Lecture Notes. A.E. Glennie '3001

The Automatic Coding of an electronic computer.

Summary.

The possibility of using the computing machine itself to do its own coding is discussed. For such a purpose a notation for programmes is needed. The use of a quasi-mathematical, and descriptive notation is proposed and an example of such a system now in use for the Manchester University Computer is described.

It is pointed out that using a familiar notation for programming has very great advantages, in the elimination of errors in programmes, and the simplicity it brings to a subject, now labouring under the disadvantages of obscure notations.

An example is given of the programme for evaluating a simple function of one variable.

14/12/52

Lecture was delivered at University of Cambridge mid-February 1953.

Automa tic coding; using the machine to programme itself.

Anybody who has to programme for an automatic computer knows that successful programming is difficult, not becausd it is difficult to imagine ways of doing the job, but because it is almost impossible to avoid blunders; and of course even one blunder will cause trouble, perhaps immediately and certainly in the long run. The difficulty of programming has become the main difficulty in the use of machines.

Aiken has expressed the opinion that the solution of this difficulty may be sought by building a coding machine, and indeed he has constructed one. However it has been remarked that there is no need to build a special machine for coding, since the computer itself being general purpose should be used for coding. The use of the machine itself for coding is to be the subject of my remarks this afternoon.

What do we need to do to make programming and coding easy. I think the answer is simple:-

to ~~reduce~~ make it easy one must make coding comprehensible

This may be done only by improving the notation of programmimg. Present notations have many disadvantages, all are incomprehensible to the novice, they are all different (one for each machine) and they are never easy to read. It is quite difficult to decipher coded programmes even with notes and even if you yourself made the programme several months ago.

Assuming that the difficulties may be overcomm, it is obvious that the best notation for programmes is the usual mathematical notation, because it is already known. But since the computing machine is limited in its repertoire of operations there will be a similar restriction in the notation of programming. The notation would not include operations like differentiation or integration not because these operations may not be performed approximately using the machine but because of the uncertainty of what is meant by integration for example. There are dozens of methods of integration and quadrature and to consider a notation with several different types of integration would be as unwise as it is unnecessary..

On grounds such as these, I suggest that it ~~si-ne-~~ is best to restrict the notation to include only the basic arithmetic operations of the machine and the basic operations of programming techniques, such as the transfer of control, unconditional or not; the use of a subroutine ; a loop or repetitive cycle ; and so on.

Another and more vital reason for a simple and restricted notation is that of the small number of distinct symbols availible for the representation of programmes on the input medium of the machine. Thus with five hole tape there are 32 distinct symbols and no more out of which to build the notation. The scope for expression of the written word shows that there is no limit to the complexity of representation with a similar number of letters of the alphabet. However the notations of mathematics exhibit a quite different tendancy; that of using a multitude of symbols (single symbols). I tried counting the different symbols on a page of W and W' and found no fewer than 35 symbols used in mathematical senses, expressing such operations as integration, raising to a power, infinite summation and so on as well as the letters used for algebraic variables. To programme the machine to cope with integration for instance is to programme the machine to exhibit an activity in the field of numerical amalysis. Although it may become possible in the future to do so there are many simpler things to do first.

We therefore seek a notation for numerical analysis. and so we only require a simple notation capable of expressing the simple arithmetic operations, in fact

only those ~~notations of~~ operations that the machin can perform, addition multiplication and subtraction and perhaps divisioh.

Let us explore the possibilities of a simple notation, for example the addition of several variables. written

~~a+b+c+d~~ $h+n+m$

How may this be coded on the machine? Take Edsac code for example. We may use the preset parameters H, N and M to represent addresses in the usual manner so that the coded representation is

A H
A N
A M

There is no reason that we should not adjust our notation so that A is written + . Now the coded representation is the same as the mathematical one. This is achieved by relabelling the A key on the perforator.

This shows the essential features of a simple notation. It uses letters like the H, N and M to represent storage locations containing the numerical values of the variables. and a few operation signs like +, −, etc ~~Thus the notation represents processes.~~ In the simplest form the operations ~~processes~~ will be those of addition subtraction and multiplication.

Hence expressions like these will occur

+x +yx -yz -zx -z

Having formed this quantity we may require to keep it for future use. This may be indicated by directing it to one of the addresses for which a variable is defined thus +x + yz → t.

You will notice that in the expression that is given there are no products of three or more variables, and hence the expression may be accumulated numerically in the accumulator. (I may remark here that I am considering the use of this notation for machines with a single address code. The extension to multiaddress codes is not so simple.) Making this simple restriction allows the notation to be easily translated to a coded form. It might be recommended as a useful notation for general use; for this particular purpose it is very suitable as the automatic translation of this ~~is~~ by the machine is fairly simple to programme. You will notice that if a tape bearing these symbols is fed into the machine the translation programme need only be able to operate on the components of the "equation" ~~lik-or-the~~ since the translation of each compoment is independent of the preceding or following terms. Thus the translation of the following expression

+x +y +z → a

is made term by term.

