UNIVERSITY OF MELBOURNE

COMPUTATION LABORATORY

PROGRAMMING MANUAL

for the Automatic Electronic Computer

CSIRAC

This production by the staff of the Computation Laboratory is based upon papers by T. Pearcey and G. W. Hill

TABLE OF CONTENTS.

PROGRAMMING MANUAL FOR CSIRAC

CSIRAC is a general-purpose automatic electronic digital computer. The word electronic indicates that it uses electric circuits and valves. 'Digital' means that numbers are represented in the machine by sets of discrete entities (actually trains of pulses), analogous to the familiar representation of numbers by decimal digits; in contrast there are 'analogue' machines in which each number is represented by a single physical quantity, such as a voltage or angle of rotation, capable of continuous variation. 'Automatic' means that the machine performs extended calculations under the control of programmes stored in advance in the machine; the human function is to make the programme, feed it into the machine, and then start the machine, which then proceeds by itself. Finally 'general purpose' means that a programme can be constructed whereby the machine will perform any determinate calculation at all, and many logical operations also, provided the programme and its data can be fitted into the stores of the machine.

Electronic computers are popularly called electronic brains, and it is true that they possess a few quasi-human attributes: they have functions analogous to memory, to the obeying of commands, and to the power of choosing between alternative courses of action in accordance with determined criteria. " They are intended to carry out any definite rule of thumb process which could have been done by a human operator working in a disciplined but unintelligent manner" (Turing). The analogy leadsto the use of terms such as 'memory' and 'command' and makes their intended meanings clear.

To construct the programme for a desired calculation, i.e. to 'programme the calculation', means to draw up a set of commands, each of which calls on an operation that the machine can perform and which together will do what is required. For this purpose the programmer needs to know what are the chief components of the machine and the broad lines on which its operation is organized, but he needs no detailed physical knowledge. It is on the whole with these broad lines that Chapter I is concerned, but a certain amount of detail is important; since it is difficult to fix the boundary between the necessary and the unnecessary, the reader is advised to take the chapter first in his stride, and to refer to it subsequently as he finds necessary.

Select references.

General: M.V.Wilkes, Automatic Digital Computers (Methuen, 1956)

For CSIRAC: T.Pearcey and G.W.Hill, Australian Journal of Physics, 6, 316-356 (1953) and 7, 485-504 (1954).

CHAPTER 1. Binary arithmetic. Organization of the machine.

1.1 Representation of numbers. Computer-arithmetic.

(i) Numbers are stored in the machine as trains of pulses travelling round closed circuits (fig.1). Part of any such circuit is a column ('delay-line') of mercury, and here the pulse is a pressure-pulse; part is a wire, where the pulse is in voltage or current; and there are intervening valves where the pulse is a burst of electrons. The conversion between the electronic and mechanical forms is made by piezo-electric crystals.

The spatial pattern of such a train of pulses while in the mercury column can be represented graphically (fig.2);

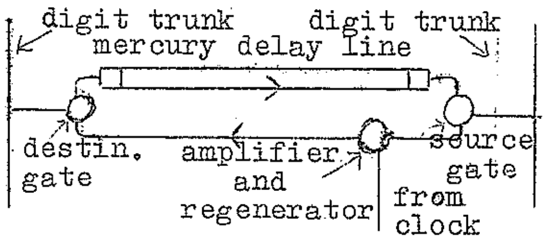


Fig. 1.

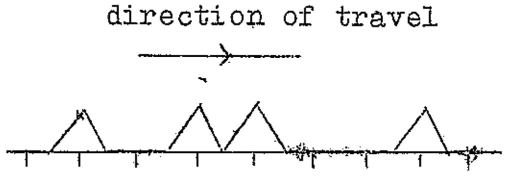


Fig. 2.

the spaces indicated by the scale are equal (actually about $\frac{1}{2}$ cm.), and for a pulse-train covering n of them there are 2^n different pulse-patterns according to the alternation of pulses and gaps. The same graph represents also the succession-in-time with which pulses and gaps pass any definite point of the circuit; the rate of passage is about 3.3×10^5 per sec.

To see how such pulse-trains can represent numbers we recall first that the decimal symbol for a number is a string of digits in which there is a position-value determined by the placing of a decimal point; e.g. 5029·6 stands for $5 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 9 \times 10^0 + 6 \times 10^{-1}$. Each digit can have any of the values 0, 1, ... 9, and there are just ten of these values because the radix is the number ten; the essential fact is that in this way we can get a decimal representation of <u>any</u> positive number, and for terminating decimals the representation is <u>unique</u>. Now there is a similar representation with any of the integers 2,3, ... as radix in place of 10; and if we choose 2 as the radix there are just the two possible digit values 0, 1. The system with radix 2 is called the binary system or scale. For example the symbol

10110.01 (binary)

represents the number $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$ (= 22·25 in the decimal representation). Such a binary symbol is a pattern of 1's and 0's which is the same as the pattern of a related pulse train, in which to each 1 corresponds a pulse and to each 0 a gap. By means of this correspondence, therefore, a pulse-train can be taken to represent a number. The representation is however not quite perfect, because there is nothing in the pulse-train which answers to the 'point' in the binary number symbol. This is no disadvantage; indeed it is an advantage because the same pulse-train can be taken on different occasions to represent numbers of quite different orders of magnitude.

We write a number-symbol with the most significant end on the distinction between right and left corresponds in the machine the direction of travel of the pulse-trains: they travel with their least significant or 'bottom' end leading, or passing any fixed point first in time.

(ii) Arithmetical operations are conducted in the binary representation on the same principles as in the decimal representation. It is however the machine and not its user that does the work in this way. The user will occasionally require to convert a number from decimal to binary representation, or vice versa, and he should gradually become familiar with the binary symbols for the integers 0, 1, ... 30, 31; these are given in Table 3 at the end of this chapter, but can be worked out mentally by partitioning the integer into such of the components 16, 8, 4, 2, 1 as it contains, e.g. 21 = 16 + 4 + 1 = 10101 (binary). The user should also understand the principles whereby addition and subtraction are conducted in the binary system.

To add two binary numbers the procedure is analogous to the familiar process for decimals and is founded on the 'one-digit addition table':

0 + 0	gives	0	with	0	to	carry
0 + 1	''	1	''	0	''	''
1 + 0	''	1	''	0	''	''
1 + 1	''	0	''	1	''	''

(The last entry, for example, represents the fact that $1 \times 2^a + 1 \times 2^a = 0 \times 2^a + 1 \times 2^{a+1}$.) An example using this process is

$$\begin{array}{r} 1\,1\,0\,1 \\ +\,1\,1\,0\,1 \\ \hline =1\,1\,0\,1\,0 \end{array}$$

Subtraction is similarly based on a one-digit table involving 'borrowing' in place of 'carrying'.

