DEFINING TERMS

The Commodore graphics set is extensive, but it is often necessary to create some special characters, or even to redefine the entire character set. In this instalment of our graphics series we introduce the techniques of user-defined graphics on the Commodore 64 and continue to develop the Subhunter game.

The process of creating your own characters on the Commodore 64 is not straightforward: there are no special-purpose commands in Commodore BASIC, so the whole operation has to be carried out using PEEK and POKE to access and change the contents of memory.

The Commodore 64 character set consists of a block of ROM starting at memory location 53248. Each character appears on the screen as a pattern of dots in an eight by eight dot matrix: describing this pattern of 64 dots requires 64 bits, or eight bytes. The eight bytes from location 53248 to 53255 describe the '@' character, the first character in the set; it has a screen code of 0, which means that if you POKE the value zero into one of the bytes of video RAM, this character will appear on the screen. The next eight bytes, from 53256 to 53263 describe 'A' (screen code 1), and so on.

We cannot change these dot matrix definitions in ROM, so we must copy some or all of them into RAM and make the changes there. We can then make the Commodore use our RAM character set for writing on the screen, rather than using its own definitions in ROM.

The ROM character set shares its address space in memory with input/output devices such as cassette players and disk drives. Normally, the 6510A CPU treats this memory space as an input/output area, but it can be programmed to regard it as the character set location. This may seem strange, but the CPU doesn't normally do the work of accessing character definitions from ROM and sending them to the screen. That task is delegated to a subsidiary chip under CPU control. The contents of location 1 determine the status of I/O operations, and bit 2 of this location acts as a switch on the way in which the CPU regards the character set ROM. If this bit is set to zero, then the CPU finds the I/O devices occupying the space. The other bits of location 1 have similar special functions in controlling the system, so we must be careful not to alter any of them while changing bit 2's value. This is best achieved by using the logical operators AND and OR.

Suppose that the contents of location 1 are:

Bit	7	6	5	4	3	2	1	0
	0	1	1	0	1	1	1	1

We wish to change bit 2 to zero. One way to do this would be to calculate the decimal value of 01101011, and POKE it into location 1, but this works only if we know that the previous contents of location 1 were 01101111. A better way to adjust bit 2 is to use AND and PEEK. The following command PEEKs location 1, thus establishing its original contents, ANDs them with 251 (11111011 binary), and POKEs the result back into location 1:

```
POKE 1,PEEK(1) AND 251
```

The effect of this command can be illustrated here:

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	1	1	1	= Initial contents
AND	1	1	1	1	1	0	1	1	= 251 binary
	0	1	1	0	1	0	1	1	= Result of ANDing each pair of bits

No matter what the original value of bit 2, ANDing it with zero will always produce a zero result; ANDing all the other bits of the location with one simply produces a copy of their original value. The binary number 1111101 (251 decimal) is called a *mask* or overlay, and here we are using it as an 'AND-mask'.

To set bit 2 to one without affecting any of the other bits, we use the following command:

```
POKE 1,PEEK(1) OR 4
```

Bit	7	6	5	4	3	2	1	0	
	0	1	1	0	1	0	1	1	= Initial contents
OR	0	0	0	0	0	1	0	0	= 4 binary
	0	1	1	0	1	1	1	1	= Result of ORing each pair of bits

This ensures that BASIC will not overwrite our character set. When the copy is complete, the CPU can be reset to address the I/O devices, and the interrupt time re-started.

The final piece of the jigsaw is forcing the screen-handling chip to use our character set, rather than the system set in ROM. Bits 0 to 3 of location 53272 point to the start address of the character set, and the following table shows how the Commodore 64 interprets the values of these bits as pointing to particular addresses: