FIELD WORK

Our 6809 machine code course has so far taken a close look at the operation of the registers in storing and moving data. We now consider the layout of Assembly language programs in more detail, inspect some new assembler directives and discuss addressing modes.

As it is entered into an assembler, an Assembly language statement consists of three parts — although not all of these need be present for each statement. The three parts or *fields* are:

- **The Label Field,** which is found in the leftmost column of the screen display. Labels are simply identifiers representing numbers, which are usually memory addresses. A label consists of from one to six alphanumeric characters: the first character must be alphabetic, and the name as a whole must not be the same as a register name, an Assembly language op-code or an already-defined label. These are the usual assembler conventions, but they are only conventions — different assembler programs may follow different rules. If you do not wish to label a line, then you should start it with at least one space character, indicating an empty label field. Similarly, a space terminates the label field; LABLDA, for example, is a valid label, whereas LAB LDA is interpreted as the label LAB followed by LDA, the op-code.

The assembler maintains a *location counter,* which is the equivalent of the processor's program counter register. It holds the address of the memory location where the next byte of instruction or data is to be stored. When the assembler first encounters a label, it saves the identifier in an area of memory called the *symbol table* — this is like an array in a BASIC program. Along with the identifier it stores the address held in the location counter at the point when the label was first encountered. Whenever the assembler encounters a label in the Assembly language program, it inspects the symbol table for that label. If it is in the table then the assembler replaces it by the address given for it, and if it is not in the table then the assembler stores it there along with the location counter contents.

- **The Operator, Instruction or Op-code Field** is found to the right of the Label Field. This is a mnemonic, usually consisting of three characters, and — if necessary — a register name. Thus, ADDA is formed from the mnemonic ADD and the register name A. The op-code represents the processor operation to be executed. Like the label field, it is terminated by space.

- **The Operand or Address Field,** which gives information about the data on which the op-code is to operate. Normally this data is in the form of an address, or more often, a label representing an address.

Consider this Assembly language statement:

```
LABEL1  ADDA  NUM1
```

meaning 'add the data stored at the address represented by the symbol NUM1 into accumulator A'. The address of this instruction is now stored as LABEL1; if we wish to jump or branch to this statement, then we use its label to indicate the jump destination. NUM1 is a label that is defined elsewhere in the program, and represents an address where data is stored. Now let's consider another example:

```
        CLRA
```

meaning 'CLeaR accumulator A — i.e. set its contents to zero'. This is an example of an op-code that needs no operand. Notice that we have not labelled this line. Labels are entirely optional — you should use them because the line will be branched to by some other instruction, or as a kind of REM statement, labelling significant lines with explanatory remarks.

Using labels in this second way is helpful, but it is no substitute for full explanatory comments. These can be added to any line by leaving a space after the last character of the operand, and then inserting your comment. Some assemblers may require a special character to indicate the beginning of a comment, and it is normal to start a whole new line of comment this way, usually with an asterisk.

Constants, in the form of numbers or character strings, can be used in the operand field. Numbers are usually taken as being in decimal, unless indicated as hexadecimal by a $ prefix or an H suffix (e.g. $AF08, AF08H); or octal (number base eight) by a @ prefix or a Q suffix (e.g. @6712, 6712Q); or binary by a % prefix or a B suffix (e.g. %11010011, 11010011B). The ASCII code of a character can be used as an operand by prefixing it with an apostrophe, thus 'A, meaning 65 or $41.

A particularly useful value is the current location counter contents. This is not usually known to you when you enter the program, but can be referred to in the operand field by an asterisk. Most assemblers will accept this in simple arithmetic expressions — usually restricted to arithmetic and subtraction. For example:

```
        LDA     *+5
```