TEXTUAL ANALYSIS

Before going on to investigate how machine code programs work, it is salutary to look at how BASIC programs are stored (in the BASIC Text Area of memory) and implemented (using the BASIC Interpreter program). This will serve as a reference point later when we come to discuss the way machine code operates in memory.

When you type or LOAD a BASIC program into the computer, you probably imagine that the computer is an empty vessel doing nothing until your instructions arrive. In fact, from the moment that the power is turned on, the computer is constantly running a sophisticated program of its own — the Operating System. This is a program, or set of programs, permanently burned into some of the ROM chips inside the machine. Its purpose is to make the machine work: it puts a display on the screen, it communicates with the printer and the disk drives, it scans the keyboard for keypresses, and so on. To the O.S. everything that comes into the machine is just data to be processed by its own programs.

One of these programs is called the BASIC Interpreter, and its purpose is to inspect the text of BASIC programs, and to implement their instructions. Everything in a BASIC program, therefore, is just data for the Interpreter program to process. When you type in a program, the Operating System recognises it as such because each new line begins with a valid line number. With some exceptions every character of that program line is stored in its own byte of the BASIC Program Text Area of memory. When you type RUN, the Operating System hands over control to the BASIC Interpreter, which — like any program — goes to work on processing its data (the contents of the BASIC Text Area).

The Interpreter does not change your program in any way, but simply interprets and implements it. And because the Interpreter obeys commands without question, it is quite possible to instruct it to look at the contents of any area of memory. If your program happens to allow you to inspect memory and you use it to inspect the Text Area, that's no paradox to the Interpreter. It just follows instructions if it can, and reports SYNTAX ERROR or OVERFLOW ERROR or something similar if it can't. It has neither the reasoning nor the vocabulary to issue error messages such as: TEMPORAL PARADOX or PHILOSOPHICAL DISCONTINUITY.

The Operating System stores your BASIC program character-by-character, with the exception of the BASIC keywords. Whenever it recognises the letters (or characters, or numbers, or voltage patterns) that make up a BASIC keyword, the Operating System replaces that word by a single-byte code number, called a *token*. This saves memory space — RESTORE, for example, would otherwise use up seven bytes — and means that the Interpreter's job of translating the BASIC program is much easier to perform.

Different machines use different token conventions, but, in general, token codes are numbers greater than 127. The ASCII codes for the printable characters (shown in the table on page 77) are all in the range 32 to 127. Therefore, any byte in the BASIC Text Area containing a number bigger than 127 must be a token byte put there by the Operating System. When the Interpreter encounters such a byte it simply implements the appropriate built-in subroutine.

The question arises, however, of why, when you LIST a program, you don't see unprintable characters, but rather the BASIC keywords, etc.? The answer is that during a LIST the Operating System inspects each byte of the Text Area, and whenever it finds a byte having a value greater than 127 it treats it as a token. Somewhere in memory is stored a complete list of the ASCII representations of BASIC keywords and the value of a token will point to that position. It's just the same as if the Interpreter were using the token's value to locate its implementation subroutine. And consequently, the Operating System puts the keyword rather than the token on the screen during a LIST. You can demonstrate this to yourself on a Commodore 64 very easily. (It's less straightforward on the BBC and Spectrum.) In lower-case mode, type:

```
100 rem*******h**********
```

Now LIST 100, and you should see:

```
100 rem*******leftS**********
```

On the Commodore machines, the ASCII value of 'H' in lower-case mode is 200 so when the O.S. found a value of 200 in that particular byte during the LIST, it interpreted it as the token for the keyword LEFT$. If you now type:

```
100 rem"*******H**************"
```

and LIST 100, you'll see:

```
100 rem"*******H**************"
```

This demonstrates that it is important to remember that some printable characters, usually graphics characters, do have ASCII codes greater than 127, and they will be recognised as such,