MODES OF ADDRESS

The strength and versatility of Assembly language instructions is enhanced by the variety of ways in which these instructions can address memory. The different ways consist of direct, indirect and indexed addressing, and combinations of these. In this instalment we take a closer look at modes of addressing in machine code.

Every Assembly language instruction refers explicitly or implicitly to the contents of memory, and since one byte is distinguishable from another only by its address, every Assembly language instruction must, therefore, refer to at least one address. The manner in which an address is referred to may be direct and obvious, as in LDA $E349, which means 'load the accumulator with the contents of the address $E349'. In this case, both the accumulator (a byte with a name rather than a number for its address) and the address $E349 are mentioned unambiguously, and the nature of the relationship between them is clear.

On the other hand, the reference to an address may be much less obvious: RET, meaning 'return from a subroutine call', is a good example. This may not seem to refer to an address at all until you expand it into 'the location address of the next instruction to be executed is the place where the last subroutine call was made'. Here, the address whose contents are to be changed (i.e. the program counter — the register holding the address of the next instruction to be executed) is not mentioned, nor is the address at which its new contents (i.e. the new location address) are to be found. These two instructions can be seen as highly contrasting examples of *addressing modes*.

In the course so far we have seen instructions in two kinds of addressing mode: *immediate mode*, as in LD A,$45 or ADC #$31, and *absolute direct mode*, as in STA $58A7 or LD ($696C),A. These may seem like the 'natural' addressing modes, covering every possible case except the implicit modes such as RTS or RET, but there are other possibilities as well. We shall look at these separately.

ZERO PAGE ADDRESSING

Zero page addressing (also known as *short addressing*) is used whenever an instruction refers to an address in the range $0000 to $00FF. All addresses in this range have a hi-byte of $00, and therefore lie on page zero of memory. Such instructions need only two bytes — one byte for the op-code and one for the lo-byte of the address. When the CPU detects a single-byte address at a point where it expects there to be two bytes, it assumes a hi-byte of $00, and so refers automatically to page zero. The advantage of this page zero mode is speed of execution: data on page zero is accessed significantly faster than data on any other page precisely because it requires only a single-byte address.

The Z80 and the 6502 microprocessors differ greatly in their use of the zero page mode. In 6502 Assembly language, any instruction that addresses RAM (such as LDA) can be used in the zero page mode, and all the 256 bytes of zero page are available to the CPU. In Z80 Assembly language, only one instruction — RST (the 'restart' or 'reset' instruction) — features the zero page mode, and it can address only eight specific page zero locations. Because the RST instruction is so specific in its addressing, it requires only a single-byte op-code (the address forming part of the op-code itself), which makes it very fast in execution. We will see more of the Z80 RST instruction later in the course because of the special uses to which it is put.

It may seem ridiculous to be comparing the speed of Assembly language instructions when the execution time of the slowest instruction is measured in microseconds anyway, but it isn't difficult to write Assembly language programs in which saving a microsecond per instruction execution can mean the difference between success and failure. Games programs featuring animated high-resolution three-dimensional colour graphics, for example, can involve millions of instructions per screen operation, and they must execute these commands as quickly as possible for the sake of smooth animation. Shaving a microsecond off one operation becomes very important when you put that operation inside a loop and execute it 64,000 times consecutively.

INDEXED ADDRESSING

Indexed mode is vital to Assembly language programming since it permits the construction of array-type data structures. Without such structures, programs are restricted to addressing memory locations individually by address: this is what we have done with all the programs so far in the course. Once indexing is possible, however, far more data can be handled, and more power is put at the programmer's disposal.

The essential elements of indexed mode are a *base address* and an *index*. Suppose we wish to keep a table of data — ASCII character codes, for example — in consecutive bytes. The base address of the table is the address of its first byte.