Block Shapes Part IV

by Robert R. Devine
P.O. Box 10
Adona, AK 72001

Howdy Pardner!! By now you should be starting to get a firm grasp on just how to create and deal with block shapes. If you haven't entered and spent some time playing with **Block Shape Maker** which appeared in *Nibble Vol. 4 No. 5*, you're missing a pretty good thing. Aside from its immediate utility in creating and modifying block shapes, it can be used to create and test almost any graphics effect you might like.

One of Block Shape Maker's main benefits is in helping to understand the Hi-Res screen. Trying different bit patterns within individual and adjoining bytes, as well as setting or clearing the Color bit can lead to many interesting, as well as unexpected effects on the screen. Try using Block Shape Maker as a learning and experimental tool as well as a Graphics utility.

So far, we've discovered just how easy block shapes are to create, as well as how quickly they can move about the screen.

As I've presented each new routine, or group of routines, and added them to the driver, I've tried to demonstrate how to use them by creating simple Applesoft CALLing programs. There are literally hundreds of ways that you can approach different animation requirements using these Driver routines. The tests that I have (and will) conduct, will certainly not be the only ways to handle an animation problem, and in fact they won't always be the best ways either.

You may want to think of the Driver routines as new Applesoft commands, whose proper syntax is CALL (routine address), and which cause one or more actions to take place. The test programs are very basic (and hopefully easily understood) demonstrations of how you can combine various commands to handle different graphics situations.

INTRODUCING SHIFT ANIMATION

You should now have a rather good picture in mind of how graphics animation works. Basically, in just about every method you've used, it's the same old thing: ERASE-MOVE-DRAW-ERASE-MOVE-DRAW-ERASE . . . ETC... ETC... In every method so far it's been a matter of constantly drawing and re-drawing the information in the Shape Table over and over and over again.

With **shift animation** you'll need to put away those old ideas and completely revise what you know about graphics animation. Shift animation is the closest thing there is to true animation, as we **do not use any Draw or Erase routines** in our animation. Basically it is just what it looks like: simply placing a shape on the screen, and then literally MOVING (shifting) it right or left across the screen.

The only time the shape is DRAWn is to originally place it on the screen. Another great thing about shift animation is that you can, if you wish, use only one Hi-Res screen, and the results will be just as smooth and flicker-free as what has, up until now, required page flip to accomplish. The problem is that true shift animation only works with shapes moving from right-left-right, so if your program has any shapes moving up and down, in addition to shapes moving from side to side, you may still want to work with page flip.

HOW SHIFT ANIMATION WORKS

The basic idea of shift animation is this. First you set up the proper VT, VB, HR, and HL information for your shape, and then DRAW it on the screen. Once the shape is on the screen (in Hi-Res memory), you change the value of HR (on a right moving shape) or HL (on a left moving shape), so that the HR-HL dimension is **one byte wider than the true width of the shape**. What this does is to place an entire vertical column of empty bytes immediately in **front** of the shape's direction of travel.

As you move your shape across the screen, you will shift every bit in the shape 1 position ahead in the direction of travel. The most forward bit in each byte will be shifted into the backmost position of the byte directly ahead of it.* The bits from the frontmost byte of the shape will be shifted into the empty column of bytes that were added directly in front of the shape bytes.

After seven shifts the shape will be completely shifted over one byte, at which time the empty column of bytes will have moved behind the shape. At this time the values of HR and HL are **both** incremented or decremented (depending on the direction of travel) so that the empty column of bytes is again **ahead** of the shape, in preparation for the next seven shifts.

Instead of the ERASE-MOVE-DRAW rhythm that you've become accustomed to, the rhythm with shift animation will be SHIFT SEVEN TIMES-MOVE HR/HL-SHIFT SEVEN TIMES-MOVE HR/HL . . . ETC.

*Bear in mind that this is more visually correct than technically correct. Due to the reversing effect of Hi-Res bytes, which we'll examine in some later examples, the bit shifts don't really do quite what they appear to do on the screen. There are two different Shift routines — SHFTL for moving your shape to the left, and SHFTR for moving it to the right. One pass through SHFTR or SHFTL will move the entire shape one bit (dot) to the right or left. If you want to move 20 dots in a particular direction, you'll need to go through SHFTR or SHFTL 20 times.

LEFT-RIGHT-LEFT

If you'll remember way back to the beginning of Part 1, we found out that Hi-Res graphics bytes are displayed on the screen in reverse order. What this really means is this: To **move** our shape RIGHT we need to **shift** the bits LEFT, and to **move** our shape LEFT we need to **shift** the bits RIGHT. The names of our routines will indicate the direction of apparent MOVEMENT, not the direction that we're actually shifting the bits.

THE SHFTR ROUTINE

Our first Shift Animation routine (lines 2100 through 2500 of Listing 1) is called SHFTR and moves your shape from left to right across the screen. SHFTR works basically the same as the routines that we've already looked at, except that like REVDIR, it begins to deal with the shape at VB/HL instead of VB/HR which is normal. When using SHFTR you will always need at least one empty column of bytes directly ahead (to the right) of your shape bytes, meaning that HR must always be overstated by at least one. SHFTR begins at $920E and fits directly under our Vertical Movement routines.

THE SHFTL ROUTINE

The next Shift routine (lines 1630-2070 of Listing 1) is called SHFTL and moves your shape from right to left across the screen. SHFTL is again very similar to our other routines and begins dealing with the shape at VB/HR. When using SHFTL you will always need at least one empty column of bytes directly to the left of your shape bytes, meaning that HL must always be understated by at least one. SHFTL begins at $91B5 and fits directly before the SHFTR routine.

MOVE ROUTINES

The next additions to your driver are the Move routines which fit directly under SHFTL and give the driver a new starting point of $9197 (lines 1200-1350 of Listing 1). The purpose of the Move routines is to provide a quick and easy method of changing the values of HR and HL. By CALLing one of several possible entry points, the routine can be used to INCrement or DECrement HR and HL by one or two, or INCrement or DECrement HR alone. The routine also checks to be sure that you don't accidentally allow SHFTL to go off the left edge of the screen and hang the program, with a negative value for HL.

At this point it might be a good idea to stop long enough to add these new routines to your driver, which as you should know by now can be done either with an assembler (I used the S-C Assembler) or by simply BLOADing the old routines and adding the hex bytes for the new routines. See the Letters section of this issue for information on typing in programs.

HOW THE SHIFT ROUTINES WORK

The principle involved in shift animation is very similar to what we've already used to SCAN, DRAW, and REVDIR our shapes. Each routine begins at the lowest row of screen bytes that our shape occupies, at the hindmost byte, then works through each horizontal row of bytes from the back (where we came from) to the front (where we're going) until all the rows have been processed.

