6 FOR USE BY YADDR
9248- 20 64 94  1190 L1A JSR YADDR      ** RETURNS LO=HBASL/HI=HBASH
924B- A4 FF     1200  LDY HL            ** SET Y-REG TO LEFTMOST BYTE
924D- A2 00     1205  LDX #0
924F- 86 08     1210  STX B0FLAG        ** CLEAR BIT 0 FLAG
9251- 8D 55 C0  1215 L2A STA PAGE1X     ** DRAW AUXILIARY MEMORY
9254- 18        1220  CLC               ** SET TO SHIFT A 0 INTO BIT 0
9255- A5 08     1225  LDA B0FLAG        ** GET BIT 0 FLAG
9257- F0 01     1230  BEQ J1            ** IF IT'S 0-JUMP
9259- 38        1235  SEC               ** SET TO SHIFT A 1 INTO BIT 0
925A- B1 26     1240 J1 LDA (HBASL),Y   ** GET SCREEN BYTE
925C- 2A        1245  ROL               ** SHIFT BITS 0-6 TO 1-7
925D- 91 26     1250  STA (HBASL),Y     ** REPLACE BYTE ON SCREEN
925F- 8D 54 C0  1255  STA PAGE1         ** DRAW MAIN MEMORY
9262- 2A        1260  ROL               ** NOW PUT ORIGINAL BIT 6 INTO CARRY
9263- B1 26     1300  LDA (HBASL),Y     ** GET SCREEN BYTE
9265- 2A        1305  ROL               ** SHIFT BITS 0-6 TO 1-7 / CARRY TO BIT 0
9266- 91 26     1310  STA (HBASL),Y     ** REPLACE BYTE ON SCREEN
9268- 86 08     1315  STX B0FLAG        ** SET BIT 0 FLAG=0
926A- 2A        1320  ROL               ** NOW PUT ORIGINAL BIT 6 INTO CARRY
926B- 90 02     1325  BCC NC2           ** IF BIT 6 WAS 0 BEFORE SHIFT-JUMP
926D- E6 08     1330  INC B0FLAG        ** SET BIT 0 FLAG=1
926F- C8        1340 NC2 INY            ** POINT TO NEXT ADDRESS -->
9270- C4 FE     1370  CPY HR            ** HAVE WE PASSED HR YET?
9272- 90 DD     1380  BCC L2A           ** NO-GET THE NEXT ADDRESS
9274- F0 DB     1390  BEQ L2A           ** NO-WE'RE DOING HR NOW
9276- C6 06     1410 NXTLN DEC YO       ** MOVE UP TO NEXT LINE
9278- A5 06     1420  LDA YO            ** GET NEW Y-COORDINATE
927A- C9 FF     1430  CMP #$FF         ** HAS Y-COORDINATE REACHED 0?
927C- F0 04     1440  BEQ RTN2          ** YES-WE'RE FINISHED
927E- C5 FC     1450  CMP VT           ** HAVE WE PASSED VT?
9280- B0 C6     1460  BCS L1A           ** NO-START THE NEXT LINE
9282- 60        1470 RTN2 RTS           ** EXIT ROUTINE
```

LISTING 2: SHIFTL

```
                1000   .OR $91FE        ** DHR-SHIFTL
                1010   .TA $800         ** BY ROBERT DEVINE
                1015 ** COPYRIGHT 1984 BY MICROSPARC, INC.
