HI-RES SCRN COMMAND

Add a Hi-Res *version of the Lo-Res SCRN command to your toolbox! It can be used from Applesoft or from machine language.*

One of the handier features of low resolution graphics is the SCRN function. The statement SCRN (X,Y) returns the color of the block at the point X,Y. This function is extremely useful for game programming.

But what if you want to make the big jump from low resolution to high resolution graphics? You lose the SCRN command because Applesoft BASIC doesn't have a Hi-Res equivalent. Faced with this problem a few months back, I wrote HI.RES.SCRN, a machine language utility that determines whether a Hi-Res dot (pixel) is on or off.

USING HI-RES SCRN

To use HI.RES.SCRN (**Listing 1**), first BLOAD it, then type:

CALL 768,*x,y*

where *x* is the X-coordinate and *y* is the Y-coordinate of the pixel to be checked. This can be used from within an Applesoft program or in immediate mode from the keyboard. The result is obtained by performing a PEEK(242). If the result is one, then the pixel specified by *x* and *y* is on; if it's zero, then the pixel is off.

If you wish to call this routine from a machine language program, follow this format:

1. Load the X-Register with the least significant byte (LSB) of the X-coordinate.
2. Load the Y-Register with the most significant byte (MSB) of the X-coordinate.
3. Load the Accumulator with the Y-coordinate.
4. JSR $306.

When HI.RES.SCRN is finished doing its stuff, it leaves the result in location $F2.

When HI.RES.SCRN is finished doing its stuff, it leaves the result in location $F2 . . .

A single pixel has no particular color, of course. It can only be on or off. Color is produced by the position and combination of pixels on the Hi-Res screen, and there is no way to directly read the color.

Listing 2 is a demonstration program that shows how HI.RES.SCRN can be used in a game. The program draws a box in the center of the Hi-Res screen, and then randomly bounces a dot in the box. When the moving dot hits the border of the box, you get a buzz. I call this routine "The Angry Bee." I hope it gives you a few ideas for future game programs.

ENTERING THE PROGRAMS

If you have an assembler, you may enter the assembly language code in **Listing 1** and assemble it to produce the final program. Alternatively, you may enter the machine language code directly from the Monitor, and then save the program with:

BSAVE HI.RES.SCRN,A$300,L$40

Next, type in the Applesoft program in **Listing 2** and save it with:

SAVE THE.ANGRY.BEE

For help with entering *Nibble* programs, see the Program Listings section at the end of this issue.

HOW IT WORKS

Much of the work of HI.RES.SCRN is accomplished by using built-in ROM routines. **Line 26** of **Listing 1** jumps to CHKCOM to check for a comma at TXTPTR. If one is not there, an error message is returned.

Line 27 uses a ROM routine called HFNS, which BASIC uses to get the coordinates for a Hi-Res plot for the **HPLOT X,Y** command. The value of X must be between 0 and 279, and the value of Y must be between 0 and 191. On return from this routine, the X-Register has the LSB of the X-coordinate, the Y-Register has the MSB of the X-coordinate, and the Accumulator has the Y-coordinate. This is the format that is required when using the HPOSN routine.

The first problem I encountered was how to find the byte that corresponds to the dot to be checked. You may know that the Hi-Res screen is not stored sequentially in memory. That is, line 125 on the Hi-Res screen does not come after line 124 — it comes after line 61. Fortunately, after the jump to HPOSN in **line 30**, the address of the left end of the screen display line upon which the desired point appears is stored in HBASL and HBASH ($26 and $27). To find the byte in the display line that has the point to be checked, the X-coordinate is divided by seven. **Lines 31-43** are the division routine. By adding the result of the division to the address in HBASL and HBASH, the byte to be checked is located.

After clearing RESULT to zero in **lines 44 and 45**, the specific byte to be checked is loaded into the Accumulator. We are now down to figuring out which bit in the Accumulator is the one to be checked.

The remainder of the division is stored in MSBDIV, which now holds the bit to be checked. If there is no remainder, then bit 0 is the bit to be checked. If one is the remainder, then bit 1 is to be checked, and so on.

The remainder is used as an index to the data table (**lines 51-60**) to find the bit alignment with which to AND the Accumulator. The AND instruction compares the corresponding bits in the Accumulator to the bits in the data table. The result of the AND instruction is stored in the Accumulator.

The following is an example of two bytes being ANDed:

```
Byte 1 = 0110010
Byte 2 = 0010000
------------
Result = 0010000
```

If a bit is a one in both values, the resulting bit is a one. If the bit of either value is a zero, then the resulting bit is a zero.

The ANDing of the Accumulator has now cleared all of the bits to zero, except the one to be checked. If it is a one, a one is stored in RESULT. If the bit is a zero, RESULT remains zero, and the program is exited.

Listing 1 for Hi-Res SCRN Command
HI.RES.SCRN

```
                  1    ******************************
                  2    * HI.RES.SCRN                *
                  3    * BY ADAM COTI               *
                  4    * COPYRIGHT (C) 1987         *
                  5    * BY MICROSPARC, INC.        *
                  6    * CONCORD, MA  01742         *
                  7    ******************************
                  8    *     Format: CALL 768,X,Y   *
                  9    *    Where X= X-coordinate   *
                  10   *      and Y= Y-coordinate   *
                  11   *    of point to be checked  *
                  12   ******************************
                  13   * MERLIN ASSEMBLER           *
                  14   ******************************
                  15             ORG    $300
                  16   *
                  17   HBASL     =      $26
                  18   RESULT    =      $F2
                  19   DIVISOR   =      $FD
                  20   LSBDIV    =      $FE
                  21   MSBDIV    =      $FF
                  22   CHKCOM    =      $DEBE
                  23   HPOSN     =      $F411
                  24   HFNS      =      $F6B9
                  25   *
0300: 20 BE DE    26             JSR    CHKCOM     ; Checks for a comma.
0303: 20 B9 F6    27             JSR    HFNS       ; Get X and Y coordinates.
0306: 86 FE       28             STX    LSBDIV     ;   Store LSB of X- coordinate.
0308: 84 FF       29             STY    MSBDIV     ;   Store MSB of X- coordinate.
030A: 20 11 F4    30             JSR    HPOSN      ; Position the hi-res cursor.
030D: A2 08       31             LDX    #$08
030F: A9 07       32             LDA    #7
0311: 85 FD       33             STA    DIVISOR
0313: A5 FF       34             LDA    MSBDIV
0315: 06 FE       35   TR0       ASL    LSBDIV     ; This is a routine that divides
0317: 2A          36             ROL               ;   the two-byte number at LSBDIV
0318: C5 FD       37             CMP    DIVISOR    ;   and MSBDIV with the number at
031A: 90 04       38             BCC    TR1        ;   DIVISOR.  The answer is in LSBDIV
031C: E5 FD       39             SBC    DIVISOR    ;   with the remainder in MSBDIV.
031E: E6 FE       40             INC    LSBDIV
0320: CA          41   TR1       DEX
0321: D0 F2       42             BNE    TR0
0323: 85 FF       43             STA    MSBDIV
0325: A9 00       44             LDA    #0         ; Store '0' at RESULT.
0327: 85 F2       45             STA    RESULT
0329: A4 FE       46             LDY    LSBDIV     ; Get byte to be checked.
032B: B1 26       47             LDA    (HBASL),Y
032D: A4 FF       48             LDY    MSBDIV
032F: 39 39 03    49             AND    DATA,Y     ; Clear byte to '0' except for bit
0332: F0 04       50             BEQ    END        ;   to be checked.  End if bit is off.
0334: A9 01       51             LDA    #1         ; Bit is on, so store '1' in RESULT.
0336: 85 F2       52             STA    RESULT
0338: 60          53   END       RTS
0339: 01          54   DATA      DFB    %00000001
033A: 02          55             DFB    %00000010
033B: 04          56             DFB    %00000100
033C: 08          57             DFB    %00001000
033D: 10          58             DFB    %00010000
033E: 20          59             DFB    %00100000
033F: 40          60             DFB    %01000000

--End assembly, 64 bytes, Errors: 0
```

END OF LISTING 1

Listing 2 for Hi-Res SCRN Command
THE.ANGRY.BEE

```
10  REM  ************************
20  REM  * THE.ANGRY.BEE        *
30  REM  * BY ADAM COTI         *
40  REM  * COPYRIGHT (C) 1987   *
50  REM  * BY MICROSPARC, INC.  *
60  REM  * CONCORD, MA  01742   *
70  REM  ************************
80  ONERR  GOTO 230
90  PRINT  CHR$ (4)"BLOAD HI.RES.SCRN"
100  HGR : HCOLOR= 3: HOME : ONERR  GOTO 220
110  HPLOT 115,50 TO 135,50 TO 140,100 TO 110
     ,100 TO 115,50
120  HCOLOR= 0: HPLOT 130,50 TO 133,50: HCOLOR=
     3
130 X = 130:Y = 75:A = 1:B = 1
140  CALL 768,X + A,Y + B
150  IF  PEEK (242) = 1 THEN  GOTO 190
160 X = X + A:Y = Y + B
170  HPLOT X,Y: HCOLOR= 0: HPLOT X - A,Y - B:
      HCOLOR= 3
180  GOTO 140
190  FOR C = 1 TO 5:S =  PEEK ( - 16336): NEXT
     C
200 A =  INT ( RND (1) * 3) - 1:B =  INT ( RND
     (1) * 3) - 1
210  GOTO 140
220  VTAB 23: HTAB 11: PRINT "FREEDOM AT LAST
     ": END
230  TEXT : HOME : VTAB 5: PRINT "THIS PROGRA
     M REQUIRES THE HI.RES.SCRN": PRINT "FILE
      TO BE ON THE SAME DISK."
```

END OF LISTING 2