If products of three or more Quantities, or bracketed expressions are included in the notation, the translation process would become extremely complicated.

I have sketched out a notation for the numerical business of programmes. But this is only half the of the majority of programmes, the other half being taken up with the organisation of the automatic sequencing of the computation. Since single symbols are used for the arithmetical notation and as amny as possible are needed for this, it is convanient to use grouped symbols for specifiying the details of the ø organising parts of the programme. In the Translation programme that I have made for the Manchester Computer English words have been used to specify such things as subroutine entryes. I will mow give an example of a complete programme in a form that may be used directly on the machine.

Exanple:-

Given that Subroutines 2 and 3 respectively print and find square roots, calculate and print the function $f(x) = 0.12x + 0.05x^{\frac{1}{2}}$, for $x = 0\ (0.001)\ 0.1$.

```
a@/E  b@:E  f@:C  Z@IC  c@IE  x@VA
FRACTIONS +12→a   +0→x  +05→b  +001→c
LOOP  100r
+x       SUBROUTINE 3   →  z
+zb  +ax  →  f
+f       SUBROUTINE 2
+X  +c   →X
REPEAT  r
ENTRY A
CONTROL A
WRITE 1 START 1
```

Notes.

The first line of symbols defines the variables x,z,a,b,c,f in terms of the addresses of the store. Thus a "is defines to be" the storage location /E .(In the Manchester computer addresses are given in the scale of 32, in which the symbols / and E stand for certains numbers).

The word FRACTIONS brings into action a routine for placing decimal fractions in the store;here the decimals are sent to the locations a, b, c and x.

The words LOOP and REPEAT form a loop which counts the correct number of repeats.In the loop the necessary computation is given.

When the loop is completed, there is a "dynamic stop". This is a transfer of control indefinitely repeated, shown here by the word CONTROL which denotes a transfer of control, control being directed to the part of the programme at which the word ENTRY occurs. These words are olabelled with a single $ symbol for the purpose of identification, corresponding words having the same label.

This completes the programme, the remaining words WRITE 1 and START 1 are instructions to the input routine to record the programme on the magnetic drum so that it may be used as the first subroutine, and then to begin to obey the orders of the programme.

With the Mahchester computer, the subroutines of a programme are recorded on the magnetic drum and when required, are transferred to the fixed position in the electronic store, normally the first two storage tubes containing the addresses 0 to 127. In the example given the subroutines print (or square root) a number which is carried in the accumulator, and in the case of the square root the result of the subroutine appears in the accumulator after the operation of the subroutine. Thus in the example, where subroutine 3 is used the symbols +x put the value of x into the accumulator ready for the operation of the subroutine which leaves the answer in the accumulator whence it is transferred to the location z for future use.

This example shows a few of the possibilities of the programme that I have made for the Manchester machine. In the complete notation there are words for three types of transfer of control, two types of subroutine, together with those shown in the example.

When a programme like this has been written down, it must be punched as written with blank tape punched where spaces occur naturally on the page. It is punched exactly as written with the single exception of the → sign which is not shown on the keyboard of the tape punch. The symbol " is used for this. There are certain other rules for punching that are merely a matter of common sense such as not leaving spaces in the middle of words or mispelling them. I have arranged that such accidents will cause the input programme to exhibit symptoms of distress.

The input programme that translates this into machine code is quite long but not excessively long, about 750 orders This means that such a programme, unless considerable economies are made will not be possible on machines lacking an auxiliary store.

I will describe now the mode of action of the translation. This is not a difficult matter to understand as it must be similar the way in which the meaning is extracted from any similar notation by a human being. The method is as follows.

The machine starts by reading blanktape until it finds a symbol punched on the tape. On receipt of this symbol depends the subsequent action of the programme depending on

(1) whether the symbol was +,- or → or

(2) whether it was any other symbol.

Take first the second possiblity; this covers all notation that is not algebraic. The next action is to examine the next symbol on the tape. If this is @ it means that the machine is redding the symbol groups defining the variables like the x and z or the example. In this case the next two symbols define the variable whose address is now stored for use

whenever the corresponding variable appears in the algebraic ƒ(x) notation.

If this is not the case the only other possibility is that the symbols being read belong to an English word.