00FC-           1020 YT .EQ $FC         ** DECIMAL 252
00FD-           1030 YB .EQ $FD         ** DECIMAL 253
00FE-           1040 HR .EQ $FE         ** DECIMAL 254
00FF-           1050 HL .EQ $FF         ** DECIMAL 255
0026-           1060 HBASL .EQ $26      ** DECIMAL 38  (SCREEN BASE
0027-           1070 HBASH .EQ $27      ** DECIMAL 39  ADDRESS)
0006-           1080 YO .EQ $6          ** DECIMAL 6
9464-           1110 YADDR .EQ $9464    ** DECIMAL 37988 (READ YTABLE)
C054-           1120 PAGE1 .EQ $C054
C055-           1130 PAGE1X .EQ $C055
0008-           1140 B6FLAG .EQ $08
91FE- A5 FD     1180 SHIFTL LDA VB      ** CALL 37374 TO ENTER
9200- 85 06     1190  STA YO            ** STORE IN $6 FOR USE BY YADDR
9202- 20 64 94  1200 L1A JSR YADDR      ** RETURNS-LO=HBASL/HI=HBASH
9205- A4 FE     1202  LDY HR            ** SET Y-REGISTER TO RIGHTMOST BYTE
9207- A2 00     1205  LDX #0
9209- 86 08     1210  STX B6FLAG        ** CLEAR BIT 6 FLAG
920B- 8D 54 C0  1215 L2A STA PAGE1      ** DRAW MAIN MEMORY
920E- B1 26     1220  LDA (HBASL),Y     ** GET SCREEN BYTE
9210- A6 08     1225  LDX B6FLAG        ** GET BIT 6 FLAG
9212- F0 02     1230  BEQ J1            ** IF BIT 6 WILL BE 0-JUMP
9214- 09 80     1235  ORA #$80          ** SET BIT 7=1 FOR SHIFT TO BIT 6
9216- 18        1240 J1 CLC             ** SET TO SHIFT 0 INTO BIT 7
9217- 6A        1245  ROR               ** SHIFT BITS 1-7 TO 0-6/BIT 0 INTO CARRY
9218- 91 26     1250  STA (HBASL),Y     ** REPLACE BYTE ON SCREEN
921A- 8D 55 C0  1275  STA PAGE1X        ** DRAW AUXILIARY MEMORY
921D- B1 26     1280  LDA (HBASL),Y     ** GET SCREEN BYTE
921F- 90 02     1290  BCC J3            ** IF CARRY=0 THEN BIT 6 WILL BE 0
9221- 09 80     1295  ORA #$80          ** SET BIT 7=1 FOR SHIFT TO BIT 6
9223- 18        1300 J3 CLC             ** SET TO SHIFT 0 INTO BIT 7
9224- 6A        1305  ROR               ** SHIFT BITS 1-7 TO 0-6/BIT 0 INTO CARRY
9225- 91 26     1310  STA (HBASL),Y     ** REPLACE BYTE ON SCREEN
9227- A2 00     1315  LDX #0            ** GET READY TO PUT 0 IN B6FLAG
9229- 90 01     1320  BCC J4            ** IF CARRY=0 THEN BIT 6 FLAG WILL BE 0
922B- E8        1325  INX               ** GET READY TO PUT 1 IN B6FLAG
922C- 86 08     1330 J4 STX B6FLAG      ** CONDITION BIT 6 FLAG
922E- 88        1360  DEY               ** POINT TO NEXT SCREEN ADDRESS
922F- C0 FF     1370  CPY #$FF          ** HAS Y-REGISTER PASSED 0 ?
9231- F0 04     1380  BEQ NXTLN2        ** YES-GOTO NEXT LINE
9233- C4 FF     1390  CPY HL            ** IS Y-REGISTER >=HL ?
9235- B0 D4     1400  BCS L2A           ** YES-JUMP TO LOOP2A
9237- C6 06     1410 NXTLN2 DEC YO      ** MOVE UP YO NEXT LINE
9239- A5 06     1420  LDA YO            ** GET NEW Y-COORDINATE
923B- C9 FF     1430  CMP #$FF         ** HAS YO PASSED 0 ?
923D- F0 04     1440  BEQ RTN2         ** YES-WE'RE FINISHED
923F- C5 FC     1450  CMP VT           ** HAVE WE REACHED VT YET ?
