ASSEMBLY LINES

In this part of the Machine Code course, we summarise the main conventions used when dealing with memory, particularly: lo-hi addressing, tokenisation, and the importance of context. We also introduce some differences between Assembly language programs for the 6502 and Z80 microprocessors.

When you RUN a program, the first thing that the Operating System does is to inspect the Start of BASIC Text pointers in order to determine where in memory the program to be executed resides. But to do this, the Operating System has to store the pointers' addresses, so why doesn't the OS simply store the addresses to which they point?

The main reason is flexibility. The Operating System, you will remember, is a permanent program resident in the ROM, and any data that it contains (such as memory addresses) is similarly permanent. Suppose that different versions of the computer are released over a period of time, and that, although it was convenient to have the Start of BASIC Text at byte2048 in Version 1, it becomes necessary to relocate it at byte4096 in Version 2. This will mean that the later machine will not be able to use the Operating System of the earlier version because of the different locations of the BASIC Text Area. Furthermore, new ROMs would have to be developed for each new version of the machine, which is expensive; and software written for one version may not be able to be run on the other. If, however, the Operating System ROMs contain only the pointer addresses, then the same ones can be used for all versions of the machine and only the pointer contents need be changed from model to model. The location of the pointers themselves can remain constant because the Operating System requires a relatively small block of memory for workspace and data storage (typically about 1,000 bytes). Fixing the position of this block — usually the first four pages of memory — and designing or re-designing the system around it does not greatly constrain the design team. On the other hand, having the location of, say, the BASIC Text Area fixed (a block of 3,000 to 40,000 bytes) is a severe restriction.

STANDARD PRACTICE

It is conventional to store addresses in pointers in what is called *lo-hi* form. If byte43 and byte44, for example, are to point to the address 7671 (page 29, offset 247), then byte43 will contain 247 (the offset or lo byte of the address), while byte44 will contain 29 (the page or hi byte of the address). This may seem confusing but it is convenient for the microprocessor. It is also logical in that the lo byte of the address is stored in the lo byte of the pointer, and correspondingly the address hi byte in the pointer hi byte.

If we repeat the above example using hex rather than decimal numbers, the great advantage of the hexadecimal system can be seen (from now on addresses and other numbers will always be written in hex prefixed by '$'). The pointer bytes are $2B and $2C, and the address to which they point is $1DF7. Therefore, $2B contains F7 (the address lo byte), while $2C contains $1D (the address hi byte). Notice that when the address is in hex the rightmost two hex digits are the lo byte, and the leftmost two digits are the hi byte, which makes much better sense than using decimal numbers.

It's worth remarking that the BBC and Spectrum are unusual in storing program line numbers as two-byte numbers in hi-lo rather than lo-hi form. It's true that these are program parameters rather than byte addresses, but they work against the usual convention, nonetheless.

Another convention of memory addressing is that pointers, although they are two-byte quantities, are often referred to by the address of the lo byte alone. We might say, for example, that byte43 in the Commodore 64 points to the Start of BASIC Text. It is understood here, however, that byte43 and byte44 together are the pointers.

Other things to consider include tokens (see page 76). The significance of these for machine code programmers is two-fold: they represent multi-character English commands (such as PRINT or RESTORE) by single-byte numerical codes; and they use offsets as well. A BASIC command is one word, but executing it is not a single operation for the Operating System. The command PRINT, for example, requires that the data to be printed be found in memory or evaluated, and then sent character-by-character to the screen in ASCII code. These various tasks are carried out by a subroutine of the BASIC interpreter program. When the interpreter encounters the PRINT token in a program line it uses the value of that token to locate and then execute the corresponding subroutine.

Suppose there are only three commands in our version of BASIC: INPUT, PRINT, and STOP; and these are assigned the tokens: $80, $81, and $82 respectively. Furthermore, let's say that the interpreter subroutines that execute these commands start at bytes $D010, $EA97, and $EC00 respectively, and that these three addresses are stored in lo-hi form in the six bytes from $FA00 to