THE UNIVERSITY OF NOTTINGHAM

FACULTY OF APPLIED SCIENCE

Applications of Computers

LECTURE 7

"THE TASK OF THE PROGRAMMER"

by

S. GILL, M.A., Ph.D.

THE TASK OF THE PROGRAMMER

PROGRAMMING TECHNIQUES

We have already seen the part played by programming in controlling the operations to be carried out by a computer. The programmer's task is to prepare a set of instructions for a given problem, according to the instruction code and other characteristics of the computer available. In this, and the following lecture we shall consider what this task entails.

In order to illustrate a few simple programming jobs, we shall imagine a machine with a single store containing one hundred registers, numbered from 00 to 99, each containing seven decimal digits. The instruction code will be of the three-address type: the first decimal digit of an instruction will indicate the type of function (e.g. one for addition, two for multiplication, etc.), and the remaining six digits will be taken in three pairs to represent three addresses in the store. In the case of addition and multiplication, the first two addresses will be the addresses of the registers containing the operands, and the last address will be the address of the register which is to receive the result.

SIMPLE ARITHMETIC

Suppose that at some stage in a calculation, the numbers p and q have been formed and placed in registers 10 and 11 respectively, and the number x is in register 0. It is then required to form $px + q$ and to place this in register 1.

This task requires two machine operations: a multiplication, followed by an addition. Two instructions are therefore required as

follows:-

EXAMPLE 1.

```
2100001      put  px  in register 1
1011101      put  px + q in register 1
```

Assuming that the computer takes its instructions from consecutive registers, these two instructions should be placed consecutively. Any two consecutive registers will do, provided of course that they are not already reserved for data or for another part of the programme. The programme must be designed so that, at the moment at which this piece of calculation is to be performed, the machine arrives at the first of these instructions.

Suppose now that the number r has also been formed in register 12, and it is required to put $px^2 + qx + r$ into register 2. This problem illustrates in a simple way the need for a little preliminary planning. The machine could be made to form px^2 and qx separately, but this would require it to perform three multiplications. The amount of work can be reduced if px + q is first formed as in the last example, the complete task being programmed as follows:

EXAMPLE 2.

```
2100002      put  px  in register 2
1021102      put  px + q in register 2
2020002      put  px^2 + qx in register 2
1021202      put  px^2 + qx + r in register 2
```

A COMPLETE PROGRAMME

In addition to performing a calculation, the computer must also read the data for the calculation, and cause the results to be printed or punched out. This input and output is also controlled by the programme, which must therefore include the necessary instructions for it.

Owing to the variety of forms in which data is commonly presented, the programming of input and output operations in practice tends to be a little complicated. For simplicity, however, we may assume that the machine has an input instruction that will cause a number to be read from cards or tape and placed in one of the storage registers, and an output instruction that will print or punch a number from a storage register. These types of instructions will be distinguished by the function digits 3 and 4 respectively, and the first address part of these instructions will denote the storage register involved. The remaining address parts are not required and will be ignored.

For example, a complete (if trivial) programme might cause the computer to read in succession the numbers p, q and x and to print the quantity px + q. The programme would be as follows:-

EXAMPLE 3

```
3100000   read p into register 10
3110000   read q into register 11
3000000   read x into register 0
2100001   ⌐  compute px + q
1011101   ⌙  as in example 1
4010000   print result
```

When the machine is required to execute this calculation, it is first necessary for these instructions to be loaded into a suitable part of the store. When this has been done, the machine is directed to begin obeying the programme at its first instruction.

For loading, the programme must first be prepared in a suitable form on tape or cards, following certain established conventions. The actual loading operation is then itself controlled by a special "programme input" routine, which is used for all programmes. Various means are used to get the programme input routine into the store initially.

SOME COMPLICATIONS OF PROGRAMMING

In the simple examples given so far, the instructions were merely carried out as given in the sequence in which they appeared in the store. There are two complications which characterise programmes in practice, and which lead to a tremendous degree of flexibility in the operations which a computer can be made to perform.

The first complication is the fact that the sequence in which the instructions are actually obeyed can be interrupted by certain special "jump" or "transfer of control" instructions, and that these breaks in the sequence can be made to depend upon the numbers which are being computed. Thus we can envisage a jump instruction, distinguished by the function digit 5, which causes the machine to take its next instruction from the register whose address is given by the first address part of the jump instruction. Furthermore we may suppose that this jump is conditional upon the values of two numbers in the store, i.e. that it will occur only if the first number is less than the second. The addresses of the registers containing these two numbers may be specified in the last two address parts of the jump instruction.

The other complication is the fact that, since the instructions look and are stored exactly like numbers, and any storage register may contain either a number or an instruction, there is nothing to prevent the programmer from writing instructions that will cause the machine to carry out arithmetic on another instruction. The result is a programme which changes itself as the calculation proceeds.

Both of these techniques are in common everyday use. Unfortunately there is not space here to illustrate them by means of simple individual examples, but the following example combines them together and also illustrates the way in which a programmer proceeds to construct

one of the commonest patterns that appear in programmes: the cycle of instructions.

Suppose that a vector of order 50 has been read into the machine, and that its elements are stored in the registers 50 to 99 inclusive. It is required to read another similar vector, and to print out the scalar product.

The programmer does not approach this problem at the beginning, or at the end, but in the middle. The body of the calculation consists of a few simple operations which are obeyed repeatedly, once for each successive term of the scalar product. He therefore uses the same set of instructions for each repetition, and causes the machine to obey them repeatedly by jumping back from the last to the first. He begins by considering the essential instructions which are to be repeated.

The operations which must be carried out for each term are: to read one element into the computer, to multiply it by one of the elements already in the machine, and to add the product to the partial sum obtained so far. Suppose that the element which is read in is put into register 0, the product is formed in register 1, and the partial sum is kept in register 2. Then the following instructions are required.

EXAMPLE 4

```
10)  3000000   read element into register 0
11)  200--01   multiply by one of elements in 50 to 99
12)  1020102   add product into register 2
```

It will be seen that the middle address part in the middle instruction cannot be filled in yet, because it must vary from one repetition of these instructions to the next. In fact it must be increased by one at each repetition. The programmer therefore attends to this next, and adds the instruction:

EXAMPLE 5.

```
13)  1110311   increase middle address in instruction
               in register 11 by 1, assuming that the
               following constant is stored in register 3.
 3)  0000100
```

The next matter for the programmer to attend to is the closing of the cycle, i.e. arranging a jump instruction to cause the cycle to be repeated. This jump instruction must be given careful attention, for it must cause repetition only on the first 49 occasions when it is encountered, and not on the 50th. If care is not taken, the machine may find itself obeying this same cycle of instructions indefinitely. The jump is therefore made conditional upon the value of a number which is changing. In this instance, the second instruction is itself already changing, and can therefore be used as a subject of test. If we assume that a suitable constant, e.g. 2009950, is stored in register 4, then the following jump instruction will compare the changing instruction with this constant and cause a jump under the required conditions.

EXAMPLE 6.

```
14)  5101104   jump to instruction in register 10
                if changing instruction is still less
                than 2009950
```

Finally, the programmer may consider any preparations that are necessary before the cycle is entered (in this case none), and the operations which must succeed the cycle. He then writes down the initial state of all the relevant registers as follows:

EXAMPLE 7.

```
 2)  0000000   scalar product added in here
 3)  0000100 ] constants (not instructions)
 4)  2009950 ] used by programme

10)  3000000 ┐
11)  2005001 │ cycle, as composed above
12)  1020102 │
13)  1110311 │
14)  5101104 ┘
15)  4020000   print out result
```

It will be noticed that two instructions in the above example are concerned with operations which are purely incidental to the main calculation.

These are known as "red tape" operations. The proportion of red tape instructions is often even higher, and one of the main tasks facing machine designers is the reduction of time taken by red tape operations. Many devices are employed, including the idea of "modifier" registers, which complicate the instruction codes of real computers in varying degrees. The classification of instruction codes according to the number of address parts in an instruction therefore gives only a rough guide to their structure. However, whatever the instruction code, the above techniques for constructing programmes still apply.

PROBLEM DEFINITION

We turn now to some of the broader aspects of the work of a programmer. He is not merely a high priest attending the machine, but must rather act as a link between the machine and the person posing the problem. His first task therefore is to obtain a clear idea of the problem itself.

It is rare for a problem to be couched in unambiguous terms divorced from the peculiar phraseology of the subject of origin. The programmer may therefore need to be familiar with this subject in order to be able to understand the problem that is presented to him. Familiarity with the subject may also be of great help to the programmer by enabling him to anticipate the behaviour of the solution.

Often his knowledge of the capabilities of the computer may lead a programmer to suggest a reformulation of the problem, in order to take full advantage of the facilities offered. The final formulation may then be a result of discussions between the programmer and the originator of the problem.

Obviously such discussion can be eliminated if the originator of the problem also does the programming, and a more efficient utilisation of

the machine is then likely to be obtained.

DECIDING ON THE METHOD OF SOLUTION

When the problem is formulated, or perhaps while it is being formulated, the programmer must work out the method of solution to be adopted. Sometimes a problem is stated in a way which implies that a particular method of solution will be used. If this has been done by a person who is not familiar with the characteristics of computers, it may be necessary for the programmer to re-trace the steps back to the original problem and to investigate other ways of tackling it.

A satisfactory method is one which works, which produces the required degree of precision in the results, and which is cheapest (taking into account both the programmer's time and computer time). If there is already a method which is known to work, this may well be the best one to adopt, since developing a new method may very well absorb large and unpredictable amounts of the programmer's time. If furthermore the method has already been used on that computer, it will often be possible to make use of parts of the programme again.

Once source of trouble which can take up a great deal of programming time is the possibility of overflow of numbers beyond the values accommodated in their registers. In most programmes it is possible in theory at least to assign scale factors to all numbers in such a way that they do not exceed capacity, and yet remain large enough to be represented with sufficient precision. However, working out such scale factors can be a very tedious task, at least in scientific calculations. Hence, even if it is not strictly necessary, the "floating point" method of representing numbers is often used. Considerations of number representation, and of rounding-off errors, may affect the choice of mathematical method to be employed.

Another practical consideration, which is even more likely to affect the choice of method, is the allocation of storage space within the computer. The programme must be so arranged that at no time is it necessary to store within the computer more than a certain amount of information (remembering also that there must be room for the programme itself in the store). Often the storage space requirements may be reduced at the expense of more input and output operations; it must be remembered however that these extra input and output operations will consume more computer time.

The choice of method is therefore bound up intimately with the formulation of the problem itself, the characteristics of the computer, and the available programming techniques. The programmer himself must weigh up all these factors and come to a reasonable decision.

In the case of the scientific calculation, the method of solution can usually be described quite simply in a few sentences and equations. With more complicated calculations, however, such as those found in business, it may be necessary to draw up a flow chart to depict the sequence of the various parts of the calculation.