9241- B0 BF     1455  BCS L1A           ** NO-START THE NEXT LINE
9243- 60        1470 RTN2 RTS           ** DONE-EXIT ROUTINE
```

LISTING 4: SHIFT.TEST1

```
10  REM  ************************
20  REM  *      SHIFT.TEST1       *
30  REM  *   BY ROBERT DEVINE     *
40  REM  *  COPYRIGHT (C) 1984    *
50  REM  *  BY MICROSPARC, INC    *
60  REM  *  CONCORD, MA. 01742    *
70  REM  ************************
80  PRINT  CHR$ (4)"BLOAD DHR.DRIVER $91FE": CALL
     37999: HIMEM: 37374
90  PRINT  CHR$ (4)"BLOAD SHAPE#144"
100  CALL 37953: REM  INIT
110  HGR : CALL 37928: REM  CLEAR DHR SCREEN
120  POKE 49153,0: POKE 49234,0: REM  80STORE
     /FULL SCREEN
130  POKE 251,144: POKE 252,0: POKE 253,13: POKE
     254,2: POKE 255,0: CALL 37780: REM  DRAW
      SHAPE ON THE SCREEN
140  POKE 252,1: POKE 253,12: REM  REMOVE EXT
     RA ROWS ABOVE AND BELOW
150  POKE 254,3: REM  ADD 1 ADDRESS AHEAD
160  FOR HR = 3 TO 39: FOR SHFT = 1 TO 14: CALL
     37444: NEXT SHFT: CALL 37548: NEXT HR: REM
      SHIFTR/MOVERT TO THE RIGHT SIDE OF SCRE
     EN
170  FOR HR = 39 TO 3 STEP  - 1: FOR SHFT = 1
      TO 14: CALL 37374: NEXT SHFT: CALL 3755
     9: NEXT HR: REM  SHIFTL/MOVELF TO THE LE
     FT SIDE OF SCREEN
180  GOTO 160: REM  DO IT ALL OVER UNTIL THE
     POWER GOES OFF
```

LISTING 5: SHAPE#144

```
9000- 00 00 00 00 00 00 00 00
9008- 01 70 00 00 00 00 7F 7F
9010- 60 00 00 0F 7F 7F 7E 00
9018- 00 3F 7F 7F 7F 40 01 7F
9020- 7F 7F 7F 70 07 7F 7F 7F
9028- 7F 7C 1F 43 61 70 78 3F
9030- 1F 7F 7F 7F 7F 7F 01 7F
9038- 7F 7F 7F 70 00 0F 7F 7F
9040- 7E 00 00 00 7F 7F 60 00
9048- 00 00 07 7C 00 00 00 00
9050- 00 00 00 00
```

LISTING 3: DHR.DRIVER $91FE

```
91FE- A5 FD
9200- 85 Ø6 20 64 94 A4 FE A2
9208- ØØ 86 Ø8 8D 54 CØ B1 26
9210- A6 Ø8 FØ Ø2 Ø9 8Ø 18 6A
9218- 91 26 8D 55 CØ B1 26 9Ø
9220- Ø2 Ø9 8Ø 18 6A 91 26 A2
9228- ØØ 9Ø Ø1 E8 86 Ø8 88 CØ
9230- FF FØ Ø4 C4 FF BØ D4 C6
9238- Ø6 A5 Ø6 C9 FF FØ Ø4 C5
9240- FC BØ BF 6Ø A5 FD 85 Ø6
9248- 2Ø 64 94 A4 FF A2 ØØ 86
9250- Ø8 8D 55 CØ 18 A5 Ø8 FØ
9258- Ø1 38 B1 26 2A 91 26 8D
9260- 54 CØ 2A B1 26 2A 91 26
9268- 86 Ø8 2A 9Ø Ø2 E6 Ø8 C8
9270- C4 FE 9Ø DD FØ DB C6 Ø6
9278- A5 Ø6 C9 FF FØ Ø4 C5 FC
9280- BØ C6 6Ø A9 51 2Ø 92 92
9288- A9 26 4C 9F 92 A9 EA 2Ø
9290- 9F 92 8D 63 93 8D 72 93
9298- 8D AB 93 8D BA 93 6Ø 8D
92AØ- 64 93 8D 73 93 8D AC 93
92A8- 8D BB 93 6Ø A5 FE C9 27
92BØ- BØ Ø4 E6 FE E6 FF 6Ø A5
92B8- FF FØ Ø4 C6 FE C6 FF 6Ø
92CØ- A5 FC FØ Ø4 C6 FC C6 FD
92C8- 6Ø A5 FD C9 BF BØ Ø4 E6
92DØ- FC E6 FD 6Ø A5 FC 38 E5
92D8- E3 3Ø Ø9 85 FC A5 FD 38
92EØ- E5 E3 85 FD 6Ø A5 FD 18
92E8- 65 E3 C9 CØ BØ Ø9 85 FD
92FØ- A5 FC 18 65 E3 85 FC 6Ø
92F8- A9 ØØ 8D Ø1 CØ 85 FA A5
93ØØ- FD 85 Ø6 2Ø 64 94 A4 FF
93Ø8- 8D 55 CØ 2Ø 2B 93 8D 54
931Ø- CØ 2Ø 2B 93 C8 C4 FE 9Ø
9318- EF FØ ED C6 Ø6 A5 Ø6 C9
932Ø- FF FØ Ø4 C5 FC BØ DC 2Ø
9328- DA 93 6Ø A2 ØØ A1 FA C9
933Ø- 7F FØ 1Ø C9 Ø1 9Ø ØC 86
9338- F9 4A 26 F9 E8 EØ Ø7 9Ø
934Ø- F8 A5 F9 91 26 E6 FA DØ
9348- Ø2 E6 FB 6Ø A9 ØØ 8D Ø1
935Ø- CØ 85 FA A5 FC 85 Ø6 2Ø
9358- 64 94 A4 FE A2 ØØ A1 FA
936Ø- 8D 54 CØ 51 26 91 26 E6
9368- FA DØ Ø2 E6 FB A1 FA 8D
937Ø- 55 CØ 51 26 91 26 E6 FA
9378- DØ Ø2 E6 FB 88 CØ FF FØ
938Ø- Ø4 C4 FF BØ D9 E6 Ø6 A5
9388- Ø6 C9 FF FØ Ø6 C5 FD 9Ø
939Ø- C6 FØ C4 6Ø A9 ØØ 8D Ø1
9398- CØ 85 FA A5 FD 85 Ø6 2Ø
93AØ- 64 94 A4 FE A2 ØØ A1 FA
93A8- 8D 54 CØ 51 26 91 26 E6
93BØ- FA DØ Ø2 E6 FB A1 FA 8D
93B8- 55 CØ 51 26 91 26 E6 FA
93CØ- DØ Ø2 E6 FB 88 CØ FF FØ
93C8- Ø4 C4 FF BØ D9 C6 Ø6 A5
93DØ- Ø6 C9 FF FØ Ø4 C5 FC BØ
93D8- C6 6Ø A9 ØØ 8D Ø1 CØ 85
93EØ- FA A5 FD 85 Ø6 2Ø 64 94
93E8- A4 FE A2 ØØ 8D 54 CØ B1
93FØ- 26 81 FA E6 FA DØ Ø2 E6
93F8- FB 8D 55 CØ B1 26 81 FA
94ØØ- E6 FA DØ Ø2 E6 FB 88 CØ
94Ø8- FF FØ Ø4 C4 FF BØ DD C6
941Ø- Ø6 A5 Ø6 C9 FF FØ Ø4 C5
9418- FC BØ CA 6Ø A9 Ø4 85 3D
942Ø- 85 43 A9 Ø7 85 3F DØ ØA
9428- A9 2Ø 85 3D 85 43 A9 3F
943Ø- 85 3F A9 ØØ 85 3C 85 42
9438- A9 FF 85 3E 38 2Ø 11 C3
944Ø- 6Ø 8D 5E CØ 8D ØD CØ 8D
9448- 5Ø CØ 8D 57 CØ 6Ø 8D 5F
945Ø- CØ 8D ØC CØ 8D 51 CØ 8D
9458- 56 CØ 8D ØØ CØ 8D 54 CØ
946Ø- 2Ø 58 FC 6Ø A4 Ø6 B1 CE
9468- 85 26 B1 EE 85 27 6Ø A9
947Ø- 8Ø 85 CE A9 94 85 CF A9
9478- 4Ø 85 EE A9 95 85 EF 6Ø
948Ø- ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
9488- 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
949Ø- ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
9498- 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
94AØ- ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
