means 'load the accumulator with the contents of the memory location whose address is five bytes higher than the present contents of the location counter'.

ASSEMBLER DIRECTIVES

An assembler will normally accept a number of *directives* or *pseudo-ops*, which are used in the program like ordinary op-codes. We have already used two of them (see page 538):

```
CR        FCB     13
```

(Fix Constant Byte) reserves a single byte at the current location counter address, and gives it the value of the operand — here, the location whose address is represented by the symbol CR is initialised with 13.

```
MEMTOP    FDB     $7FFF
```

(Fix Double Byte) does the same for a two-byte (16-bit) value. Memory space can be reserved in blocks without fixing the value of its contents by using the RMB pseudo-op, as in this example:

```
TABLE1    RMB     7
TABLE2    FCB     $F6
```

which reserves seven bytes for a table of values whose first byte has the address represented by the label TABLE1. In practice, this means that if TABLE1 represents the address $C104, say, then TABLE2 will represent an address seven bytes higher — that is, $C10B.

An entire character string can be placed in memory, like this:

```
ERRMSG    FCC     'ERROR
```

This initialises five bytes of memory with the ASCII codes for E,R,R,O and R. Consequently, it is much easier to include user messages and prompts in Assembly language programs.

An important pseudo-op is ORG (ORiGin), which specifies a value for the location counter, and is used at the start of a block of program to instruct the assembler where in memory to begin locating the program when it is translated into machine code. You should always start programs with an ORG statement, though some assemblers may supply a default value if you don't. It is permissible, and sometimes desirable, to have more than one ORG statement in a program. An ORG directive does not take a label or an operand.

Perhaps the only statement that you might want to put before the ORG is EQU (EQUals or EQUate), since it specifies variable symbols and does not refer to the location counter. For example:

```
RESET     EQU     $F100
```

means that the symbol RESET is defined as representing the value $F100. RESET is thus shorthand for a number, whereas a label placed at the start of a statement line represents the address where data or machine code is stored.

Appropriately, the last directive we need consider is END, which is placed at the end of the source code as a terminator for the assembler. Like ORG, it takes neither a label nor an operand.

6809 ADDRESSING MODES

A measure of any Assembly language's power is its addressing modes — that is, the number of ways that the operand can be interpreted. The 6809 processor supports many such modes. The sample instructions that we have seen so far have all used either *direct* or *extended* mode, meaning that the value or label in the operand field is the address of the memory location that contains the data.

Direct addressing means specifying only a single byte address for the instruction operand. This is treated by the processor as the lo-byte of the full two-byte operand address, and it takes as the hi-byte of the address the contents of the *direct page register,* an eight-bit CPU register addressable by the program. The advantage of this mode is flexibility and generality: a subroutine, for example, can be written using direct addressing, and, therefore, not refer explicitly to any particular area of memory. Specifying the direct page register contents before calling the routine would 'point' it wherever the data lay in memory.

Extended addressing means specifying a two-byte address as the instruction operand. This instruction will then always refer to that particular byte of memory, and so is an inflexible piece of code. The assembler recognises direct and extended modes by the nature of the operand.

Another commonly-used mode is *immediate,* where the actual data is contained in the operand field itself. This mode is indicated by prefixing the operand with a #, the hash symbol. For example:

ORG $1000	start address of the machine code is $1000
NUM1 FDB $FFFF	initialises location NUM1 with $FFFF
LDD NUM1	loads $FFFF, the contents of NUM1, into the D register
LDD #NUM1	loads $1000, the actual (or immediate) value of NUM1, into the D register

Notice that the label NUM1 represents the address $1000. You might expect that the ORG instruction would reside at address $1000, and that the following instruction (and, therefore, its label — NUM1) would have a higher address. Remember, however, that ORG is an instruction to the assembler program about translating the Assembly language program, and not a part of the program itself. It does not, therefore, take up any memory space, so NUM1 takes the value $1000 because that is the value of the location counter when NUM1 is first encountered by the assembler. You should also notice that LDD NUM1 loads the *contents* of location NUM1 ($FFFF as specified by the FDB directive) into the D accumulator, whereas LDD #NUM1 loads the *value* of NUM1 itself (the address $1000 of the label).