The key commands in each routine are the ROL (ROtate Left) and ROR (ROtate Right) commands. While this series is not intended as a course in machine language, let's take a few moments to see what happens, using ROR as an example — see Figure 1.

As the byte is ROR'ed (ROtated Right) the contents of the carry (a special bit in the 6502's Status Register) is pushed into bit 7, which pushes all the other bits RIGHT one position. Then, like musical chairs, bit 0 falls out and lands in the carry. By conducting various tests, and setting different flags, each of the Shift routines is able to detect and affect the contents of the carry as well as keep the Color bit set properly.

Using these flags, the routines are able to push all the bits one position over, and know whether to add a '0' or a '1' to the next byte we're shifting into. The value being pushed out of bit 0 and into the carry is really intended for bit 6 of the next byte to the left.

As an example of how a Shift routine works, let's look at Figure 2, which shows SHFTL moving a two-byte wide shape, leftwards seven dots (one byte). Our sample shape will only be one byte high (a real big one), and look like this on the screen: **1110011-1001110**. In our example we will not worry about the Color bit (we'll assume that it's a zero), and only show the changes in the bits that appear on the screen.

When we originally drew the shape on the screen, it was two bytes wide and resided in bytes 8195 and 8196 at HR=4, HL=3. In order to use the SHFTL routine, HL was then changed to HL=2, adding the empty byte ahead of the shape at 8194.

You can see that after seven shifts, the shape has moved left so that the empty byte which **started in front** of our shape, **is now behind** the shape. You should also note that while the true values of HR and HL were 4 and 3 respectively before the seven shifts, after the seven shifts the true values of HR and HL changed to 3 and 2 respectively.

You'll see that the bits **appear** to move forward into the new byte in the screen display bytes; however, what's actually happening in the ROR'ed bytes is really quite different. The bits that are being rotated out of 8194 (HL) are not going anywhere. That question mark you see is really a huge black hole in space . . . No one has ever returned to tell the tale of what's there; however, we do know that the bits being shifted out of HL are going to end up forever falling through space. But that's okay, we don't need those 0 bits anyway. However, you'd better **be careful not to shift through HL more than seven times**, or you'll also begin to send your shape into that same black hole.

AFTER YOU'VE SHIFTED 7 TIMES YOU MUST CHANGE HR AND HL. If you're moving LEFT you must DECrement HR and HL every seven shifts, and if you're moving RIGHT you must INCrement HR and HL every seven shifts.

Let's try out some shifting . . . To run the tests that we're going to try out now, you'll need to add the SHFTR, SHFTL, and MOVE routines to the driver from my last article first. Once they've been added and SAVEd to disk with the name BLOCK ROUTINES $90AA, BLOAD $90AA them along with our sample spaceship shape #144, and enter the Applesoft program lines in Listing 2.

The very first thing you should notice when you RUN this program is just how smoothly the shape moves, as well as a total absence of any flicker — even though we're doing all of our work on one page. Normally you would need page flip for a shape to run this well.

Let's see how our program works . . .

Line 10 takes care of graphics, YTABLE, and shape# set-up.

Line 20 sets the starting values for VT, VB, HR, and HL, then draws the starting shape.

Line 30 INCrements HR to add an empty column of bytes RIGHT.

Line 50 sets the number of BYTES that we'll shift through.

Line 60 shifts through the current bytes seven times. (Moving right.)

Line 70 INCrements HR and HL in preparation for the next seven shifts.

Line 80 changes the empty column of bytes **left** for the return trip.

Line 110 shifts through the current bytes seven times. (Moving left.)

Line 120 DECrements HR and HL in preparation for the next seven shifts.

As you can see, the program is rather simple and straightforward. All we're doing is using loops so that we CALL a Shift routine seven times, and then change HR and HL before CALLing the same shift routine seven more times.

FIGURE 1: THE ROR COMMAND

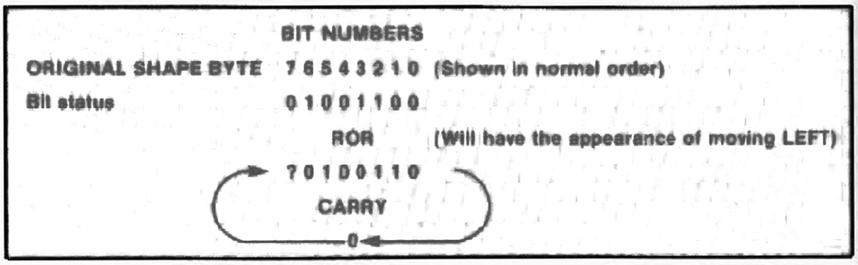
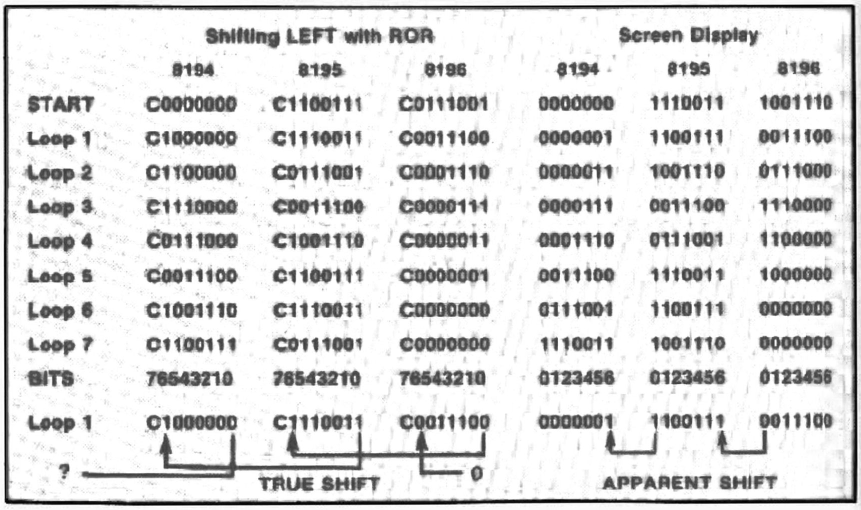

BIT NUMBERS
ORIGINAL SHAPE BYTE 7 6 5 4 3 2 1 0 (Shown in normal order)
Bit status 0 1 0 0 1 1 0 0
ROR (Will have the appearance of moving LEFT)
7 0 1 0 0 1 1 0
CARRY
0