(iii) The machine has a mechanism, of which the principle will be indicated later, for carrying out on two pulse trains an operation which corresponds to the binary addition operation and thus generating a 'machine-sum' pulse train; and similarly for subtraction. But the machine process is in an important respect different from the arithmetical one. The registers of the machine hold only pulse-trains of a standard length (20 digits in the case of CSIRAC); a carry-pulse from the left-most or 'top' position is simply lost (as with a desk calculating machine), and a 'borrowed' pulse in the top position is nowhere paid back. Suppose for example that to a pulse-train consisting entirely of gaps (0 0 0 ...) we repeatedly machine-add a pulse-train having a pulse in the second position from the left and gaps elsewhere (0 1 0 0 ...). The successive results have the representation

```
(1) 0 0 0 ...
(2) 0 1 0 ...
(3) 1 0 0 ...
(4) 1 1 0 ...
(5) 0 0 0 ...
(6) 0 1 0 ...
   ............
```

where the 5th, 6th, ... entries are the same as the 1st, 2nd, ... This shows that we must properly take the arithmetical significance of the top pulse as ambiguous: a gap in this position can represent any even multiple of 2^a and a pulse can represent any odd multiple of 2^a, where 2^a is the position-value of the top position. In number-theoretical terms, machine-addition corresponds preceisely to arithmetical-addition-mod 2^{a+1}; in geometrical terms it corresponds in an obvious way to the addition of angles.

In practice a gap in the top position of a pulse-train always represents 0, and a pulse in this position always represents $+ 2^a$ or $- 2^a$, the posibility of other representations being excluded by the way calculations are programmed. Which of these representations is correct will depend upon the operation leading to the pulse-train. In the preceding table we have supposed line (3) to be generated by adding lines (1) and (2), and then the top '1' in line (3) must represent $+ 2^a$; and similarly for line (4)[1]. But the same table could be generated by starting with line (6) and repeatedly machine-subtracting 0 1 0 .... Then clearly line (4) would stand for $-2^a + 2^{a-1}$ $(= - 2^{a-1})$, and line (3) for -2^a, so the top '1' in these lines would stand for -2^a.

So far as machine-addition and subtraction are concerned this ambiguity can be left unresolved, but for machine-multiplication a choice must be made, and a convention must be adopted regarding the position of the binary point. The standard convention is that <u>the point lies to the immediate right of the top position</u> (so that $a = 0$) <u>and a pulse in the top position represents -1.</u> Then any number x that can be represented in the machine (on this convention) lies in the range $-1 \leq x < 1$, and the product xy of two such numbers lies in $-1 < xy \leq 1$. The machine has a mechanism for producing from the pulse-trains representing x and y a pulse-train that correctly represents xy, on the standard convention, in all cases except that where $x = y = -1$.[2] When the product is required of two numbers that are represented in the machine on some convention other than the standard one (for example with the point on the extreme right, so that the numbers represented are integers), the true product can be deduced from the 'machine product' by suitable programming; the detail of this will be shown in Chapter 2.

To the preceding statement of the standard convention we should add (as has been implied above) that <u>a pulse in the position places to the right of the top one always represents $+ 1 \times 2^{-r}$, and a gap in any position represents 0.</u> Accordingly, to find the representation of a negative number $x (-1 \leq x < 0)$ the rule is to express it in the form

$$x = - 1 + y,$$

where necessarily $0 \leq y < 1$. This gives a pulse in the top position (representing -1) followed by the pulse pattern that represents the positive number $0.y$. This is analogous to the standard representation for negative logarithms, where only the characteristic is negative. For example

$\frac{1}{2} = 0.1$ (binary), represented by the pulse train 0 1 0 0 ....
$-\frac{1}{2} = -1+\frac{1}{2} = \bar{1}.1$ (binary), " " " " 1 1 0 0 ....
$-\frac{1}{8} = -1+\frac{7}{8} = \bar{1}.111$(binary), " " " " 1 1 1 1 0 ....

The top digit in a pulse train is called its <u>sign-digit</u>. On account of its importance in programming there is a machine-operation for isolating it, which is analogous to its scrutiny by a human operator.

The convention regarding the position of the binary point which is on any occasion adopted can be made explicit by naming the digit position which carries unit weighting; the digit positions,

---

(1) In actual programming a test would be included for '1' as top digit, and the programme would not proceed to the formation of line (5). The desired operation would be handled by representing numbers at double length; see Chapter 6.

(2) In the exceptional case the 'machine-product' is a pulse in the top position, which of course here properly represents +1; and this

in decreasing order of significance, are called $p_{20}, p_{19}, \dots p_1$. Thus xp_{20} means a number x represented on the standard convention, which requires $-1 \leq x < 1$; xp_1 means a number x represented with the point to the immediate right of the bottom position, which requires that x be an integer.

(iv) It is desirable to have a symbolism for a pulse-train-in-itself which is more compact than the saw-tooth patterns or strings of 0's and 1's previously used. The method we shall adopt is the following: Write the '0,1' symbol for the train, and divide it into fives; for the pulse-trains in CSIRAC the length is 20, so they fall into four such fives. Then replace each five by the decimal symbol for the integer of which it is the binary representation. For example the pulse-train

01001 10011 00110 11011

has the symbol (9,19,6,27). This is called the 32-scale representation, because if we put the binary point to the right of the last group the whole pulse-train is the binary representation of $9 \times 32^3 + 19 \times 32^2 + 6 \times 32 + 27$. (In this representation a 1 in the top position is treated as positive, so that a pulse-train such as 10101, 00000, 00000, 00001 has the symbol (21,0,0,1).)

It is occasionally necessary to find the binary representation of a fraction, or (as is equivalent) the 32-scale symbol for the pulse-train that represents this fraction on the standard convention. First suppose the fraction x positive, and let

$$x = \frac{a_1}{2} + \frac{a_2}{2^2} + \dots\dots , \quad (a_1, a_2, \dots = 0 \text{ or } 1).$$

The top digit of the pulse train is 0 and the next four $a_1, \dots a_4$ correspond as a group to the component $\frac{8a_1+4a_2+2a_3+a_4}{16}$ of x; the next five correspond to the component $\frac{16a_5+\dots+a_9}{16x32}$; and so on.

