PROGRAMMER'S LOGIC

So far in this series on logic we have looked at the design of *hardware* using logic gates, but the logical operators AND and OR are also useful *software* tools within programs. Most BASIC dialects and machine code instruction sets include AND and OR in their commands.

These two logical operators have several uses in both machine code and BASIC. The most familiar use of AND and OR is to relate two or more statements within a conditional statement. For example, try predicting the outcome of this BASIC program:

```
10 FOR I=1 TO 5
20 FOR J=1 TO 5
30 IF I=3 AND J=2 THEN PRINT I,J
40 NEXT J
50 NEXT I
60 END
```

The program will run through the pair of nested loops but will print out the values of I and J only if I=3 and J=2. This program will therefore print on the screen the following result:

```
3 2
```

OR can be used in much the same way. If we amend line 30 to read:

```
30 IF I=3 AND J=2 OR J=4 THEN PRINT I,J
```

The following output is produced:

```
1 4
2 4
3 2
3 4
4 4
5 4
```

The computer carries out the AND operation in priority to the OR operation. I and J will be printed out if either I=3 and J=2, or if J=4. The order of priority can be changed by the use of brackets. What will be the output from the program if line 30 is again amended to:

```
30 IF I=3 AND (J=2 OR J=4) THEN PRINT I,J
```

NOSE-OPERATED KEYBOARD

The MATE (Memory Assisted Terminal Equipment) system was developed at Essex University, England, in 1978 on a Vector 111 computer. This system was developed by a severely spastic programmer, Geoff Busby, who could manipulate the keyboard only with his nose.

MATE has a special keyboard that does not require a shift key to be held down at the same time as another key for some characters, and stores a database of words that can be input by a single keystroke without typing all characters

ISOLATING BITS IN A REGISTER

Many home computers use special registers to control various machine functions. Each bit within such a register may control a different aspect of that operation. For example, on the Commodore 64 there is an eight-bit register that controls the switching on and off of sprites. Each bit in the register relates to one of the eight sprites available. If any bit in the register is set to a one then the sprite that it controls is visible on the screen. If the bit is set to a zero then the sprite is switched off and cannot be seen. Using BASIC it is a simple matter to switch on any combination of sprites by working out the required eight-bit binary number and POKEing its decimal equivalent into the register. This method, however, doesn't take account of the state of the register prior to the POKE command and may result in switching off sprites that were previously on. The solution to this problem is to develop a technique that will allow the programmer to isolate the bits that are needed to be changed without altering any of the others.

In order to demonstrate this technique let us assume that originally sprites 0, 1, 5 and 6 are turned on. The register controlling the switching will look like this:

Sprite Number	7	6	5	4	3	2	1	0
Corresponding Bit	0	1	1	0	0	0	1	1

Let us now switch on sprite 4, by PEEKing the register, ORing the contents with 16 (00010000 in binary) and POKEing the result back.

Original Byte	0	1	1	0	0	0	1	1
ORed With	0	0	0	1	0	0	0	0
Gives New Byte	0	1	1	1	0	0	1	1

Using the BASIC command POKE reg,PEEK(reg) OR16 we can now turn on bit 4 in the register. To turn bit 4 off again we must PEEK the register and AND its contents with 239.

New Byte	0	1	1	1	0	0	1	1
AND	1	1	1	0	1	1	1	1
Original Byte	0	1	1	0	0	0	1	1

Notice that the number 239 can be quickly calculated by subtracting 16 from 255. Using the BASIC command POKE reg,PEEK(reg)AND239 we have restored the register to its original state.

These techniques are more widely applied in machine code programming where altering the state of control registers may form an important part of the program.