FIGURE 2: HOW SHIFT ROUTINE WORKS



	Shifting LEFT with ROR			Screen Display		
	8194	8195	8196	8194	8195	8196
START	C0000000	C1100111	C0111001	0000000	1110011	1001110
Loop 1	C1000000	C1110011	C0011100	0000001	1100111	0011100
Loop 2	C1100000	C0111001	C0001110	0000011	1001110	0111000
Loop 3	C1110000	C0011100	C0000111	0000111	0011100	1110000
Loop 4	C0111000	C1001110	C0000011	0001110	0111001	1100000
Loop 5	C0011100	C1100111	C0000001	0011100	1110011	1000000
Loop 6	C1001110	C1110011	C0000000	0111001	1100111	0000000
Loop 7	C1100111	C0111001	C0000000	1110011	1001110	0000000
BITS	76543210	76543210	76543210	0123456	0123456	0123456
Loop 1	C1000000	C1110011	C0011100	0000001	1100111	0011100

LISTING 1: THE SHFTR ROUTINE / THE MOVEL AND MOVER
THE SHFTL ROUTINE / ROUTINES

```
:ASM

                1000  .OR $9197
                1010  .TA $800
00FC-           1020 VT .EQ $FC          ** DECIMAL 252
00FD-           1030 VB .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
00FA-           1090 BASL .EQ $FA        ** DECIMAL 250 (TABLE BASE
00FB-           1100 BASH .EQ $FB        ** DECIMAL 252 ADDRESS)
9391-           1110 YADDR .EQ $9391     ** DECIMAL 37777 (READ YTABLE)
9197- A5 FF     1200 MOVEL2 LDA HL       ** CALL 37271 TO ENTER
9199- C9 02     1210  CMP #2             ** IS HL<2?
919B- 90 0E     1220  BCC EXIT           ** YES-CANCEL EXECUTION
919D- C6 FF     1230  DEC HL             ** HL=HL-1
919F- C6 FE     1240  DEC HR             ** HR=HR-1
91A1- A5 FF     1250 MOVEL1 LDA HL       ** CALL 37281 TO ENTER
91A3- C9 01     1260  CMP #1             ** IS HL<1?
91A5- 90 04     1270  BCC EXIT           ** YES-CANCEL EXECUTION
91A7- C6 FF     1280  DEC HL             ** HL=HL-1
91A9- C6 FE     1290  DEC HR             ** HR=HR-1
91AB- 60        1300 EXIT RTS            ** DONE-EXIT ROUTINE
91AC- E6 FF     1310 MOVER2 INC HL       ** CALL 37292 TO ENTER
91AE- E6 FE     1320  INC HR             ** HR=HR+1:HL=HL+1
91B0- E6 FF     1330 MOVER1 INC HL       ** CALL 37296 TO ENTER
91B2- E6 FE     1340  INC HR             ** HR=HR+1:HL=HL+1
91B4- 60        1350  RTS                ** DONE-EXIT ROUTINE
91B5- A5 FD     1630 SHFTL LDA VB        ** CALL 37301 TO ENTER
91B7- 85 06     1640  STA YO             ** PUT IN $6 FOR USE BY YADDR
91B9- 20 91 93  1650 L3 JSR YADDR        ** RETURNS-LO=HBASL/HI=HBASH
91BC- 18        1660  CLC
91BD- A4 FE     1670  LDY HR             ** SET Y-REGISTER TO RIGHTMOST BYTE
91BF- A9 00     1680 ST LDA #0
91C1- 85 08     1690  STA $8             ** ZERO-BIT 0 FLAG LAST CYCLE
91C3- 85 09     1700  STA $9             ** ZERO-BIT 7 FLAG
91C5- 85 07     1710  STA $7             ** ZERO-BIT 0 FL:G THIS CYCLE
91C7- 90 02     1720  BCC CT1            ** IF BIT 0=0 JUMP
91C9- E6 08     1730  INC $8             ** SET-BIT 0 FLAG LAST CYCLE
91CB- B1 26     1740 CT1 LDA (HBASL),Y   ** GET SHAPE BYTE FROM SCREEN
91CD- C9 80     1750  CMP #$80           ** IS COLOR BIT SET ?
91CF- 90 02     1760  BCC CT2            ** NO-JUMP
91D1- E6 09     1770  INC $9             ** SET BIT 7 FLAG
```

Remember that this test is simply a way to demonstrate the way the routines work. In a real program environment, I wouldn't recommend regular FOR . . . NEXT loops for animation. They're easy and convenient, yes . . . but there are faster ways to build loops.

SHIFTING AND REVERSING

Now let's try out some shifting and reversing . . . To run this test you'll first need to reload the sample shape #146 (the Arkansas Good-Ole-Boy pickup truck) that we introduced in Part 2. Since we've added more to our Driver routines, you won't be able to run it as shape #146 anymore; so let's move it now to $9000 and rename it shape #144. Once you've got it in memory, enter the Monitor (CALL-151) and then use the Monitor MOVE command **9000<9200.9259M** to move it to page 144. Once it's moved, reSAVE it to disk with **BSAVE TRUCK #144,A$9000,L90**. You'll also need to reload the driver (you damaged it when you loaded the old truck shape), and then enter the Applesoft program lines in Listing 3.

When you RUN this program, you'll see the truck move smoothly across the screen and reverse direction before making the return trip. Once it gets back to the start position, it will again reverse so that it's always pointed in the proper direction. This test should also run as smoothly as the first, even though we're still working on only one page. Since the REVDIR routine does its work on the screen, the shape will never flicker, even during the reversing action. Let's first go through the program, as written, and then we'll look at how we could improve it by eliminating a few steps.

Lines 10-70 do all the same things as Listing 1 except that we're dealing with different values for VT, VB, HR, and HL.

Line 80 first removes the extra column of bytes ahead of our shape, returning it to its true dimensions, and then reverses the shape. After the shape is reversed, the extra column of bytes is replaced and moved over to the left side.

Line 130 again removes the extra column of bytes, reverses the shape, and replaces the extra column of bytes.

Our program worked quite well. However, it would seem that if we could eliminate the extra steps required to reset the true HR-HL dimensions prior to each reverse, it would be an improvement. Any steps that we can save will not only speed execution, but also save memory.

MAKE EMPTY COLUMN PERMANENT

Until now, we've been drawing our shape at its true HR/HL and then adding an empty column of bytes ahead. Now let's see what would happen if we made the empty column of bytes a permanent part of the shape. Before you enter Listing 4, let's modify the Shape Table using Listing 2.