If so the tape is read by the machine until blank tape is reached and the complete word has been taken in. During this process the numerical values of the characters are used to form a 20 binary digit word in a similar manner to the process used for the input of numbers on any computing machine, the number being formed by adding the number from the tape to a multiple f(0) of the number formed at the previous stage. This 20 digit number is then compared ≠ with the 20 digit numbers corresponding to the repertoire of words of the system and if agreement is foundwith one of themthe appropriate subroutine is entered. If ≠ the 20 digit words does not agree with any of the words in this vocabulary the input programme exhibits the signs of distress that I mentioned previously. This consist of the programme coming to a stop and the machine making no further moves.

Corresponding to each word of the notation there is a subroutine of the input programme which searches the tape for the next symbol or group of symbols and takes the appropriate action to make up the corresponding coded form. Thus for the word SUBROUTINE three orders are made up, one of which is made up using the label number so that the correct subroutine is entered when the programme is used.

It is instructive to consider the way in which the transfers of control operate as they are the most used of the programme operations and the f(0)s(x) easiest to describe.

In the code of the Manchester machine it is necessary to set aside 2 storage locations for each transfer of control, one of which contains the ~~funct~~ the instruction calling for a transfer

Transfers of control can occur in two ways, with contorl transferred to an order with a higher or a lower address, i.e. either forward or backward. I will consider only the first case, denoted in the notation when the word ENTRY occurs after the corresponding word calling for the transfer of control.

When the transfer of control word appears two orders are added to the partly constructed programme, the first being the transfer of control order itself with its address part referring to the second part which is left blank until the corresponding ENTRY word appears. At this point the necessary transfer of control number is known and this is then filled into the second of the two locations specified by the transfer of control word. (The use of two storage locations to specify a transfer of control is a feature peculiar to tthe Manchester machine and not to the style of coding produced by this method.

The method of dealing with the descriptive words is very similar in allcases. The subroutine of the input programme corresponding to the word is brought into play and this then searches the tape until it finds either the single symbol or multi-symbol group that ought to follow the word.

The part of the input programme that deals with the translation of the algebraic notation is the most intricate programme that I have ever devised, but some of the difficulty is traceable to the particular machine for which the programme was designed and is not to be expected for some other machines. In any case the number of orders required for the translation of the algebraic notation is a small fraction of the total, about 140 orders.

I have explained how thw beginning of an algebraic group is recognised. This group consists of those symbols occurring between two operation signs like + or −.

Although the complete algebric equation may contain a large number of separate terms or groups it is possible to translate the symbols group by group since the translation of each group does not depend on the translation of th se groups which precede of follow it. The notation has been simplified with this end in view.

When a +, - or → sign is recieved, the translation process of the previous group is made before the symbols of the present group are taken in. Neglecting for the moment this activity we follow the action of the machine in reading the symbols after the opeßation sign. As each symbol corresponding to each algebraic variable is recieved it is translated into the appropriate address which has been ø/ø previously defined and placed into storage locations for future use. When the end of the group is reached (shown by blank tape or the operation symbol of the next group) the translation proper starts.

The first thing that is looked for is for the precence of multiplication, shown by the existence of two algebraic variables after the operation sign. If this shows that multiplication is called for the second variable's address is used to form an order for setting the multiplier register if this hasnot already been set similarly by a previous term. Then the address of the first ø/ø/ variable is used for the multiplication order.

The case of simple addition or subtraction is dealt with by combining an add or subtract function to the address of the algebraic variable.

A group denoting transfer of the content of the accumulator to a storage location is synthesised in a manner similar to the add or subtract term.

I have simplified the description somewhat, in that I have not described how the translation takes advantage of some

of the orders availible in the Manchester code such as the orders calling for the transfer from the store to the a accumulator so that the previous content of the accumulator is discarded. Description of all the tricks would ~~it~~ take too long.

Nor have I described the way in which the notation is used to specify the use~~x~~ of the B tube. The use of the B tube more than anything else makes this type of notation easy of use, since on does not have to specify a notaion for the alteration of orders in the accumulator.

The B tube is use in the LOOP and REPEAT notation shown in the example, Certain letters of the alphabet are alloted soley to the description of B modification They appear on the written sheet as suffices; terms with suffices are B modified so that they will have added to them the number in a certain B line. AS an example consider the programme for the addition of the numbers contained in the consecutive storage locations beginning at the storage location denoted by x .

LOOP 10r

$+x_r$

REPEAT r

In this example the addition order will be made up with B digits calling for modification by the content of B7. The loop will commence with the number 10 in the B tube and will be repeated till this numb r has been progressively reduced to zero.

My experience of the use of this method of programming has been rather limited so far but I have been much impressed by the speed at which it is possible to make up programmes. and the certainty of gaining correct programmes. Most errors so far in programmes so constructed have been errors in the

mathematics rather than errors in the construction of the programmes. The most important feature I think is the ease with which it is possible to read back and mentally check the programme.

And of course on such features as these will the usefulness of this type of programming be judged.