Hence the procedure is to express x in the form $\frac{b_1}{16} + \frac{b_2}{16x32} + \frac{b_3}{16x32^2} + \frac{b_4}{16x32^3}$ where $0 \leq b_1 < 16$ and $0 \leq b_2, b_3, b_4 < 32$, and the required symbol is (b_1, b_2, b_3, b_4). For a negative x we put it in the form $-1+y$, find the symbol for y, and add (16,0,0,0), which represents -1.

1.2. The electronic performance of arithmetical and logical operations.

A brief indication will be given of principles whereby arithmetical and logical operations can be performed by electronic means; the programmer however does not need knowledge in this matter, and he can skip this Section.

It is clear that extended additions and multiplications will be possible provided we can get a physical realization of the one-digit addition table shown in §1.1(ii), and we shall indicate a method of getting this by means of triode valves.

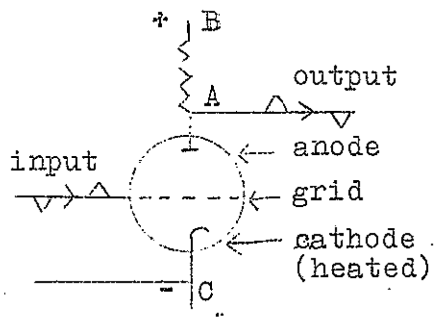


Fig.3: Triode valve

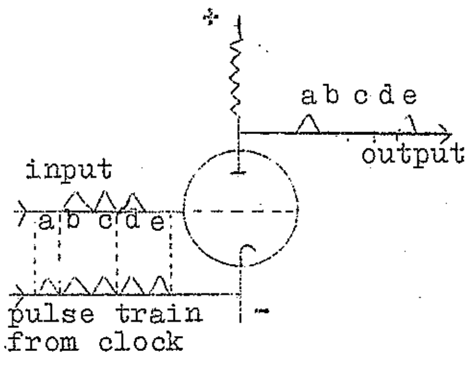


Fig.4: Not-gate

Suppose a triode valve to be at first in a steady state with the grid a few volts positive to the cathode C, so that a steady current flows through from B (which is maintained say 100 volts positive to the cathode) to C. Then if a positive-going voltage pulse comes on to the grid the current is increased, so the voltage at A falls and there is a negative-going output pulse; and a negative-going grid pulse gives similarly a positive-going output pulse. A pulse on the cathode has the same output effect as a pulse of the opposite sense on the grid, and for pulses of the same sense and magnitude simultaneously on both grid and cathode the output effects cancel. Hence (fig.4) if a regular succession of positive-going pulses is supplied to the cathode from a 'clock' a synchronized input-train of positive-going pulses on the grid will lead to a complemented output train: symbolically, a 1 input gives a 0 output, and a 0 input gives a 1 output. This arrangement is called a 'not-gate' because the output is in an evident sense the negation of the input.

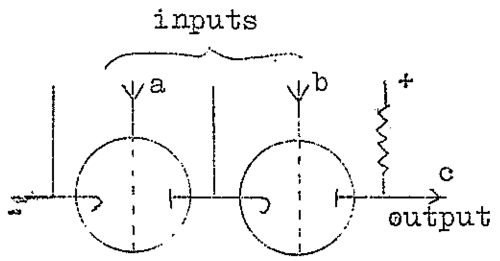


Fig.5: And-gate

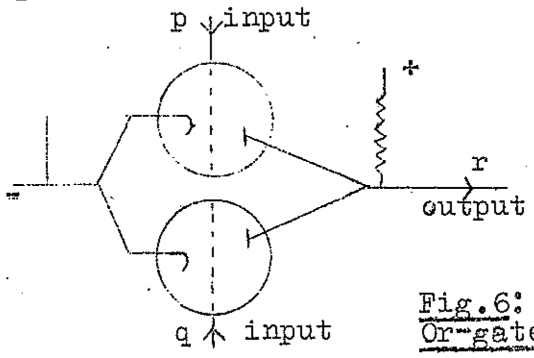


Fig.6: Or-gate

The arrangement shown in fig.5, in which for each valve the grid (unless pulsed) is maintained slightly negative to the cathode, will give an output pulse c only if positive-going pulses arrive simultaneously at a and b. Symbolically the values of c corresponding to the possible values of a and b are given by the table

a	b	c
0	0	0
0	1	0
1	0	0
1	1	1

and since this is the truth-table for the logical proposition 'a and b' the arrangement is called an 'and-gate'. Similarly for fig.6 there will be an output pulse at r if there is an input pulse at either p or q (or both), the value table is

p	q	r
0	0	0
1	0	1
0	1	1
1	1	1

Referring now to the one-digit addition table for a + b of §1.1 (ii), it is seen that an and-gate with a,b as inputs gives for output the correct value for the carry-digit. To obtain the correct sum-digit we must combine a number of operations, as shown in the logical table

a	b	a and b	not(a and b)	a or b	[not(a and b)] and [a or b]
0	0	0	1	0	0
1	0	0	1	1	1
0	1	0	1	1	1
1	1	1	0	1	0

Physically, this is achieved by using the pulse-states a,b as inputs for an and-gate (column 3) and also for an or-gate (column 5), putting the output from the and-gate through a not-gate (column 4), and finally by another and-gate achieving the state shown in column 6.

One further physical arrangement is needed. The successive digits of the pulse-trains appear on the conductors of the machine in a uniform time-succession, of interval T, say (actually about 10^{-6} second). The carry-digit from any one-digit addition has therefore to be used in a further addition at an interval T later than it was generated, which requires that it be put through a 'delay'. In principle this could be done by means of a short mercury delay-line, but the short delay that is actually required is more conveniently achieved by purely electrical circuitry that will not here be described.

By passing a pulse-train through a not-gate it is complemented, e.g. 011100 input gives 100011 output. If the pulse-train 000001 is now machine-added to the output the final output is 100100 which represents the negative of the original input: the two when machine-added gives zero. To add the final output is thus equivalent to subtracting the original input, and it is thus that machine subtraction is performed.

1.3 The electronic performance of commands.

Commands are represented in the machine by pulse trains. An indication will now be given of mechanism whereby the appearance of such a pulse-train on the conductors of the machine can lead to the performance of a determinate operation, and how trains of different patterns can lead to different operations.