First add the following line to Listing 3: **35 STOP**. Now RUN Listing 3. At this point your shape will be on the screen, and HR/HL will be properly set with an empty column of bytes ahead (to the right) of your shape. Now **CALL 37729 (SCAN)** to create a new Shape Table, and save it to disk with **BSAVE TRUCK+1 #144,A$9000,L105**.

```
91D3- A5 08     CT2 LDA $8          ** GET-BIT 0 LAST CYCLE FLAG
91D5- F0 07      BEQ CT3            ** IF IT'S CLEAR-JUMP
91D7- B1 26      LDA (HBASL),Y      ** GET SHAPE BYTE FROM SCREEN
91D9- 09 80      ORA #$80           ** SET BIT 7
91DB- 4C E2 91   JMP CT4            ** BIT 7 SET-CONTINUE
91DE- B1 26     CT3 LDA (HBASL),Y   ** GET SHAPE BYTE FROM SCREEN
91E0- 29 7F      AND #$7F           ** PUT 0 IN BIT 7
91E2- 6A        CT4 ROR             ** ROTATE BYTE RIGHT
91E3- 91 26      STA (HBASL),Y      ** LOAD BYTE ON SCREEN
91E5- 90 02      BCC CT5            ** IF BIT ZERO=0 JUMP
91E7- E6 07      INC $7             ** BUMP 0 BIT FLAG THIS CYCLE
91E9- A5 09     CT5 LDA $9          ** GET BIT 7 FLAG
91EB- C9 01      CMP #$1            ** IS IT SET ?
91ED- 90 06      BCC CT6            ** NO-JUMP
91EF- B1 26      LDA (HBASL),Y      ** GET SHAPE BYTE FROM SCREEN
91F1- 09 80      ORA #$80           ** PUT A 1 IN BIT 7
91F3- 91 26      STA (HBASL),Y      ** LOAD BYTE ON SCREEN
91F5- C4 FF     CT6 CPY HL          ** HAVE WE REACHED HL ?
91F7- F0 08      BEQ NXTLN3         ** YES-GOTO NEXT LINE
91F9- 88         DEY                ** POINT TO NEXT BYTE <--
91FA- A5 07      LDA $7             ** GET-0 BIT FLAG THIS CYCLE
91FC- C9 01      CMP #$1            ** IF 1 SET CARRY WITH CMP
91FE- 4C BF 91   JMP ST
9201- C6 06     NXTLN3 DEC YO       ** MOVE UP TO NEXT LINE
9203- A5 06      LDA YO             ** GET NEW Y-COORDINATE
9205- C9 FF      CMP #$FF           ** HAS Y-COORDINATE REACHED 0 ?
9207- F0 04      BEQ RTN3           ** YES-WE'RE FINISHED
9209- C5 FC      CMP VT             ** HAVE WE REACHED VT YET ?
920B- B0 AC      BCS L3             ** NO-START THE NEXT LINE
920D- 60        RTN3 RTS            ** DONE-EXIT ROUTINE
920E- A5 FD     SHFTR LDA VB        ** CALL 37390 TO ENTER
9210- 85 06      STA YO             ** STORE IN $6 FOR USE BY YADDR
9212- 20 91 93  L4 JSR YADDR        ** RETURNS-LO=HBASL/HI=HBASH
9215- 18         CLC
9216- A4 FF      LDY HL             ** SET Y-REG TO LEFTMOST BYTE
9218- A9 00     ST1 LDA #0          ** PUT A 0 IN ACCUMULATOR
921A- 85 08      STA $8             ** CLEAR BIT 7 FLAG
921C- 85 09      STA $9             ** CLEAR BIT 6 FLAG
921E- B1 26     SHF LDA (HBASL),Y   ** GET SHAPE BYTE FROM SCREEN
9220- 2A         ROL                ** ROTATE LEFT
9221- 91 26      STA (HBASL),Y      ** LOAD BYTE BACK ON SCREEN
9223- B0 02      BCS S1             ** IF BIT 7=1 BEFORE SHIFT-JUMP
9225- 90 02      BCC CT1A           ** IF BIT 7=0 BEFORE SHIFT-JUMP
9227- E6 08     S1 INC $8           ** SET BIT 7 FLAG
9229- C9 80     CT1A CMP #$80       ** IS BIT 7 NOW A 1 ?
922B- B0 02      BCS SET64          ** YES-GO SET BIT 6
922D- 90 02      BCC CT2A           ** NO-CONTINUE
922F- E6 09     SET64 INC $9        ** SET BIT 6 FLAG
9231- A5 08     CT2A LDA $8         ** GET BIT 7 FLAG
9233- D0 09      BNE S2             ** IF IT'S SET-JUMP
9235- B1 26      LDA (HBASL),Y      ** GET SHAPE BYTE FROM SCREEN
9237- 29 7F      AND #$7F           ** PUT A 0 IN BIT 7
9239- 91 26      STA (HBASL),Y      ** LOAD BYTE TO SCREEN
923B- 4C 44 92   JMP S3             ** BIT 7 OKAY-JUMP
923E- B1 26     S2 LDA (HBASL),Y    ** GET SHAPE BYTE FROM SCREEN
9240- 09 80      ORA #$80           ** PUT A 1 IN BIT 7
9242- 91 26      STA (HBASL),Y      ** LOAD BYTE TO SCREEN
9244- C4 FE     S3 CPY HR           ** HAVE WE REACHED HR YET ?
9246- F0 09      BEQ NXTLN4         ** YES-GOTO NEXT LINE
9248- C8         INY                ** POINT TO NEXT BYTE --->
9249- 18         CLC
924A- A5 09      LDA $9             ** GET BIT 6 FLAG
924C- C9 01      CMP #$1            ** USE CMP TO SET CARRY
924E- 4C 18 92   JMP ST1            ** GO SHIFT MORE BYTES
9251- C6 06     NXTLN4 DEC YO       ** MOVE UP TO NEXT LINE
9253- A5 06      LDA YO             ** GET NEW Y-COORDINATE
9255- C9 FF      CMP #$FF           ** HAS Y-COORDINATE REACHED 0 ?
9257- F0 04      BEQ RTN4           ** YES-WE'RE FINSIHED
9259- C5 FC      CMP VT             ** HAVE WE REACHED VT YET ?
925B- B0 B5      BCS L4             ** NO-START THE NEXT LINE
925D- 60        RTN4 RTS            ** DONE-EXIT ROUTINE
```

Now you can delete Listing 3, and enter Listing 4, which you'll notice is much simpler than Listing 3.

When you RUN Listing 4, you'll find that it runs just as well as our previous test even though it has fewer instructions. You probably won't be able to detect any speed differences, but in a program where you manipulate lots of shapes the differences could be quite apparent.

Here's how Listing 4 works:

Line 10 still does the same as in our previous tests.

Line 20 is the same as before, except that HR has now changed from 5 to 6.

Line 30 There is no line 30 . . . We don't need to add an extra column of bytes because it's already built into our shape!!

The only other changes are in lines 80 and 130:

Line 80 simply cancels out the last HR/HL INCrement from line 70 and reverses the shape, which also places the empty column of bytes to the left.