94A8- 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
94BØ- ØØ ØØ ØØ ØØ ØØ ØØ ØØ ØØ
94B8- 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø 8Ø
94CØ- 28 28 28 28 28 28 28 28
94C8- A8 A8 A8 A8 A8 A8 A8 A8
94DØ- 28 28 28 28 28 28 28 28
94D8- A8 A8 A8 A8 A8 A8 A8 A8
94EØ- 28 28 28 28 28 28 28 28
94E8- A8 A8 A8 A8 A8 A8 A8 A8
94FØ- 28 28 28 28 28 28 28 28
94F8- A8 A8 A8 A8 A8 A8 A8 A8
95ØØ- 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø
95Ø8- DØ DØ DØ DØ DØ DØ DØ DØ
951Ø- 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø
9518- DØ DØ DØ DØ DØ DØ DØ DØ
952Ø- 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø
9528- DØ DØ DØ DØ DØ DØ DØ DØ
953Ø- 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø 5Ø
9538- DØ DØ DØ DØ DØ DØ DØ DØ
954Ø- 2Ø 24 28 2C 3Ø 34 38 3C
9548- 2Ø 24 28 2C 3Ø 34 38 3C
955Ø- 21 25 29 2D 31 35 39 3D
9558- 21 25 29 2D 31 35 39 3D
956Ø- 22 26 2A 2E 32 36 3A 3E
9568- 22 26 2A 2E 32 36 3A 3E
957Ø- 23 27 2B 2F 33 37 3B 3F
9578- 23 27 2B 2F 33 37 3B 3F
958Ø- 2Ø 24 28 2C 3Ø 34 38 3C
9588- 2Ø 24 28 2C 3Ø 34 38 3C
959Ø- 21 25 29 2D 31 35 39 3D
9598- 21 25 29 2D 31 35 39 3D
95AØ- 22 26 2A 2E 32 36 3A 3E
95A8- 22 26 2A 2E 32 36 3A 3E
95BØ- 23 27 2B 2F 33 37 3B 3F
95B8- 23 27 2B 2F 33 37 3B 3F
95CØ- 2Ø 24 28 2C 3Ø 34 38 3C
95C8- 2Ø 24 28 2C 3Ø 34 38 3C
95DØ- 21 25 29 2D 31 35 39 3D
95D8- 21 25 29 2D 31 35 39 3D
95EØ- 22 26 2A 2E 32 36 3A 3E
95E8- 22 26 2A 2E 32 36 3A 3E
95FØ- 23 27 2B 2F 33 37 3B 3F
95F8- 23 27 2B 2F 33 37 3B 3F
```

```
KEY PERFECT 4.Ø
RUN ON
DHR DRIVER $91FE
==================================
CODE      ADDR# - ADDR#
-------   --------------
294B      91FE  - 924D
22CA      924E  - 929D
2C22      929E  - 92ED
25E3      92EE  - 933D
27EF      933E  - 938D
262Ø      938E  - 93DD
2B16      93DE  - 942D
27B5      942E  - 947D
24C3      947E  - 94CD
272E      94CE  - 951D
2E47      951E  - 956D
27B2      956E  - 95BD
1FF8      95BE  - 95FF
PROGRAM CHECK IS : Ø4Ø2
```

LISTING 6: CREATE.PRE-SHIFTS

```
1Ø  REM  ************************
2Ø  REM  *   CREATE.PRE-SHIFTS  *
3Ø  REM  *   BY ROBERT DEVINE   *
4Ø  REM  * COPYRIGHT (C) 1984   *
5Ø  REM  * BY MICROSPARC, INC   *
