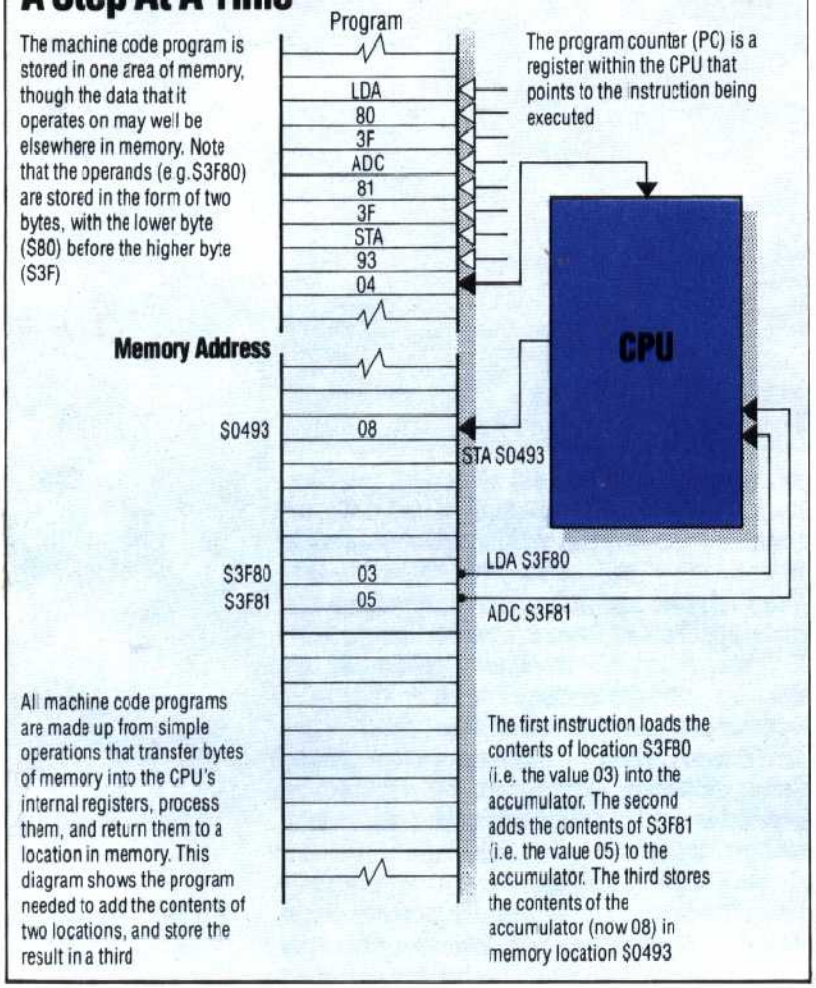
Machine Code

Learning machine code requires a considerable conceptual jump from Basic, but it offers a massive increase in speed and efficiency

A Step At A Time



KEVIN JONES

So far in THE HOME COMPUTER COURSE, all our programming has been centred around the language BASIC, because it is both versatile and easy to use. However, as your experience grows, and the programming projects you tackle become more adventurous, it will not be long before you encounter the limitations of this language. You will soon find that graphics can't be moved around the screen as fast as you would like, and that you often have to resort to the confusing PEEK and POKE commands to make the best use of your machine's facilities.

By contrast, programming in machine code imposes very few constraints on what you can do, and compared with BASIC, gives the impression of almost infinite speed. However, comparatively few home computer owners make the jump from BASIC to machine code, partly because using machine code is a far more labour-intensive programming process, and also because it is conceptually quite different from BASIC or any other high level language. Nevertheless, it is extremely worthwhile to have an understanding of machine code; and in this article, the first of two, we look at the fundamental procedures involved in using it.

Machine code, as we have explained before, is the language understood by the microprocessor (the CPU) that forms the heart of your computer. This microprocessor can only perform very simple functions (it can add two digits of a number, for example, but can't multiply them). It does, however, perform these functions at very high speeds. Every operation of a microprocessor is specified in terms of the number of 'clock cycles' taken. If the CPU in your computer runs at 1 MHz, then a clock cycle is one microsecond, and an operation that takes four 'clock cycles' to perform does so in four millionths of a second.

As a consequence, any program written in machine code will consist of a large number of instructions, and any function must be built up 'by hand' from simple operations. All machine code programming consists of manipulating individual bits or bytes of memory, using simple logic functions like AND, OR and NOT, and elementary binary arithmetic.

This is one reason why writing machine code is a slow task; the other is that it is the programmer's responsibility to know where everything is kept in memory. In BASIC, whenever a statement like LET A=5 is encountered, it is the job of the BASIC interpreter to find a space in memory to store that variable. Furthermore, whenever A is referenced later in the program, it will remember where to look for the necessary data. When you first start programming in machine code you discover that you have to specify an address (a memory location) for every piece of data you need to store, and it is up to you to ensure that it is not accidentally overwritten with other pieces of data.

Let's look at what machine code consists of. (Incidentally, all our examples will refer to eight-bit CPUs, such as the Z80 and 6502; 16-bit devices work in a similar manner but process twice as many bits with each operation). The microprocessor is connected to the computer's memory by two busses (a bus is merely a group of wires or lines): the address bus and the data bus (see page 144). There is also something called the control bus, but this provides timing signals for the CPU and is not used by the programmer.

The address bus is 16 bits wide, and by placing a pattern of bits on this bus, the CPU can select any of the 65,536 bytes in its 'memory map' (see page 329). In a typical home computer, some of these locations will consist of RAM, some of ROM, some of special input/output chips, and some will be unused. If the CPU wants to read a memory location (one of the lines in the control bus