LISTING 2: SHIFTING EXERCISE

```
]LIST

5  REM  REQUIRES BLOCK ROUTINES $90AA AND BLOCK SHAPE
     #144
10  HGR : CALL 37799: POKE 251,144
20  POKE 252,10: POKE 253,21: POKE 254,2: POKE 255,0: CALL
     37679
30  CALL 37298: REM  INCREMENT HR-ADD EMPTY BYTE COLUM
     N RIGHT
50  FOR X = 0 TO 35
60  FOR Y = 1 TO 7: CALL 37390: NEXT : REM  SHIFT THRO
     UGH 1 BYTE/RIGHT
70  CALL 37296: REM  INCREMENT HR/HL
80  NEXT : CALL 37281: REM  SHIFT EMPTY COLUMN AHEAD/L
     EFT
100  FOR X = 0 TO 35
110  FOR Y = 1 TO 7: CALL 37301: NEXT : REM   SHIFT TH
     ROUGH 1 BYTE/LEFT
120  CALL 37281: REM  DECREMENT HR/HL
130  NEXT X
140  GOTO 50
```

LISTING 3: SHIFTING AND REVERSING EXERCISE

```
]LIST

5  REM  REQUIRES BLOCK ROUTINES $90AA AND TRUCK #144
10  HGR : CALL 37799: POKE 251,144
20  POKE 252,130: POKE 253,144: POKE 254,5: POKE 255,0
     : CALL 37679
30  CALL 37298: REM  ADD EMPTY BYTE COLUMN/RIGHT
50  FOR X = 0 TO 33
60  FOR Y = 1 TO 7: CALL 37390: NEXT : REM  SHIFT THRO
     UGH 1 BYTE
70  CALL 37296: REM  INCREMENT HR/HL
80  NEXT : CALL 37289: CALL 37606: CALL 37298: CALL 37
     281: REM  REMOVE LEAD BYTE/REVERSE/RESTORE LEAD B
     YTE/REVERSE LEAD BYTE
100  FOR X = 0 TO 33
110  FOR Y = 1 TO 7: CALL 37301: NEXT : REM   SHIFT TH
     ROUGH 1 BYTE/LEFT
120  CALL 37281: REM  DECREMENT HR/HL
130  NEXT : CALL 37289: CALL 37606: CALL 37298
140  GOTO 50
```

LISTING 4: KEEPING THE EMPTY COLUMN OF BYTES

```
]LIST

5  REM  REQUIRES BLOCK ROUTINES $90AA AND TRUCK+1 #144

10  HGR : CALL 37799: POKE 251,144
20  POKE 252,130: POKE 253,144: POKE 254,6: POKE 255,0
     : CALL 37679
50  FOR X = 0 TO 33
60  FOR Y = 1 TO 7: CALL 37390: NEXT : REM  SHIFT THRO
     UGH 1 BYTE
70  CALL 37296: REM  INCREMENT HR/HL
80  NEXT : CALL 37281: CALL 37606: REM  REVERSE SHAPE
100  FOR X = 0 TO 33
110  FOR Y = 1 TO 7: CALL 37301: NEXT : REM   SHIFT TH
     ROUGH 1 BYTE/LEFT
120  CALL 37281: REM  DECREMENT HR/HL
130  NEXT : CALL 37606: REM  REVERSE SHAPE
140  GOTO 50
```

Line 130 simply reverses the shape, again taking care of the empty column of bytes.

What we've done in Listing 4 is to make the empty column of bytes a permanent part of our shape. Each time that we go through one of the seven-shift loops, we send that first empty column of Shape Table bytes into the infamous black hole of space. However, since we replace it when we INCrement or DECrement HR/HL every seven shifts, it's okay, because we never shift far enough to damage the actual shape which doesn't begin until the second column of bytes in the table.

ANIMATION SPEED

We've been trying to demonstrate each of our tests in machine language to look at the greatest potential speed, so let's do the same with Listing 4. The hex dump in Listing 5 is a machine language version of Listing 4. To use it you'll need to put the driver and **TRUCK+1 #144** in memory, then enter the hex bytes through the Monitor beginning at $800. Once everything is on-board, enter HGR then CALL 2048.

This time your truck will move much more quickly across the screen. The only thing possibly wrong here is that since the shape is executing so fast your eye may actually detect some of the shifting that is going on. I don't know why this is, but I suspect that it's part of the same phenomenon that causes your eye to see individual blades slowly rotating on a rapidly spinning fan. In your normal program you'll probably be moving other shapes, as well as conducting various tests after each shift, which will slow things down enough to eliminate this effect.

The next thing we'll look at is how to deal with shift animation using page-flip, which you'll probably use in many of your programs.

USING SHIFT ANIMATION WITH PAGE-FLIP

Changing to page-flip with shift animation is fairly simple; however, there will be two major differences in how we will approach the subject. The first difference is that instead of moving only one dot per move, we will change to two dots per move, which is still a small enough move to keep the animation nice and smooth.

LISTING 5: MACHINE LANGUAGE VERSION OF LISTING 4

```
*800.851

0800-  20 A7 93 A9 90 85 FB A9
0808-  82 85 FC A9 90 85 FD A9
0810-  06 85 FE A9 00 85 FF 20
0818-  2F 93 A9 21 85 EF A9 07
0820-  85 EE 20 0E 92 C6 EE D0
0828-  F9 20 B0 91 C6 EF D0 EE
0830-  20 A1 91 20 E6 92 A9 21
0838-  85 EF A9 07 85 EE 20 B5
0840-  91 C6 EE D0 F9 20 A1 91
0848-  C6 EF D0 EE 20 E6 92 4C
0850-  1A 08
```

A very important benefit of using page flip animation is that it is color protected. If you were using a shape that contained color in our previous one-bit shift tests, the color would have flickered every shift. With page-flip, we'll only display the shape every two shifts, which will always keep the bits on ODD or EVEN coordinates, eliminating any visible change in color as we shift across the screen.

The second difference is that we will always place two columns of empty bytes directly in front of the shape bytes, and DOUBLE INCrement HR and HL every seven shifts. (Actually it's every 14 shifts, since we're shifting both pages.) To keep things easy to work with, we will incorporate the extra two columns of empty bytes as part of our shape.

THE SHIFT AND FLIP ROUTINES

At this time, let's look at some additional routines for the Block Shape Driver. We're now at the point where most of the essential parts of the driver have already been presented. The routines that we're looking at now are simply generalized routines intended to combine several Applesoft statements and CALLs into one machine language routine to help save memory, speed execution, and make your finished program easier to debug. If you're writing your CALLing program in assembly code, you may wish to discard that program and rewrite it so that it's more appropriate to your particular needs.