6Ø  REM  * CONCORD, MA. Ø1742   *
7Ø  REM  ************************
8Ø  PRINT  CHR$ (4)"BLOAD DHR.DRIVER $91FE": CALL
     37999: HIMEM: 37374
9Ø  PRINT  CHR$ (4)"BLOAD SHAPE#144"
1ØØ  CALL 37953: REM  INIT
11Ø  HGR : CALL 37928: REM  CLEAR DHR SCREEN
12Ø  POKE 49153,Ø: POKE 49234,Ø: REM  8ØSTORE
     /FULL SCREEN
13Ø  POKE 251,144: POKE 252,Ø: POKE 253,13: POKE
     254,2: POKE 255,Ø: CALL 3778Ø: REM  DRAW
      SHAPE ON THE SCREEN
14Ø  GOTO 17Ø
15Ø  POKE 49236,Ø:C =  INT (X / 7): IF C / 2 =
      INT (C / 2) THEN  POKE 49237,Ø
16Ø XC =  INT (C / 2) + X / 7 - C:XC =  INT (
     XC * 7 + .5): RETURN
17Ø  HCOLOR= Ø: FOR X = Ø TO 4Ø STEP 4: GOSUB
     15Ø: HPLOT XC,Ø TO XC,13: NEXT
18Ø  POKE 252,1: POKE 253,12: POKE 254,4: REM
      REMOVE LINES ABOVE/BELOW - ADD 2 SHIFT
     ING ADDRESSES
19Ø  FOR X = 137 TO 144: POKE 251,X: REM  SET
     UP SHAPE #S
2ØØ  CALL 3785Ø: REM  SCAN THE SHAPE
21Ø  FOR SHFT = 1 TO 4: CALL 37444: NEXT SHFT
     : REM  MOVE IT OVER 4 DOTS
22Ø  NEXT X: CALL 37966
23Ø  PRINT  CHR$ (4)"BSAVE SHAPES 137-144,A$8
     9ØØ,L$8ØØ
```

LISTING 7: SHIFT.TEST2

```
1Ø  REM  ***********************
2Ø  REM  *    SHIFT.TEST2      *
3Ø  REM  *  BY ROBERT DEVINE   *
4Ø  REM  * COPYRIGHT (C) 1984  *
5Ø  REM  * BY MICROSPARC, INC  *
6Ø  REM  * CONCORD, MA. Ø1742  *
7Ø  REM  ***********************
8Ø  PRINT  CHR$ (4)"BLOAD DHR.DRIVER $91FE": CALL
     37999: HIMEM: 37374
9Ø  PRINT  CHR$ (4)"BLOAD SHAPES 137-144"
1ØØ  CALL 37953: REM  INIT
11Ø  HGR : CALL 37928: REM  CLEAR DHR SCREEN
12Ø  POKE 49153,Ø: POKE 49234,Ø: REM  8ØSTORE
     /FULL SCREEN
13Ø  CALL 37517: REM  TURN OFF EOR FUNCTION
14Ø  POKE 252,1Ø1: POKE 253,112: POKE 254,4: POKE
     255,Ø: REM  STARTING SHAPE LOCATION
15Ø  FOR HL = Ø TO 35 STEP 2
16Ø  FOR SHNUM = 138 TO 144: POKE 251,SHNUM: CALL
     3778Ø: NEXT SHNUM: REM  DRAW 7 SHAPE SEQ
     UENCE
17Ø  CALL 37548: CALL 37548: REM  DOUBLE INCR
     EMENT HR/HL
18Ø  NEXT HL
19Ø  CALL 37559
2ØØ  FOR HL = 35 TO Ø STEP  - 2
21Ø  FOR SHNUM = 143 TO 137 STEP  - 1: POKE 2
     51,SHNUM: CALL 3778Ø: NEXT SHNUM: REM  D
     RAW 7 SHAPE SEQUENCE
22Ø  CALL 37559: CALL 37559: REM  DOUBLE DECR
     EMENT HR/HL
23Ø  NEXT HL
24Ø  GOTO 15Ø
```