Assembly Line

Continuing our introduction to machine code, we look at the many different forms in which programs can be expressed — from binary to Assembly language

One of the conceptual difficulties that most newcomers experience with machine code is that the programs can take various forms. Any data stored in computer memory ultimately takes the form of eight-bit binary numbers. However, when these are listed out on paper, they occupy a lot of space, are difficult to read and remember, and are prone to typing mistakes. So instead we usually make use of hexadecimal numbers. This has the advantage that the contents of any byte can be expressed as a two-digit number, and any address in the computer's memory range (0 to 65535 in decimal) can be represented by four digits.

When we write a hex number on paper we usually precede it with a $ sign to distinguish it from a decimal number, although the sign does not feature in the computer's memory when the program has been entered. Secondly, when an opcode has a two-byte operand (e.g. LDA $3F80) the two bytes are entered into the machine in the opposite order — i.e. the low byte followed by the high byte. In the example given, therefore, the three bytes would be AD (the hex representation of the LDA opcode in 6502 language) followed by 80, followed by 3F. This makes things easier for the processor, but it can be confusing for the user.

Usually a machine code program is printed as a 'hex dump' — a long list of two-digit hexadecimal values. In addition, a starting address will be given (either in hex or decimal) and the first hex value must be loaded into this location, the second into the next location, and so on. Loading can be achieved by means of the BASIC POKE command. If the starting address is $1000 (4096 in decimal) and the hex dump is:

```
AD    (173 in decimal)
80    (128 in decimal)
3F    (63 in decimal)
```

the program can be loaded with the three BASIC statements:

```
POKE 4096,173
POKE 4097,128
POKE 4098,63
```

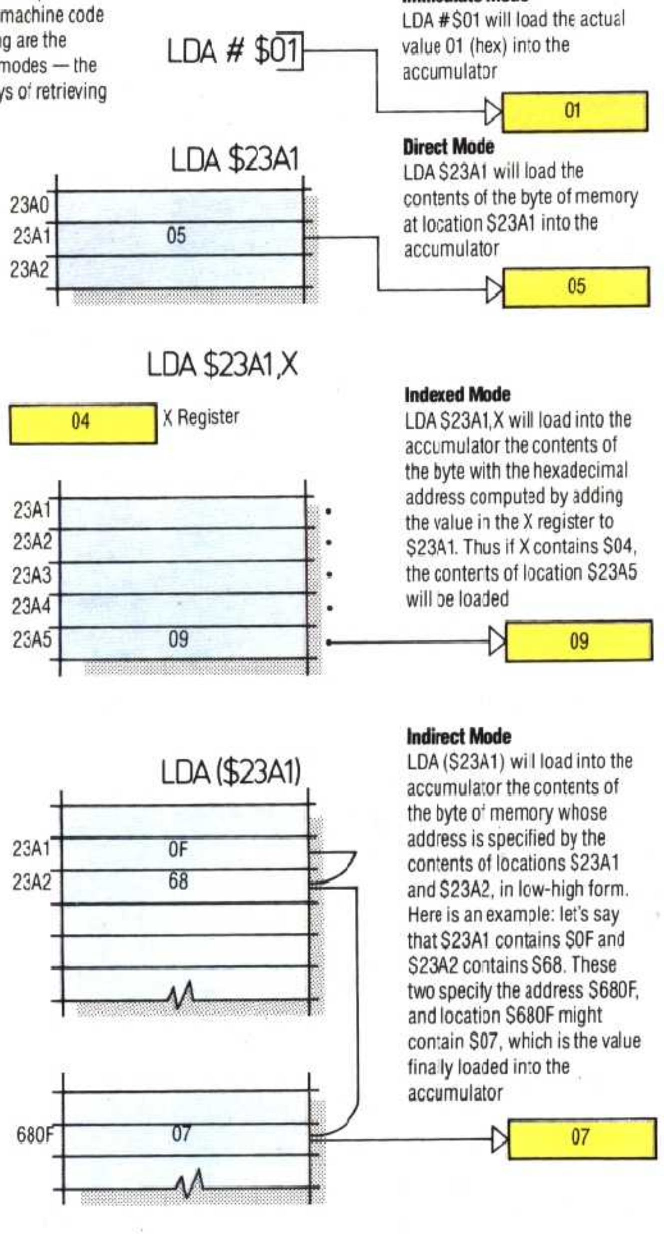
Note how we have to convert all the values from hex to decimal before they can be used in the POKE statement — inside the machine they will be stored in binary.

For longer hex dumps it is normal to use a short BASIC program called a 'machine code loader'. This asks for the start address and then the hex values. As each is entered, the short BASIC routine converts the hex value to decimal, and POKEs it into the next location. Alternatively, the hex dump can be READ by the program from DATA statements.

Once the machine code has been loaded, the BASIC loader program can be dispensed with. It's therefore important to load the machine code somewhere in memory where it won't be 'trampled over' by the BASIC program, nor be

Addressing Modes

Among the most powerful concepts in machine code programming are the addressing modes — the different ways of retrieving data



Immediate Mode
LDA #$01 will load the actual value 01 (hex) into the accumulator

Direct Mode
LDA $23A1 will load the contents of the byte of memory at location $23A1 into the accumulator

Indexed Mode
LDA $23A1,X will load into the accumulator the contents of the byte with the hexadecimal address computed by adding the value in the X register to $23A1. Thus if X contains $04, the contents of location $23A5 will be loaded

Indirect Mode
LDA ($23A1) will load into the accumulator the contents of the byte of memory whose address is specified by the contents of locations $23A1 and $23A2, in low-high form. Here is an example: let's say that $23A1 contains $0F and $23A2 contains $68. These two specify the address $680F, and location $680F might contain $07, which is the value finally loaded into the accumulator