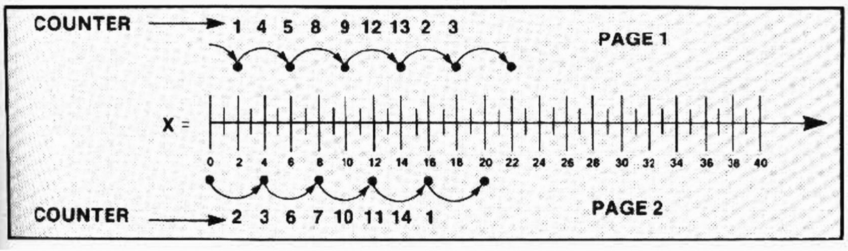
The SETCTR routine is used to set or reset the shift-loop counter to 14 so that you can keep track of where you are in the loops, and when you need to change the values of HR and HL.

The problem here was that, since I was trying to make these general purpose routines, I had to allow for the possibility of having several different shapes on the screen at the same time — all in different stages of their shift-loops, and each requiring separate counters. To handle this possibility, I built in a Counter Table, which is located at the end of memory page 143.

As you add more routines to the driver, you might find that you'll need to move the table lower, perhaps to $8E6F or $8D6F. To do so, simply change the address shown in lines 1090, 1500, 1540, and 1570. The way it's presently set up, there will be separate counters allocated for each individual shape#, with counter #144 being on the last byte of the page, and each shape#–counter# being at the next lower byte. You can think of the table as an Applesoft array.

POTENTIAL CHANGES TO THE COUNTER TABLE

If you've only got a few shapes to deal with, you might consider using an unused zero page address for your counters. This will speed execution; however, you'll need to revise the driver somewhat.



If you're dealing with a shape that is more than one page in length, you'll need to change things around a bit because once BASH is incremented by the DRAW or REVDIR routine, your shape# will point to the wrong counter. If you're packing more than one shape per page, this approach won't work at all.

For the normal one shape per page, this Counter Table should do quite nicely. Just beware of the potential hazards mentioned with other set-ups, and be careful not to let any Shape Tables that might be on page 143 overrun the Counter Table. The lowest byte used in the Counter Table will be **$8F6F+** (value of the LOWEST shape#).

The DRAW1 and DRAW2 routines in Listing 6 simply set the page to DRAW on and the page to DISPLAY. The name of the routine indicates which page will be DISPLAYed during the present drawing actions.

The SHFTR4 and SHFTL4 routines are our Page-Flip-Shift routines, and they handle the major part of our shift manipulation. SHFTR4 moves the shape right, and SHFTL4 moves it left. As we go through how they work, refer to **Figure 3**

Here's how they work. Our normal format for entering a Flip routine has always been to first display page 1 while manipulating page 2, then reverse pages, repeat the process, and exit the routine. We will use that same format here. Both routines work the same, so let's use SHFTR4 to see what they do.

Lines 1210-1230 set up our DISPLAY 1-DRAW 2 format and jump to line 1250.

Lines 1250-1260 shift the shape two dots right. (To the same coordinates as page 1).
Line 1270 jumps to MVCTRR where the counter is DECremented and tested for 14 shifts. If the counter indicates less than 14 shifts (actually it's 14 DOUBLE shifts), then the routine returns to line 1280. If you've completed 14 shifts, you jump to MOVER2 where HR and HL are **DOUBLE** INCremented. The counter is then reset to 14 and you again return to line 1280.
Lines 1280-1300 simply repeat the actions of 1250-1270, leaving the shape on page 2 moved two dots ahead of page 1.
Lines 1310-1330 test to see which page you're presently drawing on. If you're on page 2, the routine jumps to line 1240 where the page-flip occurs, and you drop through to execute lines 1250-1300 again.

When we get to lines 1310-1330, this time we'll be on page 1 with its shape two dots ahead of page 2, at which point we exit the routine.

As you can see, SHFTR4 and SHFTL4 do our total page-flip for us. The MVCTRR and MVCTRL routines simply handle DECrementing the counter, resetting it when 14 double shifts are completed, and double INCrementing (or DECrementing) HR and HL when needed.

HOW SHIFT PAGE-FLIP WORKS

Let's refer to Figure 3 to see just what we need to do to use our routines. In our example, X will refer to the rightmost dot in the shape.

When we first set up to work our shape, we need to DRAW the shape on Hi-Res page 2 at X=0. We then DRAW the shape on Hi-Res page 1 and shift it ahead two dots to X=2, and DECrement our counter to indicate that one double shift has been completed. At this time we're in proper starting position, with the shape on page 1 two dots ahead of the shape on page 2. For the balance of our trip across the screen, the SHFTR4 routine will do all we need to keep moving right.

At the end of our first loop through SHFTR4, the counter will read 5 and our shape will be at 6-page 1 and 4-page 2. After the second loop, the counter will read 9 and our shape will be at 10-page 1 and 8-page 2. The third loop will end with the counter reading 13 and our shape being at 14-page 1 and 12-page 2. As soon as we make the first double shift on page 2 during loop 4, the counter will have reached 14, at which time both HR and HL will be double INCremented. When we complete loop 4, the counter will read 3 and the shape will be at 18-page 1 and 16-page 2. This explanation of the counter contents is included for clarity but is not technically correct. We're not really starting at 0 and counting to 14; what we're really doing is starting at 14 and testing for when we reach 0.

Now that we've gone through the details, let's try to build a program that drives our pickup truck back and forth across the screen, and reverses it at each side so that it's always pointed in the proper direction. To run this test, you'll need to add the Shift and Flip routines to the driver and enter the Applesoft program lines in Listing 7. You'll also need to create a new Shape Table of your truck that has two columns of empty bytes ahead of it. The easy way to do this would be to reload Listing 4 along with **TRUCK+1 #144**. Then add line 30 **POKE 254,7:STOP**. Once you've RUN the program, your truck will be on the screen and the two empty bytes will be ahead. Then **CALL 37729** (SCAN) and finally **BSAVE TRUCK+2 #144,A$9000,L120**.

When you RUN this program you will, as in our previous tests, see the truck move smoothly back and forth, reversing direction at each side of the screen. Bear in mind that your shape doesn't need to be non-symmetrical to use this method of changing direction.

Let's see how the program works.
Line 10 clears both Hi-Res screens, and sets up YTABLE and our shape#.
Line 20 establishes the initial values of VT, VB, HR, and HL.
Line 30 Since the last screen we cleared was page 2, we're still on that page so we draw our starting shape at 0-page 2. (See Figure 2.)
Line 35 CALLs CTRSET to initialize our shift-loop counter.
Line 40 changes to page 1 (without changing the page that's displayed), DRAWs the starting shape, moves it right two bits (to its proper starting point), and DECrements the shift-loop counter. Note here that we were able to both execute the double shift right, and counter DECrement in only one step by

entering SHFTR4 at line 1280.

YOU SHOULD ALWAYS LOOK THROUGH EACH ROUTINE FOR ALTERNATIVE ENTRY POINTS which can often be used to solve special problems.

