THE FX EFFECT

Having introduced the BBC Micro's operating system in recent instalments, we return to the subject with a closer examination of OSBYTE calls, a convenient way of accessing many of the OS functions of this computer.

When we first considered how to access the operating system of the BBC Micro (see page 879), we briefly discussed a group of OS calls known as OSBYTE calls. These enable us to modify the behaviour of various parts of the operating system. For example, the OSBYTE call *FX4,1 enables us to change the way in which the computer responds when one of the cursor keys on the BBC keyboard is pressed.

When you realise that there are well over 100 of these calls in the Version 1.2 OS (see page 858), it's not surprising that they offer us a convenient way of accessing many of the OS functions of the BBC Micro. We've already seen that the use of indirect OS calls provides us with insurance against changes in the hardware and software configuration of the machine; OSBYTE provides us with a major method of using OS routines.

Before we go on to examine OSBYTE in detail, it should be pointed out that if your machine has an old Version 0.1 OS, some of the OSBYTE calls mentioned in this course and in the BBC user guide are not actually supported. You can find out which version your machine has by typing *HELP ‹RETURN›.

Let's look first at how OSBYTE is used from BASIC and machine code. Like many BBC OS calls, OSBYTE is vectored (see page 878). The OSBYTE vector is at addresses &20A and &20B. The ways in which we can issue an OSBYTE call are shown below. These will all execute the OSBYTE call mentioned above — *FX4,1:

Using *FX From BASIC	Using USR From BASIC	Using Machine Code
*FX4,1	A%=4:X%=1:D%=USR(&FFF4)	LDA #4 LDX #1 JSR &FFF4

It's clear from these three examples that the address at which we call the OSBYTE routines is &FFF4, and that parameters are passed over to the OSBYTE call in the A, X and Y registers of the 6502 processor. Assigning a value to A%, X% or Y% from BASIC and then calling a machine code routine with the USR or CALL command from BASIC will enter the machine code program called with the A, X and Y registers of the processor holding the values that were in the A%, X% and Y% variables, respectively. The use of USR in the second example enables us to get a result from a machine code program back into a BASIC variable. This is useful with regard to some OSBYTE calls, as they can return information to BASIC — we will discuss this in detail later.

In the third example, we use a short machine code program to make the OSBYTE call; we simply load the necessary registers with the appropriate values and then call OSBYTE at its call address of &FFF4. These examples also show how the parameters (4 and 1) passed to the OS with an FX call, correspond to the parameters passed in the A, X and Y registers when we call OSBYTE routines from machine code or with a USR call.

No matter which method of calling the OSBYTE routine we choose, the contents of the A register always specify which of the many OSBYTE routines is to be used. X and Y registers are then used to pass over parameters to the desired OSBYTE routine. Some OSBYTE calls require no parameters; some require just one parameter to be passed over in the X register; and others, less commonly, require two parameters, passed over in the X and Y registers.

Here are some examples of all these types of OSBYTE calls:

No Parameters	One Parameter	Two Parameters
*FX0	*FX4,1	*FX151,96,200

There are a couple of points to note about calling OSBYTE routines via the use of the *FX call. One is that OSBYTE is totally ignorant of BASIC variables — like all * commands. Executing the code below will give the Bad Command error message:

```
a=4:b=1
*FX a,b
```

However, we can get round this problem by passing the FX command to the OS using OSCLI:

```
10 DIM C 100
20 a=4
30 b=1
40 $C="*FX "+STR$a+","+STR$b
50 X%=C MOD 256
60 Y%=C DIV 256
70 CALL &FFF7
```

The other point to note about FX calls is that you cannot put anything else on a program line after them. Thus:

```
*FX 4,1:PRINT "Oooops!"
```

will generate another Bad Command — the OS