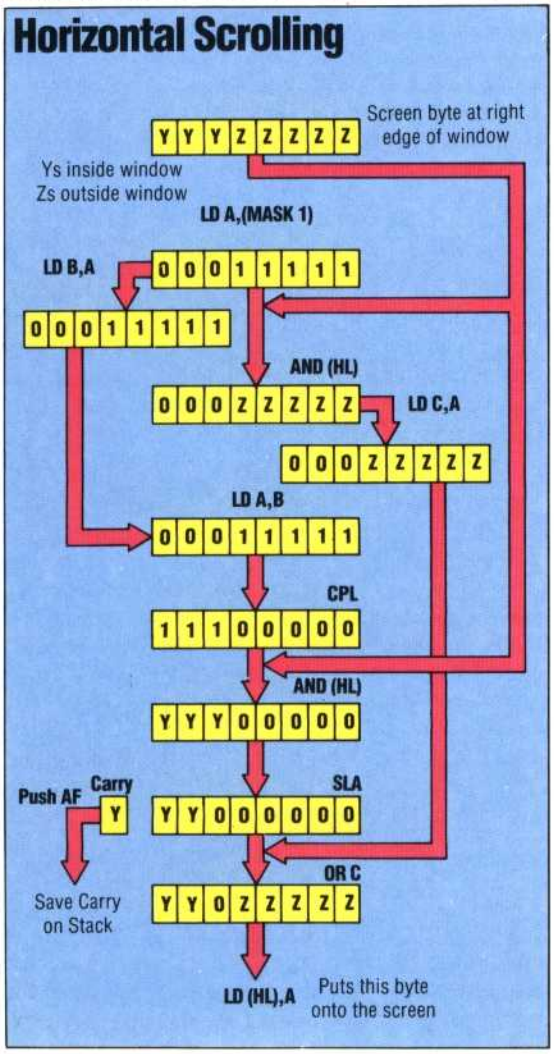
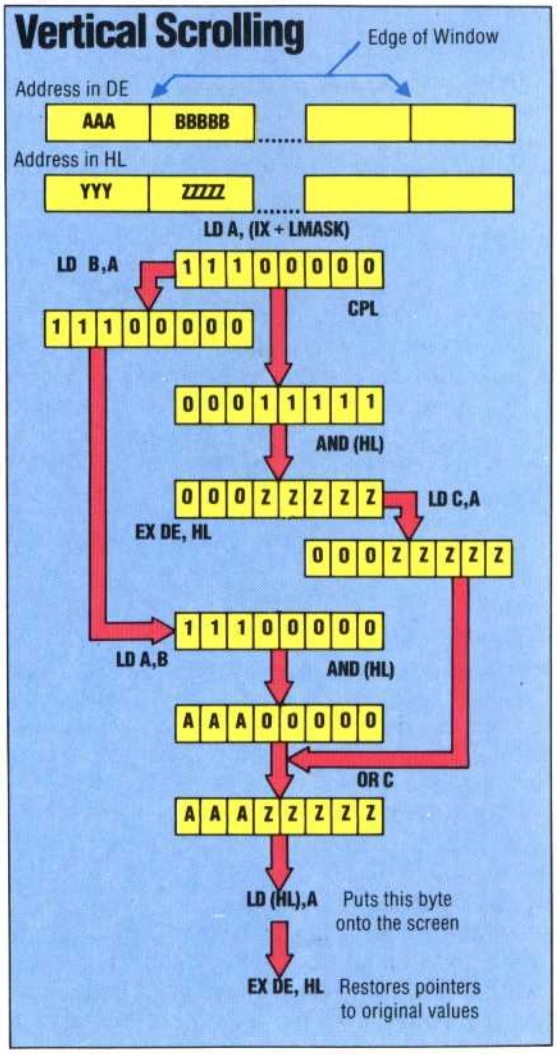
the stack by a PUSH AF instruction, and is restored to the carry flag by POP AF. For a leftward scroll the routine selects the RL (HL) instruction and the byte on the screen is shifted left, with the bit from the carry flag being moved into the left-hand end of the byte and the right-hand bit going into the carry flag. PUSH AF saves the carry flag, so this bit can be moved into the next byte. To test for the end of the row the routine simply compares the L and E registers; this is because the high byte of a screen address will be the same for all screen bytes in the same row. Partial scrolling of the last byte is achieved in a similar way to that of the first byte.

Up and down scrolling routines are combined in a similar way. If we examine what happens when VERT is scrolling upwards, we see that the routine begins by copying the y co-ordinate for the top row into temporary storage for the current row. The screen addresses of both left and right-hand margin bytes in the row are then calculated and the length of the row determined. The routine puts the address of the left-hand margin byte into DE and the address of the corresponding byte of the row immediately below into HL before calling the subroutine VLNSCR to do the scrolling. VERT then tests to see if it has reached the bottom of the window. If it hasn't, it moves down one line before jumping back to VERT5 to scroll another row of pixels. If the bottom has been reached, the section of the routine beginning at CLREDG fills the bottom pixel row with zeros to blank the row on the screen.

VLNSCR treats margin bytes separately, in a similar manner to HLNSCR. The way the code handles these edge bytes is illustrated in our second diagram. To move the central section of the pixel row, the routine increments HL and DE so that they point to the first interior byte on the current line and the corresponding byte on the line above. VLNSCR then calculates the length of the central section (the bytes that fall wholly within the window area), loads this information into BC and uses the block move instruction LDIR to move the whole central section of this row of the window up one line.

These scrolling routines are relatively slow. This is partly because left and right and up and down routines are combined and the program must make frequent tests to decide which piece of code must be used, and partly because the bytes at the left and right-hand margins require special treatment if they straddle the window's borders. The scrolling could be made faster if the program were rewritten to give a separate routine for each direction of scroll, and if the window were restricted to starting and ending at the boundary of a byte of screen memory.





Down The Line

Horizontal scrolling presents particular problems at the window edge: the screen byte must be masked to isolate the pixel bits inside and outside the window, and the byte contents must be shifted. Both processes employ logical AND, OR and SHIFT, and the stack is used to save the PSR status.

Vertical scrolling is made simpler by the Spectrum's screen memory map (see page 358). The screen byte must be masked to isolate pixels inside and outside the window, and the EX instruction is used to swap the contents of the DE and HL registers, which contain screen address pointers