WINDOW DISPLAY

The ability to create 'windows' on a screen display is a particularly useful facility. We take a detailed look at a machine code program to do just this on the Spectrum, and give an accompanying BASIC demonstration program that shows the effect of scrolling on the screen.

Open Window
The word "HELLO" is scrolled horizontally and vertically around the screen window, one pixel at a time

Our machine code program allows you to define rectangular windows on the Spectrum screen, and to scroll these windows left and right, or up and down. The windows may be of any size and can be placed anywhere on the screen; they do not need to occupy eight by eight pixel character squares.

The machine code program uses tables starting at address $B004 (45060 decimal) to handle window parameters and for temporary data storage. It takes the address of the table for the required window from addresses $B000 and $B001 (45056 and 45057 decimal). The table for each window takes up 11 bytes, so if more than one window is required, the table for the second window will start at $B00F (45071 decimal), the table for the third window will begin at $B01A (45082 decimal), and so on.

The BASIC demonstration program uses one window only. The details of this window are POKEd into memory in lines 180-230, and the window initialisation routine is called at line 240. If more than one window is required, each one must be defined in this way before it can be used. To change windows you must POKE the address of the new window table into memory locations WT and WT+1. Scroll directions are set by POKEing values into memory location WNDWTB + DIR — simply POKE 0 to scroll left, 1 to scroll right, 2 for up, or 3 for down.

The Assembly language program begins by defining a number of constants. PIXADR is a subroutine in the Spectrum ROM that calculates the address of the screen byte, and the bit number within that byte, of a point on the screen that is defined by its PLOT co-ordinates. PIXADR takes the y co-ordinate in the B register and the x co-ordinate in the C register and returns the screen address in the HL register pair and the bit position in the A register.

The routine INITW first checks that the co-ordinates for the bottom right corner of the window are in fact below and to the right of the co-ordinates of the top left corner, and also ensures that the left and right margins are not both in the same byte of screen memory. This last check ensures that the window is at least one character square in width — extra code would be required to scroll a window that is any narrower.

Errors in window initialisation are printed out by the ROM error message routine. The instruction RST 8 (line 2110 of the Assembly listing) calls the ROM routine and returns to BASIC command mode, and the DEFB 25 in line 2120 provides the message 'Q Parameter error'.

The last section of INITW calculates LFTMSK and RTMASK, which are used when scrolling the screen bytes at the window margins, where part of the screen byte may be inside the window and part outside. The individual bits in the masks that correspond to screen bits that lie outside the window margins are set to one, while bits corresponding to bits inside the window area are set to zero.

The scroll program proper begins at the label SCROLL. The program tests the direction of scrolling and calls HORIZ for left or right scrolling, or VERT for up or down scrolling.

Left and right scrolling are very similar in operation, so instead of using two separate routines we have combined them in one. The correct code for each scroll direction is selected by testing bit zero of the direction byte. To see how HORIZ works, we will look at leftward scrolling.

Both left and right scrolling start with the top row of pixels in the window and work downwards, so HORIZ begins by copying the y co-ordinate for the top row into the temporary store for the current row. When scrolling left, we must start at the right-hand end of each row of pixels in the window and work towards the left. To prepare for this, RMASK and LMASK are copied to MASK1 and MASK2 respectively, the address of the screen byte at the left-hand end of the current row of pixels is calculated and stored in the DE register pair, and the address of the screen byte at the right-hand end of the current row of pixels is calculated and stored in the HL register pair. The subroutine HLNSCR is then called to scroll the row of pixels. The routine tests to see if the bottom row of the window has been reached, and if not it takes the next pixel row and jumps back to HORIZ3 to scroll again.

HLNSCR begins at one end of the pixel row, with a byte that may have some bits inside and some outside the window, then moves along the bytes that are contained wholly inside the window and finishes at the other window margin, where again some bits may be inside and some outside the window area. We illustrate this with a diagram that shows how the code operates on the right-hand byte when scrolling left. The section of HLNSCR beginning at NEXT scrolls the bytes inside the window area. The bit that was moved out of the previous byte into the carry flag has been saved on