# JUMP LEADS

**Following our examination of indexed addressing on the 6809 processor, we now consider how indirect addressing is used, and illustrate this by describing routines to write characters to a screen display.**

First of all, it should be stated that indirect addressing is not a separate addressing mode in its own right but is an additional feature that may be used in combination with most other modes; it is really a further stage in calculating the effective address (the address from which the data is actually to be fetched). The effective address is calculated in any of the ways we have described (by direct access, by indexed addressing, or by effective address calculation), but if indirection is specified then the contents of the address so calculated and the next consecutive memory location are treated as an address. It is this address that becomes the final effective address, from which data is loaded.

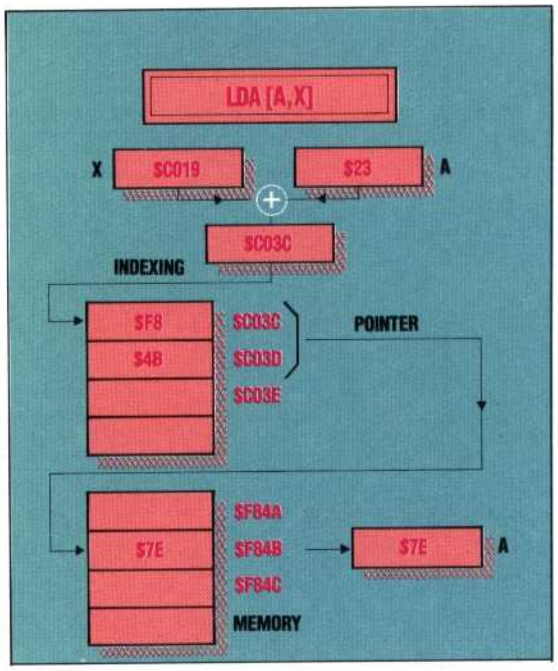
For example, if the following values are stored:

| Address | Contents |
|---|---|
| 3000 | 40 |
| 3001 | 0A |
| 400A | F2 |

then the instruction LDA $3000 will load the value $40 into accumulator A, the effective address being $3000. Indirection is always specified by placing square brackets around the operand, so LDA [$3000] will load the value $F2 into A, the effective address being the value stored in the address that is in turn stored in $3000 and $3001 — in this case, $400A. The contents of $3000 and $3001 form a pointer or vector to the effective address, $400A. Notice the 6809 convention that addresses are stored with hi-byte before lo-byte: thus $40 is stored in $3000 and $0A is stored in $3001. This is called the hi-lo convention. The Zilog Z80 and MOS Tech 6502 processors use the opposite convention — they would require $0A (the lo-byte of the address) to be stored in $3000 and $40 (the hi-byte) to be stored in $3001.

Indirection can often be most effectively used in combination with indexed addressing. The instruction LDA [A,X], which is in indirect indexed addressing mode, will calculate an address by first adding the contents of A and X and then using the 16-bit value that is stored at this and the next location as the effective address whose contents will be loaded into A.

The 6809 has, in fact, less use for indirect addressing than most processors (both 6502 and Z80 programs use it frequently) because of its wealth of indexed addressing modes. There are, however, situations in which indirection can be very useful — one of these, which we will deal with at greater length in a future instalment, is the use of peripheral interface devices. Motorola processors, unlike Intel's 8080 and 8086 families, have memory-mapped I/O (Input/Output). The communications registers in the interface devices appear in the system's main memory map, and values can be stored to or loaded from them as if they were any other memory locations instead of being, effectively, a channel to the interface device. A routine to control one of these devices — for example, a print routine — needs the address of the device's interface register. If the device is relocated in the memory map, or if there is more than one device of that type, then it is much simpler to deal with this by changing one memory location that contains the address of the device communication register (a pointer to the device) rather than having to find and change every occurrence of the device address. The routine refers to the device indirectly, using the pointer.



**Indirect Indexed Addressing**
The argument [A,X] of the LDA instruction is in brackets, meaning that the contents of X ($C019 here) are to be added to the contents of A ($23), giving a 16-bit address ($C03C). This byte and the next ($C03D) are to be treated as a pointer to the effective load address ($F84B) whose contents ($7E) are finally loaded into A. Because X is added to A before the indirect access, this is known as pre-indexed indirection; the alternative, post-indexing, requires that the indirect address be calculated before the indexing takes place

This example illustrates the general usage of indirect addressing — when addresses that a program refers to may be changed, it is more convenient to use fixed-address pointers to refer to these locations. In this way, changes in the actual locations require only changes in the pointer contents: the program refers to the addresses indirectly.

The most common use of this technique is in a structure known as a *jump table*, which is simply a table of pointers. Any operating system contains a large number of useful routines that carry out the elemental functions of the machine — for example, reading a character from the keyboard or displaying a