ACTION STATIONS

Our introduction to the BBC operating system concludes with this instalment. We take a detailed look at the use of vectors, and investigate how the OS enables us to interact with the computer via the keyboard and VDU.

The majority of BBC OS routines are said to be *vectored* (see page 878). The OS, on being told to call the OSCLI routine, first calls a routine at address &FFF7. This routine then calls the main OSCLI routine, but not directly. It finds out the address at which the OSCLI routine is to be found by inspecting the contents of two bytes of memory in page 2 of the RAM. These two bytes are called a *vector:* the low byte of the address of the routine concerned is found in the lower numbered byte of the vector, and the high byte of the address is to be found in the higher numbered byte of the vector. Thus, for OSCLI, which is vectored through locations &208 and &209, the low byte of the OSCLI addresses is held in location &208 and the high byte is held in location &209. This is known as the Lo-Hi addressing convention and is followed for all stored addresses in all 6502 machines. The addresses held in each vector are set up by the OS whenever the machine is reset. Why bother with such a complicated way of calling a routine in the OS? It's not because Acorn are determined to make the life of a programmer as miserable as possible. On the contrary, this process is designed to make life easy! How can this be?

You may have noticed that all BBC OS routines mentioned so far are called at an address in the range &FF00 to &FFFF. This is no accident. When such an address is called, a routine is entered that causes a jump to the address held in the vector for that particular OS routine, as we've seen in the case of the CLI and the OSCLI call. Now, the address that we call between &FF00 and &FFFF is the same in *all* BBC OS versions and will continue to be in all versions to come. If it becomes necessary to change the OS ROM internal programs then the OS designers simply ensure that the addresses of the ROM routines that are put in the vector locations are altered to take the changes into account. The user is thus protected from such changes in the OS provided that the OS routines are called at the correct entry points. The contents of a vector may therefore differ in different versions of the OS, but you won't notice this as long as you use the entry point addresses in the range &FF00 to &FFFF.

A second advantage of the use of vectors is that this method provides us with a means of modifying the behaviour of the OS routines. We can simply alter the contents of a vector so that it points to a machine code routine of our own devising if we so desire, thus intercepting the normal OS calls. In later parts of the course, we'll look at the vectors that are used with each of the major OS routines.

For the moment, let's consider the vector called USERV, which is pointed to at locations &200 and &201. This is a rather special vector, in that it normally does nothing. It is used by two * commands, called *CODE and *LINE. If you type these in normally then the message Bad Command is issued. Before sending off a letter of complaint about a new BBC OS bug, read on!

USERV enables us to define the function carried out by the *CODE and *LINE commands – user defined commands, if you like! Why should we want to do this? Well, *CODE is a particularly useful way of passing parameters into machine code programs, as shown in the following table:

Register	*CODE x, y	*LINE Text String
A	0	1
X	Holds the value of the first parameter after *CODE. Parameter must therefore be between 0 and 255	Holds the low byte of the address in memory at which you can find the first character of the string after *LINE
Y	Holds the value of the second parameter after the *CODE command. Again the parameter must be between 0 and 255	Holds the high byte of the address in memory at which you can find the first character of the string after *LINE

This table shows the state of the three CPU registers on entering the routine pointed to by the contents of USERV. A holds either zero or one, and thus indicates which of the two commands caused USERV to be entered. X and Y hold values depending on whether it was a *CODE or a *LINE command. Thus, *CODE 3, 2 will enter the routine pointed to by USERV with 0 in A, 3 in X and 2 in Y. Obviously, the routine pointed to by the address held in USERV will be the routine that we want to pass the two parameters to.

The program given on the following page shows a simple *CODE command in action. The machine code routine itself is assembled into memory starting at address &C00 as a result of the assignment statement in line 40 – the integer variable P% 'maps onto' the processor program counter, just as A%, X% and Y% map onto the A, X and Y processor registers. USERV is set up to point to the routine by putting the low byte of this