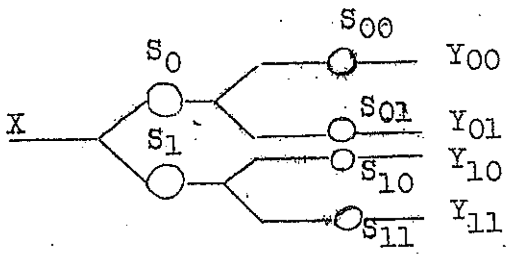
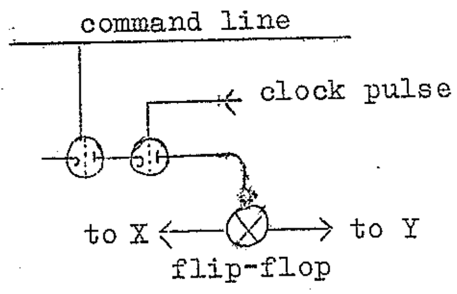


Fig.7.



detail of each gate S.

As a set of operations that is sufficiently typical we take the opening of a conducting path from a point X to one of four points Y_{00}, Y_{01}, Y_{10}, Y_{11}. In Fig.7 the circles S_0,...S_1, represent switches or gates of some sort, and it has to be arranged that of the pair S_0, S_1 one is opened and the other closed, and similarly for the pairs S_{00},S_{01} and S_{10},S_{11}. The 'command' pulse-train that is to secure this consists of two pulses p_2p_1, each of which may be 0 or 1, and it is to be arranged that this pulse-train shall open the path from X to $Y_{p_2p_1}$.

Suppose first that each S is an and-gate, of which one input terminal is connected to a 'command-line' on which the pulses p_1, p_2 will appear: first p_1, and then p_2 after an interval T. The connections to S_1, S_{01} and S_{11} are direct, but those to S_0, S_{00}, S_{10} are made through not-gates. For each of the gates $S_{00}, S_{01}, S_{10}, S_{11}$, the second input terminal is connected to a 'control-line' to which a pulse is sent from a 'clock' at the same time as the pulse p_1 appears on the command-line; and for S_0, S_1 the second input terminal is connected to another control-line, to which a pulse is sent contemporaneously with the appearance of p_2, i.e. at an interval T later than p_1. The effect is that if the pulse p_1 is a 1 (or 0) it will open the gates S_{01}, S_{11} (or S_{00}, S_{10}), but be without effect on the gates S_0, S_1; while p_2 will open S_0 or S_1 without affecting S_{00}, S_{01}, S_{10} or S_{11}. For each of the four values (00,01,10,11) of p_2p_1 the command pulse-train will therefore activate the gates on the path from X to Yp_2p_1.

It is however desired that the gates on this path shall remain simultaneously open for a time, in order that a pulse-train (representing a number N) which appears at X after all the gates have been opened may be transferred to Yp_2p_1. This is secured by using the output from each of the and-gates to trigger a 'flip-flop' a flip-flop, which will not here be described, is an electronic analogue of an ordinary ceiling-switch, which after being triggered remains on or off until it is again triggered.

The points $Y_{00}, Y_{01}, \dots$ may be input terminals to 'memory-cells' where the pulse-train N which entered at X is to be stored. Or one of them may be an input terminal to an adding unit, which already contains a number-train N' and to which the number-train N is to be added. And so on.

1.4. <u>The elementary operations and command-structure of CSIRAC</u>.

(i) The main store or 'memory' of CSIRAC consists of 1024 parts or 'cells' each of which holds one 20-digit pulse-train which we shall call a 'word'. While any specific calculation is in progress some of these words will represent numbers and others will represent commands. The cells are connected to a main conductor (bus-bar or digit trunk) through a 'decoding tree' as illustrated in fig. 7. Since $1024 = 2^{10}$, a command-train of length 10 digits will suffice to open the path from the digit-trunk to any chosen one of the 1024 cells. This command-train, as a string of ten 0's or 1's, will be the binary symbol for one of the integers 0,1,... 1023, and we think of the cells as correspondingly numbered: the command-train which represents the integer n serves to open the path from the digit trunk to cell number n.

There are also 20 arithmetic registers, called $A, B, C, H, D_0, D_1, \dots D_{15}$, each of which holds one word, and these are connected to the digit trunk through various 'source-gates' and 'destination-gates'. For example, register C has three source-gates: (1) a gate called (C) through which the word $p_{20}p_{19}\dots p_1$

(2) a gate called s(C) through which the sign-digit p_{20} only of the word in C can be read out; and (3) a gate called r(C) reading through which gives the word 0 $p_{20} \dots p_2$, on the digit trunk. The symbols (C), s(C), r(C) are also used, without confusion, for the words that are read out through these gates; r(C) stands in an obvious way for 'right shift of the word in C'. For register C there are also three destination gates: (1) a gate called C through which a word can be read from the digit trunk into the register C, thereby erasing or 'over-writing' the word which may originally have been in the register; (2) a gate called +C, reading through which adds the entering word to the word originally in the register and leaves the machine-sum in the register; and (3) a gate called −C, reading through which subtracts the entering word from the word originally in C.

There are also 3 input-registers with source gates only, 2 output registers with destination gates only; an auxiliary magnetic-disc store containing 4096 cells with source and destination gates; a sequence-register and interpreter with source and destination gates; a loud-speaker with a destination gate; and finally 3 source gates for reading constants from the control unit and a destination gate connected to a stop switch.

In all there are 32 source-gates and 32-destination gates. Since $32 = 2^5$, a command train of length 5 digits will suffice to select a source-gate; this command train will be the binary symbol for one of the integers 0,1,... 31, and the source-gates are correspondingly numbered; e.g. the command train 10000 opens the source gate r(C), which is accordingly no.16. The destination-gates similarly are numbered 0,1,...31 according to the patterns of the 5-digit control trains that open them; for example the command train 01111 opens the gate +C, which accordingly is no.15. A full list of the sources and destinations is given in Tables 1,2 at the end of this Chapter.

(ii) Each command to CSIRAC calls for the opening of one source-gate and one destination gate, and perhaps also for a connection with one memory cell.

A full command-word consists therefore of 20 digits and is functionally partitioned into 3 groups indicated:

$$\underbrace{p_{20}\,p_{19}\cdots\overbrace{p_{14}\cdots p_{11}}^{\text{D-cell no.}}}_{\text{memory-cell no.}}\quad \underbrace{p_{10}\cdots p_{6}}_{\text{source-no.}}\quad \underbrace{p_{5}\cdots p_{1}}_{\text{dest.no.}}$$

The digits, $p_{20} \dots p_{11}$, taken by themselves as the binary symbol for an integer, give the number of the cell that will be called by the command, as explained above, and constitute the address part of the command. The digit-group $p_{10} \dots p_6$ similarly specifies the source that will be called by the command, and the digit-group $p_5 \dots p_1$ specifies the destination that will be called. The sub-group $p_{14} \dots p_{11}$ of the address serves also to specify which of the 16 D-cells (if any) is called.

Accordingly, when a command-word is represented in the 32-scale as explained in §1.1 (iv), the first two numbers give the number of the memory-cell that is called, the third is the code-number of the source and the fourth is the code-number of the destination. For example, in the command (2, 18, 0, 14), the source is the main store (code no. 0) the destination is transfer to register C (code no. 14), and it is store-cell no. (2, 18) = $2 \times 32 + 18 = 82$, which is called. The obeying of this command register C. When writing the command in a programme, however, we use a notation in which the meaning of the command springs immediately to the eye, viz.

$$(2,18\ M) \longrightarrow C$$

or more compactly[(1)]

$$(2,18) \longrightarrow C\ ,$$

where the brackets round 2,18 signify 'contents of'.

Similarly in the command which is coded (2, 18, 14, 0) the source is register C (code no.14) and the destination is cell no. (2,18) of the main store. The effect of the command is to write ~~read~~ the word currently in register C into this store-cell, thereby over-writing (annulling) the word previously in this cell; and the meaningful notation for the command in programme-writing is

$$(C) \longrightarrow 2,18\ M \qquad 2,18\quad C\quad M$$

or more compactly[(1)]

$$(C) \longrightarrow 2,18,$$

where again the brackets round 'C' signify 'contents of'.

In an actual programme most commands call for transfers between the arithmetic registers and do not call for the main or auxiliary store as either source or destination. In such a case the address part of the command-word, as a set of pulses, is vacuous (unless either source or destination involves a D-cell), i.e. the effect of the command will be independent of what this digit-group may actually be. For simplicity the address-group is nearly always taken to be (0, 0), so that for example the command to add the word in register C to the word in register A is coded as the pulse-train (0,0,14,5) and is conventionally written

$$(C) \xrightarrow{+} A\ ;$$

the pulse train (m,n,14,5) would have exactly the same effect,, when obeyed as a command, whatever the 'values' of m,n.

(iii) It is not necessary for the reader to learn immediately the list of sources and destinations; he will acquire the knowledge automatically as he studies programming techniques. Still less need he learn immediately the code-numbers of the sources and destinations, which are important only at the stage of checking and running programmes on the machine. However, a general knowledge of the way commands are coded is immediately useful because it makes clear what commands are possible and what are impossible. For example a transfer from one memory cell to a different one is impossible; the impossibility arises of course from the way the machine is built, but it can be seen from the impossibility of coding such an operation on the scheme that has been explained: the command-word structure has no room for two addresses in addition to its source and destination components. Similarly a transfer from one D-cell to another D-cell (which would be quite a useful operation) is impossible: the command-word structure has room only for one D-address. It is however possible to call upon the same D-cell as both source and destination, or upon a D-cell and a <u>consistently-numbered</u> memory-cell (i.e. if the digits $p_{14} \ldots p_{11}$ of the memory-cell number give also the D-cell number). For example the commands that are written

$$(D_5) \xrightarrow{+} D_5\quad ,\quad (D_0) \xrightarrow{-} D_0,\qquad (8,21) \longrightarrow D_5$$

(1) To drop the symbol M is a convention, which saves a little

and coded (0,5,17,18), (0,0,17,19) (8,21,0,17) are possible and often useful; (the last is possible because $21 \equiv 10101$ and $5 \equiv 00101$, or more shortly $21 \equiv 5$, mod 16).

1.5 Organization of CSIRAC.

For calculation on CSIRAC a suitable programme of commands must first be drawn up, and these commands must then be stored (by a process which we shall not here explain: see Chapter 4) in coded form in memory cells of the machine. For the performance of the calculation the machine must then be made to select and obey the commands in the proper order. The normal procedure is to store the commands in successively numbered cells in the order in which the commands are to be obeyed; the programme will then be correctly performed provided the machine is provided with a mechanism which arranges automatically that when the command in cell no. m has been obeyed the next command is selected from cell no. (m+1). But in almost any programme this regular sequential selection of commands must be broken, so that when the command in cell m has been obeyed the next is to be selected from cell r where $r \neq m+1$.

To secure the selection of commands in the proper order the machine is provided with a sequence register which holds what is shortly called 'the number of the next command'. More fully, at the instant when any command has been performed the sequence register holds the number of the cell where is stored the next command to be obeyed. One of the machine operations, which is automatically performed, is to add 1 to the number in the sequence register during each cycle of command-performance; this secures that commands are normally taken in order from sequentially numbered cells. When it is necessary to break the sequential order of selection, the procedure is to insert in the programme an order that the number in the sequence register is to be changed, and the machine is so built that it can obey this order.

A schematic diagram of CSIRAC is shown on p.13. In the preceding we have referred, explicitly or by implication, to all the components, but the following may be added: (i) The interpreter register includes a delay-line circuit which holds the command (or more strictly the pulse-train representing the command) currently to be obeyed, and the apparatus whereby this is decoded and the appropriate gates are activated. The interpreter also has other functions which will be explained in Chapter 2. (ii) The control unit has as its fundamental component a crystal oscillator which acts as a clock to count elapsed time. The primary unit is the pulse-interval, about 3.3×10^{-5} sec; twenty of these make a minor cycle, which is the time taken for a 20-digit word (pulse-train) to pass any fixed point in the machine. Sixteen minor cycles make a major cycle; this unit is important because the main store is arranged physically in mercury delay lines (each with associated circuitry) each of which holds 16 words, so that it is equivalent to a set of 16 consecutively-numbered memory cells; a major cycle is hence the maximum 'access-time' for a memory-cell. The control unit sends pulses to the gates and decoding apparatus at regular intervals as required during the cycle of operation, and provides the pulses at the proper instants for the source gates p_1, p_{11}, p_{20}.

The obeying of one command involves four distinct stages, each of which lasts for a major cycle when the machine is set to 'normal speed', giving a total command-time of about 0.004 sec; but there is an alternative setting to 'speed-up' in which the average command-time is about 0.0025 sec. The stages of this computer cycle are:

(i) The number n currently in the sequence register, which is the number of the memory-cell holding the command which is next to be obeyed, is sent to a decoder, and the path from cell n to the digit trunk is opened except for a main gate.

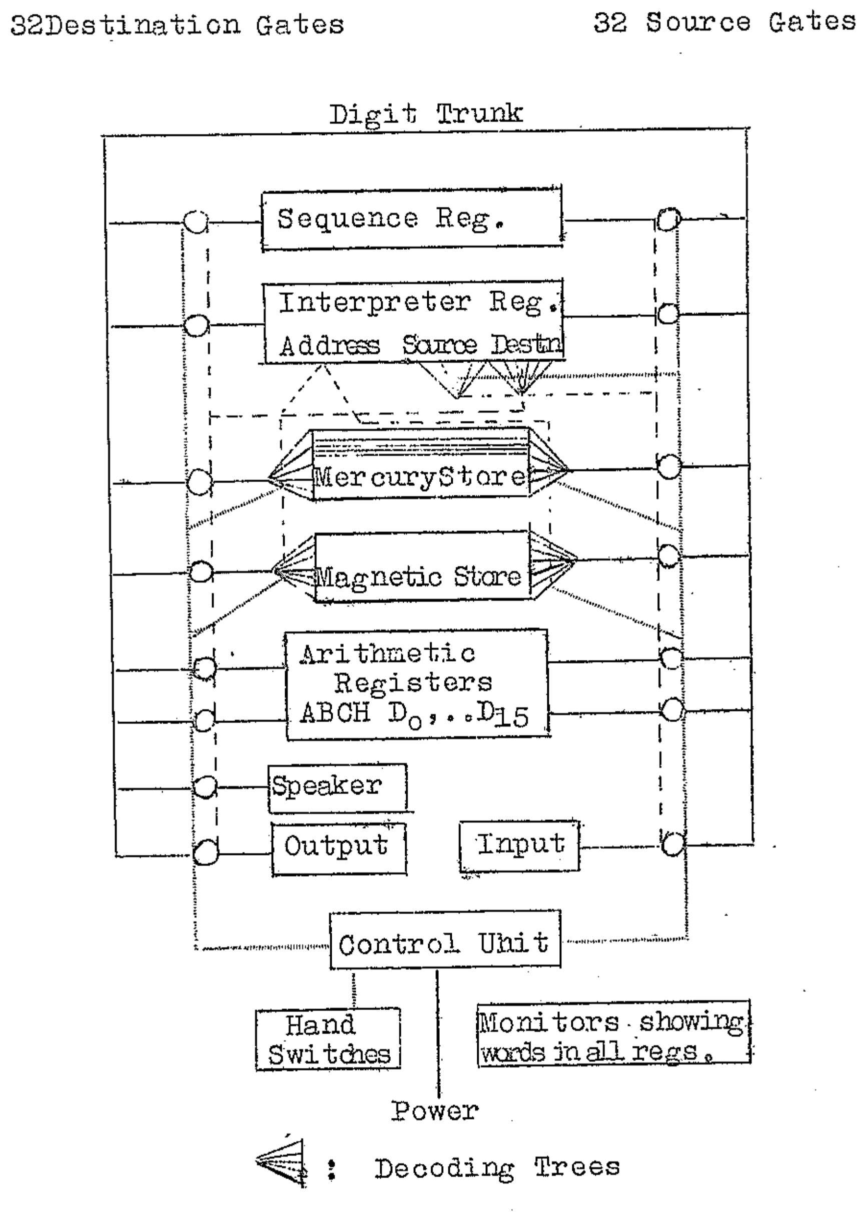
(ii) This main gate is opened, and also the destination gate from the digit trunk to the interpreter is opened, so that the word in cell n is transmitted to the interpreter.

(iii) The word, treated now as a command, is decoded by the interpreter and the appropriate source and destination gates are prepared to open. Also 1 is added to the number in the sequence register.

(iv) The source and destination gates are opened, and the transfer called for by the command takes place.

The machine then immediately returns to stage (i) of the next computer cycle, and what happens will of course be governed by the number (normally n+1) now in the sequence register.

Schematic Diagram of CSIRAC.



CSIRAC COMMAND CODE : SOURCE FUNCTION GATES

Code No.	Function	Symbol
0	Read out the content of the cell (mercury store location) $n (0 \leqslant n \leqslant 1023)$, indicated by the address digits p_{11}-p_{20}	(nM) or (n)
1	Read out the content of the input register (20 digits, p_1-p_{20}) and shift input tape	(I)
2	Read out the content of hand-set register No.1 (20 digits, p_1-p_{20})	(N_1)
3	Read out the content of hand-set register No.2 (20 digits, p_1-p_{20})	(N_2)
4	Read out the content of register A (20 digits, p_1-p_{20})	(A)
5	Read out in p_{20} position the most significant digit (0 or 1) of the content of register A	s(A)
6	Read out the content of register A divided by 2 (20 digits)	$\frac{1}{2}$(A)
7	Read out the content of register A multiplied by 2 (20 digits) (0 in p_1 position)	2(A)
8	Read out in the p_1 position the least significant digit (0 or 1) of the content of register A	p_1(A)
9	Read out the content of register A and leave it cleared to zero	c(A)
10	If the content of register A is non-zero transmit 1 in p_1 position, otherwise transmit 0	$\bar{z}$(A)
11	Read out the content of register B	(B)
12	Read out 1 in p_1 position if the most significant digit of the content of register B is unity otherwise read out 0	(R)
13	Read out the content of register B shifted to the right one place (0 in p_{20} position)	r(B)
14	Read out the content of register C	(C)
15	Read out the most significant digit of the content of register C	s(C)
16	Read out the content of register C shifted one place to the right (0 in p_{20} position)	r(C)
17	Read out the content of the location in register D indicated by the number $m (0 < m < 16)$, represented by the address digits p_{11}-p_{14}	(D_m)
18	Read out the most significant digit in the location in register D indicated by the number $m (0 < m < 16)$, represented by the address digits p_{11}-p_{14}	$s(D_m)$
19	Read out the content of the location in register D (indicated by the number $m (0 \leqslant m < 16)$, represented by the digits p_{11}-p_{14}), shifted one place to the right	$r(D_m)$
20	Read out a string of twenty 0's	(Z)
21	Read out the 10-digit content of register H in the position group p_1-p_{10}	(H_1)
22	Read out the 10-digit content of register H in the position group p_{11}-p_{20}	(H_u)
23	Read out the 10-digit content of the sequence register in the position group p_{11}-p_{20}	(S)
24	Read out 1 in the p_{11} position	p_{11}
25	Read out 1 in the p_1 position	p_1
26	Read out from the interpreter register the address part n (digits p_{11}-p_{20}) of the current command	(nK) or n
27	Read out the content of the magnetic store No.1 from the location $n (0 \leqslant n < 1024)$, indicated by the address digits p_{11}-p_{20}	(n_a)
28	As for Code 27 using magnetic store No.2	(n_b)

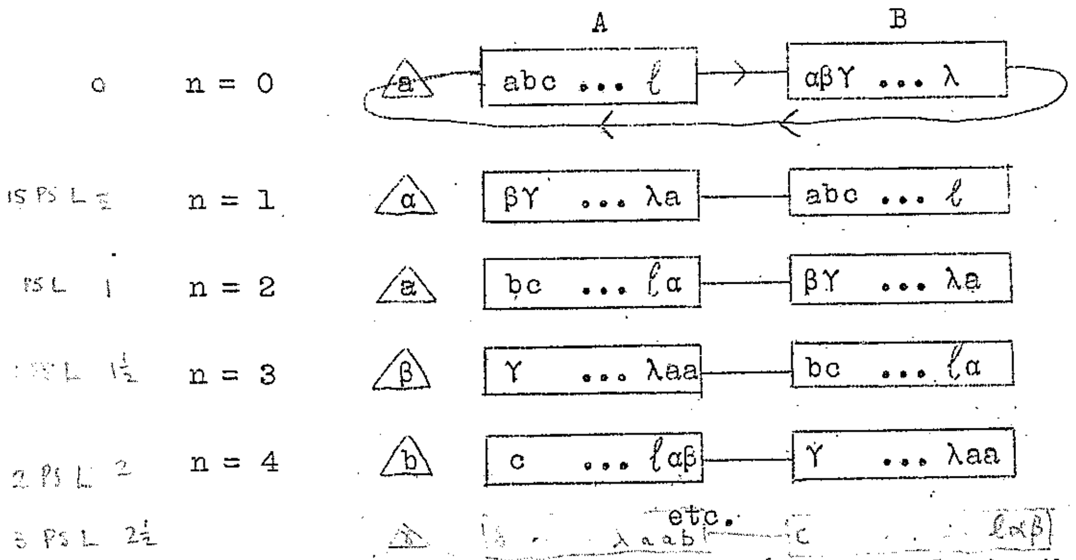
CSIRAC COMMAND CODE : DESTINATION FUNCTION GATES

Code No.	Function	Symbol
0	Substitute into the mercury store in cell $n(0 \leq n \leq 1023)$, indicated by the address digits $p_{11}-p_{20}$	nM or n
1	Null, i.e. without any effect	I
2	Substitute into the output register the logical sum of the digits received in positions p_1-p_5 and $p_{11}-p_{15}$ and print the corresponding character	O_t
3	Substitute into the output register the digits received in positions $p_{20}, p_{19}, p_1-p_{10}$ and punch on to tape	O_p
4	Substitute into the register A(20 digits)	A
5	Add into the content of register A and hold the sum	+A
6	Subtract from the content of register A and hold the difference	−A
7	Replace the content of register A by the digit by digit product of its content and the entering digits (i.e. conjunction)	$\cdot$A 0 0 / 0 1 and
8	Replace the content of register A by the digit by digit logical sum of its content and the entering digits (i.e. disjunction)	$\vee$A [illegible] 0 1 / 1 1
9	Compare, digit by digit, the content of register A with the set of digits entering, and in each digit-position place 0 or 1 according as the digits compared are the same or different	$\neq$A 0 1 / 1 0
10	Transfer the entering digit train into the loudspeaker	P
11	Substitute into register B	B
12	Substitute into register B, form the product of the content of B and register C (modulo 2) and add into the content of register A, retaining the lowest 19 digits of the product in B with a zero in the p_1 position of B	xB
13	On the command which is coded (16,2x-2,26,13) and written $16,2x-2(K) \longrightarrow L_x$, left-shift the content of registers A and B x places, for $1 \leq x \leq$ ~~8~~ 7. For x=1 an alternative coding is (0,0,31,13), written $p_{20} \longrightarrow L_1$. (For a full description of commands having destination no.13 see below.	L_x
14	Substitute into register C	C
15	Add into the content of register C and hold the sum	+C
16	Subtract from the content of register C and hold the difference	−C
17	Substitute into the location $m(0 \leq m < 16)$ of register D, indicated by the $p_{11}-p_{14}$ digits of the address	D_m
18	Add into the content of location $m(0 \leq m < 16)$ of register D, indicated by the $p_{11}-p_{14}$ digits of the address, and hold the sum	$+D_m$
19	Subtract from the content of location $m(0 \leq m < 16)$ of register D, indicated by the $p_{11}-p_{14}$ digits of the address, and hold the difference	$-D_m$
20	Null, i.e. without any effect	Z
21	Substitute into register H the digit group p_1-p_{10} of the entering number	H_1
22	Substitute into register H the digit group $p_{11}-p_{20}$ of the entering number	H_u
23	Substitute into the sequence register the digit group $p_{11}-p_{20}$ of the entering number	S
24	Add into the content of the sequence register the digits entering in the group $p_{11}-p_{20}$ and hold the sum	+S
25	Add 1 in p_{11} position into the content of the sequence register if either the p_1-p_{11} group or the $p_{15}-p_{20}$ group of the entering word is not entirely zero; add 2 if neither group is entirely zero	cS
26	Replace the content of the interpreter register by the group of digits entering. Hold the new content until the	+K

CSIRAC COMMAND CODE : DESTINATION FUNCTION GATES - cont.

Code No.	Function	Symbol
27	Substitute in the magnetic store No.1 into the location n($0 \leq n < 1024$), indicated by the address digits p_{11}-p_{20}	n_a
28	As for 27 but using magnetic store No.2	n_b
29	As for 27 but using magnetic store No.3	n_c
30	As for 27 but using magnetic store No.4	n_d
31	If one or more 1 digits received, stop the computer; do not proceed to the next command	T

Note on the left-shift operation. For the execution of this operation a single loop is first formed from the digits of the registers A, B together with a delay unit Δ of one-digit capacity into which originally the sign digit of A is copied. The 41 digits in this loop circulate in the standard feet-first (right-shift) sense, and on a command coded (16,n-2,26,13) or (16,14+n,26,13) the circulation is arrested in effect, after the lapse of n minor cycles, for n = 1,2,... 16. The state of the registers A and B, originally and at subsequent stages is as follows



At the arrest a digit is trapped in Δ, and only in the case n = 2 is this the redundant sign digit a ; in other cases a significant digit is trapped and the digit a is duplicated

The operation is thus nearly, but not quite, a cyclic shift of the digits originally in A and B, and when n is even it is more naturally taken as a short left-shift than a long right-shift.

When n is even the effect is to give the product of (A,B) by 2^n (except for the possible overcarry), and the digits that have entered at the lower end of B have the character of a rounding-error[x]. For an odd n, (A) and (B) become respectively the 2(B) and (A) of the preceding even n, and the operation then is unlikely to be useful.

