INTRODUCING FIRST CONCEPTS

Machine code programming is the key to the real power of the microprocessor, allowing the programmer direct control over all the machine's functions. This first part of a comprehensive course, covering both 6502 and Z80 operation codes, will lead to a full understanding of the fundamentals of computer programming.

Machine code is a programming language, and it looks like this:

```
INSTK: SBC $D9FA,X ;Outport flag value
```

or like this:

```
DE23 FD FA D9
```

or like this:

```
11011110 00100011 11111101 11111010 11011001
```

Sometimes it looks like this:

```
1240 LET ACC=ACC-FLAG (X)
```

and sometimes like this:

```
PERFORM FLAG-ADJUST THROUGH LOOP1
```

It's all code of a sort, and since it's destined for a computing machine it's called *machine code.* To the machine it doesn't actually *look* like anything at all, being simply a pattern of voltage levels or a current of electricity.

What we usually mean when we say machine code is Assembly language, and the first example we gave in this article is an instruction in 6502 Assembly language. The point of giving all the other examples was to demonstrate that there is no specific machine language as such, only a number of different ways of representing a sequence of electrical events, and representing them in ways that we find more or less easy to understand. So the first thing to learn about machine code (or Assembly language – we won't worry about the distinction for the moment), is that it's just another programming language. However, the programming must always come before the language: whether you write your programs in IBM Assembler, Atari BASIC, or Venusian PsychoBabble, you have to solve the programming problem in your mind before you touch a keyboard. The programming language in which you then express your solution will obviously influence the form of the final program. Indeed you may choose among various possible languages precisely to make the coding of your program easier, or shorter, or more readable. But the solution must always come first: content must precede form.

In that case, why call it machine code, and why bother to use it at all? We give the language this name because its instruction set corresponds exactly with the set of 'primitive' or fundamental operations that a particular microprocessor can perform. We use the machine code when it is important to direct the operation of the microprocessor exactly, step-by-step, rather than allowing a program language interpreter to control it in a more general way.

The commonest reason for wanting to use it is speed: if your program addresses the processor more or less directly, then you avoid the relatively lengthy business of program translation. In other words, by cutting out the middleman you save time. Program execution time, that is. The actual coding, testing, debugging, modification and maintenance of a machine code program is likely to take at least twice as long as the same operations would on a high-level language program. The unfriendliness and intractability of machine code stimulated the development of languages such as COBOL and BASIC.

If the set of machine code instructions is the set of processor operations, then what are these operations, and what does the processor do? In the simplest terms the Central Processing Unit (CPU) of a computer is a switch that controls the flow of current in a computer system between and among the components of that system. Those components are the memory, the Arithmetic Logic Unit (ALU), and the Input/Output devices. When you press a key on the keyboard, you are inputting some information; in the machine, however, you are simply generating a pattern of voltages in the keyboard unit. The CPU switches that pattern from the keyboard to part of the memory, then switches a corresponding pattern from elsewhere in memory to the screen so that a character pattern appears on the screen. To you this process may seem like operating a typewriter, but in a typewriter there is a mechanical connection between hitting a key and printing a character, whereas in a computer that linkage exists only because the CPU switches the right voltage patterns from place to place. Sometimes pressing a key doesn't cause a single character to appear on the screen: the keypress may destroy an asteroid, or save a program, or delete a disk file, or print a letter. The operation depends on how and where the CPU switches the electric current.

In this simplistic view the CPU is at the heart of the system, and all information (or electrical current) must pass through it from one component