Lines 50-60 handle the movement of the shape all the way across the screen by simply CALLing SHFTR4 over and over again.

Line 70 When you get to this point, your shape on page 1 is shifted right two bits from its starting position in the byte, so you'll need to back it up two bits before you can ERASE it in preparation for the reverse. (Remember that when using REVDIR with page flip, you ERASE it on one page, REVDIR it on the other page, and finally reDRAW the reversed shape on the first page.) First, we set up to draw page 1 (again without changing the page that's displayed). Then, using the mid-entry point of SHFTL4, we back the shape up two bits, and finally, ERASE it.

Line 80 changes back to page 2 and simply reverses the shape.

Line 85 resets the shift-loop counter in readiness for the return trip.

Line 90 switches back to page 1, DRAWs the reversed shape, and uses the mid-entry point of SHFTL4 to move the shape ahead two dots and DECrement the counter in readiness for moving left.

Line 100-110 move the shape completely across the screen by CALLing SHFTL4 over and over again.

Line 120 This time the shape is again shifted two bits from its starting position in the byte on page 1, so we set up for page 1, CALL the mid-entry point of SHFTR4 to back it up two bits, and finally ERASE it from the screen.

Line 130 switches to page 2 and reverses the shape.

Line 140 jumps back to line 35 which will reset the counter and reDRAW the reversed shape on page 1 so that we can start all over again.

Bear in mind that there are many ways to write such a program, and our test is only one possible method. If, for example, you are going to have many shapes doing the back-and-forth reverse type actions, you may want to combine lines 70 through 90, or other such sequences, into one short machine language addition to the driver which will accomplish all those tasks in one simple CALL.

As always, let's look at how this test program would look in an all-machine language version. To use this program, you'll need the driver, **TRUCK+2 #144**, and the following hex dump in memory. To run it, simply enter HGR:HGR2 and then CALL 2048.

CONCLUSION

I hope you've found horizontal shift animation a good addition to your graphics library. Next time we'll take a look at how to develop Driver routines that will simulate vertical shift animation. By the time we're finished developing our Block Shape Driver routines, you'll have your choice of smooth, flicker-free Graphics routines that can be used for page-flip animation on either one or two pages.

As we continue to add routines to the driver, we will continue to add them at memory addresses just below previously presented routines. It's not likely that you'd use every routine in any given CALLing program, but if you have the room, you can simply load the completed driver and CALL or JSR the routines you plan to use. If memory becomes scarce you may want to rearrange the order of the routines, and load only those you need to use.

LISTING 6: THE SHIFT AND FLIP ROUTINES

```
:ASM

                  1000  .OR $911A
                  1010  .TA $800
91B5-             1020 SHFTL .EQ $91B5
920E-             1030 SHFTR .EQ $920E
9197-             1040 MOVEL2 .EQ $9197
91AC-             1050 MOVER2 :EQ $91AC
00FB-             1060 BASH .EQ $FB          ** SHAPE # (251)
911A- A9 0E       1070 SETCTR LDA #14        ** CALL 37146 TO ENTER
911C- A6 FB       1080  LDX BASH             ** SET X-REGISTER OFFSET
911E- 9D 6F 8F    1090  STA $8F6F,X          ** RESET COUNTER TO 14
9121- 60          1100  RTS                  ** DONE-EXIT
9122- A9 00       1110 DRAW1 LDA #0          ** CALL 37154 TO ENTER
9124- 8D 54 C0    1120  STA $C054            ** DISPLAY PAGE 1
9127- A9 40       1130  LDA #$40
9129- 85 E6       1140  STA $E6              ** DRAW PAGE 2
912B- 60          1150  RTS                  ** DONE-EXIT
912C- A9 00       1160 DRAW2 LDA #0          ** CALL 37164 TO ENTER
912E- 8D 55 C0    1170  STA $C055            ** DISPLAY PAGE 2
9131- A9 20       1180  LDA #$20
9133- 85 E6       1190  STA $E6              ** DRAW PAGE 1
9135- 60          1200  RTS                  ** DONE-EXIT
9136- 20 22 91    1210 SHFTR4 JSR DRAW1      ** CALL 37174 TO ENTER
9139- 18          1220  CLC                  ** SET-UP FOR JUMP
913A- 90 03       1230  BCC J2               ** GO SHIFT PAGE 2
913C- 20 2C 91    1240 J1 JSR DRAW2          ** SHIFT PAGE 1 (CALL 37180)
913F- 20 0E 92    1250 J2 JSR SHFTR          ** FIRST SHIFT
9142- 20 0E 92    1260  JSR SHFTR            ** SECOND SHIFT
9145- 20 7A 91    1270  JSR MVCTRR           ** DEC CTR/TEST 14 SHIFTS
9148- 20 0E 92    1280  JSR SHFTR            ** THIRD SHIFT (CALL 37192)
914B- 20 0E 92    1290  JSR SHFTR            ** FOURTH SHIFT
914E- 20 7A 91    1300  JSR MVCTRR           ** DEC CTR/TEST 14 SHIFTS
9151- A5 E6       1310  LDA $E6              ** WHAT PAGE ARE WE ON?
9153- C9 40       1320  CMP #$40             ** ARE WE ON PAGE 2?
9155- F0 E5       1330  BEQ J1               ** YES-NOW DO PAGE 1
9157- 60          1340  RTS                  ** DONE-EXIT ROUTINE
9158- 20 22 91    1350 SHFTL4 JSR DRAW1      ** CALL 37208 TO ENTER
915B- 18          1360  CLC                  ** SET-UP FOR JUMP
915C- 90 03       1370  BCC J4               ** GO SHIFT PAGE 2
915E- 20 2C 91    1380 J3 JSR DRAW2          ** SHIFT PAGE 1 (CALL 37214)
9161- 20 B5 91    1390 J4 JSR SHFTL          ** FIRST SHIFT
9164- 20 B5 91    1400  JSR SHFTL            ** SECOND SHIFT
9167- 20 8A 91    1410  JSR MVCTRL           ** DEC CTR/TEST 14 SHIFTS
916A- 20 B5 91    1420  JSR SHFTL            ** THIRD SHIFT (CALL 37226)
916D- 20 B5 91    1430  JSR SHFTL            ** FOURTH SHIFT
9170- 20 8A 91    1440  JSR MVCTRL           ** DEC CTR/TEST 14 SHIFTS
9173- A5 E6       1450  LDA $E6              ** WHAT PAGE ARE WE ON?
9175- C9 40       1460  CMP #$40             ** ARE WE ON PAGE 2?
9177- F0 E5       1470  BEQ J3               ** YES-NOW DO PAGE 1
9179- 60          1480  RTS                  ** DONE-EXIT ROUTINE
917A- A6 FB       1490 MVCTRR LDX BASH       ** CALL 37242 TO ENTER
917C- DE 6F 8F    1500  DEC $8F6F,X          ** DECREMENT COUNTER
917F- D0 08       1510  BNE J5               ** LESS THAN 14 SHIFTS-JUMP
9181- 20 AC 91    1520  JSR MOVER2           ** DOUBLE INCREMENT HR/HL
9184- A9 0E       1530 J6 LDA #14            ** RESET SHIFT-
9186- 9D 6F 8F    1540  STA $8F6F,X          ** COUNTER TO 14
9189- 60          1550 J5 RTS                ** DONE-EXIT ROUTINE
918A- A6 FB       1560 MVCTRL LDX BASH       ** CALL 37258 TO ENTER
918C- DE 6F 8F    1570  DEC $8F6F,X          ** DECREMENT COUNTER
918F- D0 F8       1580  BNE J5               ** LESS THAN 14 SHIFTS-JUMP
9191- 20 97 91    1585  JSR MOVEL2           ** DOUBLE DECREMENT HR/HL
9194- 18          1590  CLC                  ** SET-UP FOR JUMP
9195- 90 ED       1600  BCC J6               ** GO TO COUNTER RESET
```

