ROUTINE SHAPE-UP

Sprite graphics assist in creating fast-action arcade-style games in BASIC. Machines like the Commodore 64 have chips dedicated to controlling sprite shapes; other machines do not possess this hardware but are capable of supporting sprites using software. We look at a routine to create and move a sprite on the BBC Micro.

Sprites can take many shapes and forms, but all must be designed on a common grid. There is no restriction on the choice of grid dimensions, but it is most sensible to make the grid a whole number of bytes wide (i.e. 8, 16, 24 bits etc.) and to choose a depth that gives a broad rectangular or square workspace. The routine we give uses a grid 24 pixels wide by 21 pixels deep. 63 bytes of memory will therefore be used to hold the sprite's shape. The shape is defined by designing a shape on the grid and then coding that design into binary. Here, each pixel that we wish to have turned on in the final shape is coded as a one and each pixel that is to be turned off is coded as a zero. Once the bytes making up the shape have been defined as binary values, they must then be converted to either decimal or hex, and placed in an area of memory. The design of the demonstration sprite is shown.

The 63 numbers that define the sprite can be entered as DATA statements and READ by the BASIC part of the program. As each number is READ in, it must be placed in an area of memory specially set aside for the purpose. This area can be anywhere in RAM as long as it is protected so that it cannot be overwritten during a program run. The most obvious place to store the sprite data is at the top of the BASIC program area. The top of this area is defined by the variable HIMEM. So that our data is not overwritten, it is first necessary to lower HIMEM a little. Lines 220 and 230 of the program do this and set the address of the first byte of data, SPRDAT, to start just above the new value of HIMEM. Lines 1740 to 1770 read the data and place it in the 63 bytes starting at location SPRDAT.

THE MACHINE CODE ROUTINE

The machine code's main function is to analyse the sprite design and then to carry out the operations necessary to convert the data into a screen display. To do this, the machine code routine must look at each bit of the 63 bytes of data in turn and for each bit decide whether to plot a point (if the bit is one) or leave a space (if the bit is zero). Probably the easiest way of analysing each bit of a particular byte is to use one of the Rotate instructions. Line 1100 of the source code uses the ROtate Left (ROL) instructions on a particular byte. This instruction causes each bit in the byte to move one place to the left. The old value of the carry flag is inserted at the right-hand end, and the bit that 'falls off' the left-hand end is shifted into the carry. By repeated ROLs, the routine can examine each bit in turn, whilst it is in the carry. On any subsequent ROL, the bit will be returned to the right-hand end of the byte. As long as we are careful not to alter the carry flag between ROLs, then the byte will have returned to its original value after nine rotations.

Original Contents	C	1 1 0 1 1 1 0 0
After 1st ROL	1	1 0 1 1 1 0 0 C
After 2nd ROL	1	0 1 1 1 0 0 C 1
After 3rd ROL	0	1 1 1 0 0 C 1 1
After 4th ROL	1	1 1 0 0 C 1 1 0
After 5th ROL	1	1 0 0 C 1 1 0 1
After 6th ROL	1	0 0 C 1 1 0 1 1
After 7th ROL	0	0 C 1 1 0 1 1 1
After 8th ROL	0	C 1 1 0 1 1 1 0
After 9th ROL	C	1 1 0 1 1 1 0 0

You can see from this demonstration that the original contents of the carry flag (C) do not particularly concern us, as this is rotated through the byte and eventually shifted out of the byte

Sizing The Sprite
The sprite measures 24 × 21 pixels, and so maps onto 63 bytes, one pixel per bit in the usual hi-res method. The colour of the whole sprite is set by the variable, logcol, while the sprite size is determined by the two scale factors, XSCALE and YSCALE; these should be even numbers