back into the carry flag. Notice that the ROL in line 1100 is used in the absolute indexed addressing mode so that each of the 63 sprite bytes in turn is accessed and analysed in this way.

Once the routine has isolated a particular bit and decided whether to plot a point or a space, the job of actually plotting to the screen has to be done. There are two methods of doing this on the BBC Micro. The first method is to POKE values directly into the area of memory that controls the screen display. On the BBC, this is not as easy as it sounds. Two main problems occur. Firstly, the screen memory areas vary between model A and model B machines, and between modes. Secondly, the mathematical relationship between pixels and bits in screen memory is very complex. For example, in mode 2 each byte of memory controls only two pixels on the screen. As this mode supports 16 colours, each pixel needs four bits to define its colour. In mode 0, however, a two-colour mode, each pixel requires only one bit for definition. Designing a routine to code our sprite format into mode 2 format would be feasible, but the routine would then only work in mode 2.

Fortunately, there is an alternative method. When dealing with graphics in BASIC, the BBC operating system has to do the sort of manipulations that we require. We can access this operating system routine to do most of this difficult work for us. In addition, a routine written using this operating system call will work in all the graphics modes. The routine works in the same way as a VDU command in BBC BASIC. For example, to plot a single point on the screen, the following VDU command could be used:

```
VDU25,68,300;700;
```

Here, the x and y co-ordinates are specified by the numbers 300 and 700 respectively. This VDU command can be duplicated in machine code by using the operating system call OSWRCH. This call is made repeatedly after first placing a number in the accumulator. Because the accumulator can only hold one byte at a time, the x and y co-ordinates must be split into LO-byte/HI-byte form as follows:

```
VDU25,68,44,1,188,2
```

To perform this command in machine code, OSWRCH needs to be called six times. The vector to the start address of OSWRCH is held in location &FFEE, and the routine is accessed by JSR &FFEE. Any VDU command can be done in this way, and this routine uses OSWRCH calls in several places. Note that whereas OSWRCH does not affect the values of the X, Y and A registers, it will alter the contents of the carry flag. Hence, when preservation of the carry is required, the contents of the carry must be stored away before OSWRCH is called. This is the case with the ROL routine. The easiest method of preserving the carry is to push the processor status register onto the stack (PHP) before calling OSWRCH, and pull it off again afterwards (PLP).

Let's now look at the structure of the machine code routine. The main analysis of sprite data and plotting to the screen are performed using the subroutine SPRPLT, starting at line 890 in the source code. The first operation of this routine is to set the method of plotting as an Exclusive OR operation. This is entirely similar to the GCOL command in BBC BASIC. Points plotted in this way can be erased by simply replotting over the top. Any screen data under the sprite will therefore be left intact. At the start of the machine code an absolute move is made to position the top left hand corner of the sprite. Each row is then analysed by taking three bytes of sprite data and rotating them as described earlier.

A relative plot or relative move is made over a distance determined by a horizontal scaling factor, XSCALE, depending on the value of the data bit currently in the carry. At the end of each row of three bytes an absolute move is again made, to the same x co-ordinate as the sprite's top left corner, but to a reduced y co-ordinate determined by a vertical scaling factor, YSCALE. The process is repeated until all 63 bytes have been analysed.

The SPRPLT subroutine is used in two places. Firstly, to erase the old sprite by replotting over it, and secondly to plot the new sprite. After plotting the new sprite, its co-ordinates are transferred to OLDX and OLDY in preparation for the next time the routine is used.

Using The Machine Code Routine From BASIC

The routine can be easily used by BASIC programmers without needing to understand the routine itself. The steps that must be carried out are as follows:

1) Design your sprite and place the data in an area of memory as shown in the program
2) Set the display mode you wish to use
3) Set the values of XSCALE and YSCALE as shown in line 1870
4) Set the logical colour for the sprite as shown in line 1890
5) Set the x and y co-ordinates of the position you wish the sprite to appear in and use the procedure given at lines 2010 to 2060 to convert absolute co-ordinates into LO-byte/HI-byte form.
6) CALL SPRITE

The machine code routine can be incorporated into your BASIC listing, as here. The assembler listing can be suppressed by changing line 260 to:

```
FOR opt%=0TO2STEP2
```

Alternatively, you can save the routine once assembled (i.e. after a run) using the *SAVE command, taking careful note of the start and end addresses of the code given as the assembly listing is displayed.

Erratum

The program segments given in the boxes at the bottom of page 295 for Inserting and Deleting a record in a BASIC array were inadvertently transposed. Lines 100-190 are used for inserting a record, and lines 200-250 for deleting a record from the array. The program lines and captions are otherwise correct