LISTING 7: EXPANDING THE SHAPE'S CAPABILITIES

```
]LIST

10  HGR : HGR2 : CALL 37799: POKE 251,144
20  POKE 252,130: POKE 253,144: POKE 254,7: POKE 255,0

30  CALL 37679: REM  DRAW SHAPE ON PAGE 2
35  CALL 37146: REM  SET STARTING COUNTER=14
40  POKE 230,32: CALL 37679: CALL 37192: REM  DRAW PAG
    E 1/SHIFT RIGHT 2/DEC COUNTER
50  FOR X = 1 TO 56
60  CALL 37174: NEXT
70  POKE 230,32: CALL 37226: CALL 37679: REM  MOVE BAC
    K 2/ERASE PAGE 1
80  POKE 230,64: CALL 37606: REM  REVERSE PAGE 2
85  CALL 37146: REM  RESET COUNTER=14
90  POKE 230,32: CALL 37679: CALL 37226: REM  DRAW PAG
    E 1/MOVE AHEAD 2
100  FOR X = 1 TO 56
110  CALL 37208: NEXT
120  POKE 230,32: CALL 37192: CALL 37679: REM   MOVE B
    ACK 2/ERASE PAGE 1
130  POKE 230,64: CALL 37606: REM  REVERSE PAGE 2
140  GOTO 35
```

SUMMARY OF NEW DRIVER ENTRY POINTS

Routine Name	CALL Address	Hex Address	Routine Function
SHFTR	37390	$920E	Shift entire block shape RIGHT 1 dot.
SHFTL	37301	$91B5	Shift entire block shape LEFT 1 dot.
MOVEL2	37271	$9197	Subtract 2 from HR and HL.
	37279	$919F	Subtract 2 from HR/subtract 1 from HL.
MOVEL1	37281	$91A1	Subtract 1 from HR and HL.
	37289	$91A9	Subtract 1 from HR.
MOVER2	37292	$91AC	Add 2 to HR and HL.
	37294	$91AE	Add 2 to HR/add 1 to HL.
MOVER1	37296	$91B0	Add 1 to HR and HL.
	37298	$91B2	Add 1 to HR.
SETCTR	37146	$911A	Set shape shift-loop counter to 14.
DRAW1	37154	$9122	Set to display page 1/DRAW page 2.
	37159	$9127	Set to DRAW page 2.
DRAW2	37164	$912C	Set to display page 2/DRAW page 1.
	37169	$9131	Set to DRAW page 1.
SHFTR4	37174	$9136	Execute complete page-flip-shift moving RIGHT. Page 2 - Shift four dots/bump counter two times. Page 1 - Shift four dots/bump counter two times.
	37180	$913C	Shift page 1 only. Shift four dots/bump counter two times.
	37192	$9148	Shift two dots/bump counter one time. (Must set for page 1 before entering here; otherwise, you will shift page 2 two dots/bump counter one time, then shift page 1 four times/bump counter two times.)
SHFTL4	37208	$9158	Same as SHFTR4 except moving LEFT.
	37214	$915E	Same as 37180 except moving LEFT.
	37226	$916A	Same as 37192 except moving LEFT.
MVCTRR	37242	$917A	Bump counter/test for 14 double shifts. If 14, then add 2 to HR and HL/reset counter to 14.
MVCTRL	37258	$918A	Bump counter/test for 14 double shifts. If 14, then subtract 2 from HR and HL/reset counter to 14.

This should be fairly easy as we've been looking at most routines as independent, standalone units. However, if you do move them around, you'll need to change your CALL addresses.

When we're all finished, we'll spend a moment or two discussing how to disassemble a MOVE routine to find the new entry points.

See you next time!!!

SPECIAL POKEs to use with the driver:

POKE 37201,96 **POKE 37235,96**	Modify SHFTR4 and SHFTL4 to cancel automatic page-flip and handle only one page at a time (for use with multiple shapes).
POKE 37201,165 **POKE 37235,165**	Restore SHFTR4 and SHFTL4 to normal page-flip (for use with one shape only).

As you should recall from our earlier discussions, when you're dealing with more than one shape at a time, you should move all your shapes on one page before exchanging pages. When dealing with more than one shape, first enter the POKE's to cancel page-flip, then CALL SHFTR4 or SHFTL4 to shift page 2. Next move your other shapes on page 2. Finally, enter SHFTR4 or SHFTL4 at 37180 or 37214 to flip pages and shift the shape on page 1 — after which you'll need to complete the move for your other shapes on page 1.

The first two POKE's replace the LDA in lines 1310 and 1450 of LISTING 6 with RTS, and the second two POKE's replace the LDA in these same lines.

LISTING 8: MACHINE LANGUAGE VERSION OF LISTING 7

```
*800.86C

0800- 20 A7 93 A9 90 85 FB 85
0808- FD A9 82 85 FC A9 07 85
0810- FE A9 00 85 FF 20 2F 93
0818- 20 1A 91 A9 20 85 E6 20
0820- 2F 93 20 48 91 A9 38 85
0828- 1B 20 36 91 C6 1B D0 F9
0830- A9 20 85 E6 20 6A 91 20
0838- 2F 93 A9 40 85 E6 20 E6
0840- 92 20 1A 91 A9 20 85 E6
0848- 20 2F 93 20 6A 91 A9 38
0850- 85 1B 20 58 91 C6 1B D0
0858- F9 A9 20 85 E6 20 48 91
0860- 20 2F 93 A9 40 85 E6 20
0868- E6 92 4C 18 08
```