DIRECTING THE ACTION

Good arcade games need to be written, at least in part, in machine code. This is a challenge for the beginner, so here we present a machine code sprite routine for the Spectrum. It can be used in two ways — either incorporated in a BASIC program by those who don't understand machine code, or used as a starting point by those who do.

Spectrum BASIC has many limitations, and these are especially noticeable when moving graphics are needed. Action games require the use of sprites of various shapes and sizes; these should be capable of smooth movement in all directions.

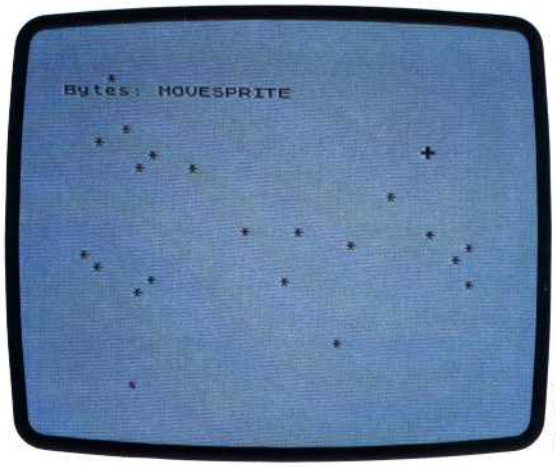
It is not easy to write graphics routines in Assembly language, but the program presented here should give you some good ideas on how to approach the task. The program prints a background of randomly placed asterisks, and allows you to move a user-defined shape (our example uses a cross) around the screen by pressing the unshifted cursor keys. The cross moves in smooth one-pixel steps, up, down, right or left, and leaves the background unchanged. The sprite-moving routine may easily be incorporated into your BASIC programs.

To instal the program in your Spectrum, you should first type in the BASIC program. The machine code may then be entered, either by typing in the BASIC loader program, which reads the code from data statements and then POKEs it into memory, or by entering the Assembly language source code by way of a suitable assembler. Both BASIC and machine code may be saved on cassette with lines 9000 and 9010 of the BASIC program.

To understand how the program works we will start by looking at the BASIC program. The subroutine at line 1000 reads the definition of the sprite from the data statements and POKEs this into RAM where it can be used by the machine code program. Lines 90 to 110 print the background, and line 120 sets the starting position for the cross. PRINT AT 10,16 makes the BASIC interpreter calculate the screen address corresponding to these character co-ordinates, and this address is stored in the system variable DFCC (addresses 23684 and 23685) where it can be read by the machine code program. Line 130 calls the initialisation section of the machine code program. Lines 140 to 180 are a loop that waits for a key to be pressed, POKEs the key value into a memory location for the machine code to read, and then calls the machine code to move the sprite one pixel in the direction indicated by the key.

The Assembly language program begins by defining names for the memory locations used. KEY is where the key value is stored. SPRPOS is used to hold the memory address of the screen position at which the sprite will appear. SPRTAB is a table in which the program stores the definition of the sprite and the contents of the screen locations that have been overwritten by the sprite. The sprite can be moved anywhere on the screen, not merely in jumps of whole character squares. So the eight bits in each row of the sprite may be divided between two bytes of screen memory, and the table uses two bytes to store the eight bits, split in the same way as on the screen. The memory location BITPOS is used to store the number of bits that the sprite data has been moved from the start of the byte.

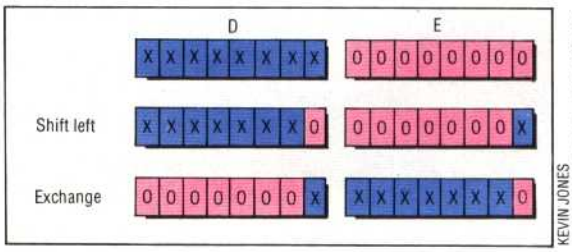
The initialisation section of the program reads the starting screen address of the sprite from DFCC, then jumps to the section labelled SAVSCR, where it stores the screen address in SPRPOS, loads the value 1 into the D register, and calls the subroutine UNDER. When D is set to 1, UNDER copies the contents of the screen area in which the sprite will be printed so that the background may be restored once the sprite has moved on. The program then calls the subroutine PRSPRT to print the sprite on the screen.



LIZ HEANEY

Sprite In Motion
The BASIC demonstration program moves the sprite (initialised as a cross in the DATA statements) across a background of stars in response to the unshifted cursor keys

The section of the program that handles the sprite movement starts at the label MOVSPR. It begins by loading 0 into the D register and calling



KEVIN JONES

Shift And Exchange
Shifting each of the sprite bytes (represented here by the Xs), in the DE register to effect left or right screen movement may cause a bit to 'wrap around'. If this happens, D and E are exchanged, re-uniting the sprite bits