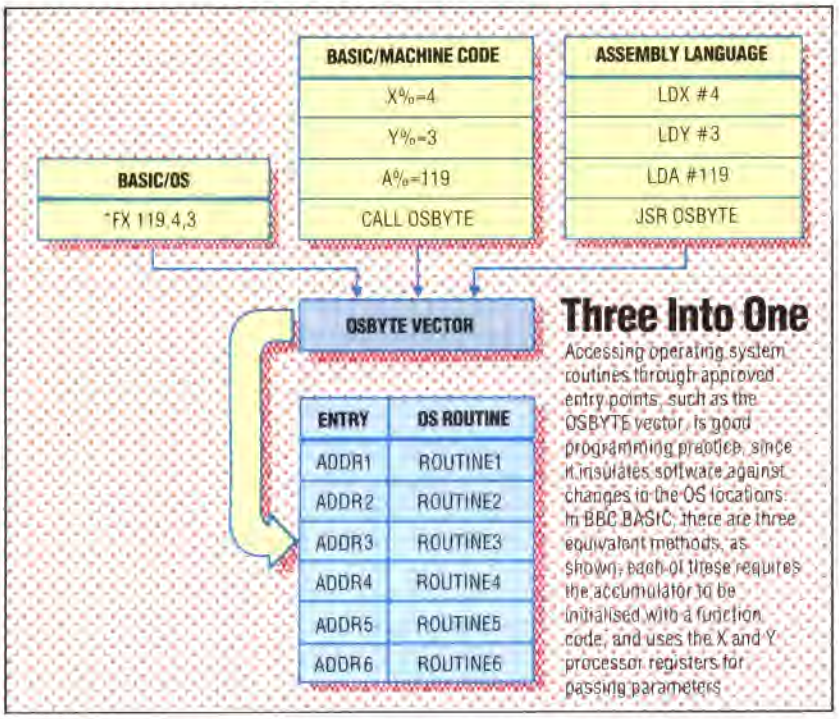

regards the colon and the PRINT statement as part of the FX command.

Both problems are caused by the fact that the *FX calls are passed through the command line interpreter (CLI) rather than the BASIC interpreter, and the CLI has no 'knowledge' of how to evaluate BASIC variables and deal with multi-statement lines.

As we have shown, it is possible to pass parameters over to OSBYTE in the X and Y registers; it is also possible to read values back from some of the system variables used by the operating system. This can be done by using the USR call or the machine code routine — as we've already shown. You'll probably find the machine code method easier to use when you're interested in getting results back from the OS — the value returned by the USR call has to be decoded to get various bits of information out of it. Parameters are passed back to BASIC in the X and Y registers, and in some of the calls the carry flag is also used to signal error conditions.

The results passed back in this way will obviously depend upon the call — that is, on the value passed to OSBYTE in the A register. Not all OSBYTE calls pass results back to BASIC. However, many of those that do provide us with some useful information about the OS.

Two kinds of information may be passed back by an OSBYTE call. The first is data read from some part of the system, such as the user port, speech processor or system variables. OSBYTE calls passing back this sort of information are referred to as 'read only' calls. A typical example of their use is the OSBYTE call with A=129. This call is used by BASIC to implement the INKEY() function.

The X and Y registers should be set up to pass the required time delay over to the operating system. The X register holds the low byte of the time delay — in centiseconds — and the Y register holds the high byte. Thus, to use this call to wait for up to one second for a keypress, we can use the section of machine code shown below. The X and Y registers pass values back; if the carry flag is set to 0, and the Y register holds a value of 0, then the call was exited by a keypress. The ASCII value of the key thus returned is to be found in the X register. If Y holds 255 and C is set to 1, then no key was pressed in the time period allowed. If C is set to 1 and Y holds 27, this indicates that the Escape key has been pressed.

The following section of code shows how this OSBYTE call can be made. If the carry flag is set on return from the subroutine a branch is made to a further handling routine. This routine may test the value in the Y register to determine whether a key has been pressed or the Escape key has been hit.

```
1000    LDA     #129    /set OSBYTE
1010    LDX     #100    /parameters
1020    LDY     #0
1030    JSR     &FFF4   /make OSBYTE call
1040    BCS     error
1050    RTS
1060    .error          /code to deal with error
```

Other OSBYTE calls, especially those with a value in A of between 166 and 255, are both read and write calls, and they enable us to either read or write certain system variables in the OS. You may begin to wonder how the OSBYTE call knows whether a read or write operation is required; it's actually quite simple.

To write a value with the OSBYTE call, the call is made with the X register holding the value we want the OSBYTE call to write and the Y register set to 0. To read a value back from one of these systems variables, X is set to 0 and Y is set to 225. The call is then made. If a value is returned, it resides in the X and Y registers.

THE USES OF OSBYTE CALLS

OSBYTE calls are the 'Civil Servants' of the operating system, being involved in many of the different OS routines. Filing systems, the keyboard, Econet, the Break and Escape keys — all are affected to a greater or lesser extent by OSBYTE calls. The number of different OSBYTE calls available makes it impossible to discuss them all, but here are a few of the more useful ones not covered in detail in the BBC Micro's user guide.

Function keys: *FX18 has no parameters, but is quite useful. It deletes from memory the current function key definitions, so is handy when you want to define a function key more than once in a program.

*FX225 to *FX228: If you've not programmed a function key with a string, you can use these calls to make the red function keys return an ASCII value. For example, *FX225,n will cause function key f0 to return the ASCII code for n when pressed, f1 will return ASCII code (n+1), and so on. *FX226 does the same job for the occasions