BYTE BY BYTE

At this stage in the Machine Code course we follow the full procedure of program development — from defining the initial task through its Assembly language interpretation to the machine code itself — and provide a BASIC program that enables you to enter and observe the results of the code thus created.

In the previous instalments of the course, we have seen how BASIC program lines are reduced on entry to tokens followed by ASCII data. From this we realised that BASIC, although it is certainly a high-level language, isn't all that high-level: it consists essentially of sequences of instructions, and each instruction consists of a command word (immediately replaced by a token, which is itself only one level above a machine op-code) followed by the data for that command. The fact that the command words and their data (variables, numbers or strings) are closer to natural language, and that the instructions are visibly separated by line numbers or colons, makes a BASIC program look very much more high-level to us than it appears to the BASIC interpreter. It follows, therefore, that machine code should need only a little cosmetic work to make it reasonably comprehensible to our eyes.

The machine code 'cosmetic' that we use is Assembly language, in which alphabetic mnemonics like LDA and ADC stand for the single-byte op-codes that the microprocessor actually understands, and in which alphanumeric symbols such as LABEL1 and TFLAG can be used instead of memory addresses and numeric data. The microprocessor does not understand Assembly language, so before a program can be executed it must be translated into machine code — either by a program called an assembler, or manually by the programmer. The point of Assembly language is that it is machine code in translation. By simply substituting opcodes for mnemonics, and numbers for symbols, it can be turned directly into executable code. But it is much more comprehensible to human eyes than machine code could ever be, and is therefore extremely useful in program development. We shall always write programs in Assembly language, and hardly ever concern ourselves with the machine code equivalent until the very last stages of developing a program. But it's worth doing both at the moment for the sake of interest and for complete clarity, remembering that, in general, Assembly language will do everything we want.

The microprocessor can perform many different operations, but essentially it can only manipulate the contents of memory. It does this by acting directly on computer memory — the RAM and ROM chips that comprise the computer system — or by working through its own internal memory, which consists of *registers*. These are several bytes of memory physically located inside the microprocessor chip, which have certain special functions, but are otherwise indistinguishable from any other bytes of memory.

ACCUMULATOR REGISTER

The most important of the microprocessor registers is called the *accumulator*. It is connected directly to the Arithmetic and Logic Unit, and so is used more often than any of the other registers. In order to use it we must be able to put information into it, a process which is called 'Loading the Accumulator'. Using Assembly language, we say that the 6502 does this by performing the LDA operation, and in the Z80 by the operation of LD A. Taking information *from* the accumulator is as essential as loading it, and in 6502 Assembly language this is achieved by the STA (STore the Accumulator contents) operation. The Z80, however, regards both loading and storing as different cases of the same thing — i.e. data transfer. Therefore, taking information from the accumulator register is also done by the LD A operation, but in a different format — as we'll see later in this article.

Suppose, then, that we want to write an Assembly language program that will copy the contents of one byte of memory into the next byte of memory. Let's start by copying byte$09FF into byte$0A00. Immediately, we can express this in Assembly language as:

6502	**Z80**
LDA $09FF	LD A,($09FF)
STA $0A00	LD ($0A00),A

Notice that we're copying the contents of byte$09FF into byte$0A00, without knowing what those contents are: it's vital to get this clear from the outset. Byte$09FF may contain any number from $00 to $FF, and all our program does is load that number into the accumulator, then transfer it from the accumulator to byte$0A00. The 6502 version of Assembly language does not make it clear that LDA refers to the *contents* of $09FF, but it does distinguish unequivocally between loading (LDA) and storing (STA). The Z80 version does not make this latter distinction in its opcodes, but its instruction format is always:

```
OPCODE   DESTINATION, (SOURCE)
```