BINARY SYMBOLS, TAPE-PRINT SYMBOLS, AND TELEPRINTER CODE

No.	Binary Symbol	Tape-print symbols				Teleprinter code	
		Source		Destination		Letter shift	Figure shift
0	00000	( )	M	M	M	A	0
1	00001	(I)	I	I	I	B	1
2	00010	(N_1)	NA	O_t	OT	C	2
3	00011	(N_2)	NB	O_p	OP	D	3
4	00100	(A)	A	A	A	E	4
5	00101	s(A)	SA	+A	PA	F	5
6	00110	$\frac{1}{2}$(A)	HA	−A	SA	G	6
7	00111	2(A)	TA	.A	c A	H	7
8	01000	p_1(A)	LA	vA	D A	I	8
9	01001	c(A)	CA	≠A	N A	J	9
10	01010	$\bar{z}$(A)	ZA	P	P	K	+
11	01011	(B)	B	B	B	L	−
12	01100	(R)	R	xB	XB	M	.
13	01101	r(B)	RB	L	L	N	)
14	01110	(C)	C	C	C	O	(
15	01111	s(C)	SC	+C	PC	P	i
16	10000	r(C)	RC	−C	SC	Q	j
17	10001	(D)	D	D	D	R	k
18	10010	s(D)	SD	+D	PD	S	∇
19	10011	r(D)	RD	−D	SD	T	φ
20	10100	(Z)	Z	Z	Z	U	ψ
21	10101	(H_l)	HL	H_l	HL	V	θ
22	10110	(H_u)	HU	H_u	HU	W	Ω
23	10111	(S)	S	S	S	X	Γ
24	11000	p_{11}	PE	+S	PS	Y	Π
25	11001	p_1	PL	cS	CS	Z	Σ
26	11010	(K)	K	+K	PK	∧	≡
27	11011	(a)	MA	a	MA	fig. shift [x]	
28	11100	(b)	MB	b	MB	letter shift [x]	
29	11101	(c)	MC	c	MC	line feed[x]	
30	11110	(d)	MD	d	MD	carriage return	
31	11111	p_{20}	PS	T	T	space	

[x] A space also is given on these calls.

CHAPTER 2. Programming: The elements

The performance of a calculation on CSIRAC falls into a number of stages that are relatively distinct: (1) Selection of method. For example, to solve an algebraic equation we could use Newton's method, or Graeffe's, or trial and error. The questions involved here belong largely to mathematics rather than to computing technique, but the choice of method may be influenced by the facilities peculiar to an automatic computer. One may for example prefer to use many repetitions of a simple process that can be easily programmed, rather than a few repetitions of a more complicated process. (2) Selection of tactics. In the detailed application of a method there will be many places at which minor variations are possible. For example, a triple product may be evaluated as a(bc) or (ab)c; answers may be extracted from the machine by typing or by punching paper tape; the number of signigicant figures retained may be more or less. There are many decisions of this sort to be made, some larger and some smaller; the larger ones merge into questions of method. (3) Drawing up the detailed programme. This is a breaking down of the calculation into a series of elementary steps each of which is a machine-operation. (4) Performing the calculation. The programme must be punched, in machine-code, on paper tape and fed into the machine, and the machine then must be correctly operated.

Our concern in this manual is mainly with stages (3) and (4), but a little as to (1) and (2) is said in Chapter 7.

The notation in which commands are written is strongly suggestive of the machine operations and is hence easy to remember; it has been illustrated in §1.4(ii).

The set of commands constituting a programme is written out and numbered in the order of the cells where they will be stored in the machine. In the simpler cases the numbers attached to the commands are those of the actual storage cells, but sometimes there is a constant to be added to the command-number to get the storage-cell number (see Chapter 5).

It is necessary to remember always that (i) when a word (or part of a word) is read out from a register or cell through a source gate it is a copy of the word which is taken; the word originally in the register normally(1) remains there ready for subsequent reading. (ii) When a word is read into (or substitued into or transferred to) a register or cell, the word originally there is erased or overwritten.

We shall show the sorts of command most commonly used, and the formation of simple sequences of them, by means of examples. Normally the pulse-trains that are operated on by the commands will represent numbers, on the standard or some other convention, and we shall call them numbers unless the context demands otherwise.

2.1. Substitution, addition, subtraction.

Ex.1. Substitute the number in cell 5,17 (i.e. cell number 177) into register C.

```
0     (5,17 M) → C
```

Ex.2. Subtract the number in D_5 from the number in D_1.

0 $\quad (D_5) \longrightarrow A$

1 $\quad (A) \xrightarrow{-} D_1$

The brackets here indicate 'contents of'; '$(D_5) \longrightarrow A$' is a contraction of 'move the contents of the place D_5 to the place A'. We use here register A as an intermediary, and the numbers originally in A and D_1 are lost in the course of the operation; but the latter could of course be regained by a similar addition of (D_5) to (D_1).

<u>Ex. 3.</u> <u>Add the number in cell 5,17 to the number in D_1</u>

$$0 \qquad (5{,}17\ M) \xrightarrow{+} D_1 \quad ^{(1)}$$

This can be done by one command because $17 \equiv 1$ (mod.16); see §1.4(iii)

<u>Ex. 4.</u> Place (minus the number in D_5) in register C or register B

0	$(C) \xrightarrow{-} C$; $(C)' = 0$	0	$(A) \xrightarrow{-} A$
1	$(D_5) \xrightarrow{-} C$	1	$(D_5) \xrightarrow{-} A$
		2	$(A) \longrightarrow B$

We cannot assume that C contains zero at the start, and command 0 is needed to secure this; the notation (C)' means 'contents of C after the operation'. For the second case, subtraction into B is not a machine operation, so we must proceed via a register having this facility; it is normal to use A for this purpose.

2.2. <u>Multiplication.</u>

<u>Ex.5.</u> <u>Cells p, q contain words that represent numbers on the standard convention. Find their product, assuming that they are not both −1.</u> The commands are

$$\begin{array}{ll} 0 & (A) \xrightarrow{-} A \\ 1 & (p\ M) \longrightarrow C \\ 2 & (q\ M) \xrightarrow{x} B\ . \end{array}$$

For machine multiplication one factor must first be placed in C (command 1). For numbers represented on the standard convention, the bottom (p_1) digit has the weighting 2^{-19}, and in the exact product of two such numbers the bottom digit will have the weighting 2^{-38}; so two registers are required to accommodate the exact product. The effect of command 2 is (i) the number (q M) called by the source is read into the register B, thereby erasing whatever word may originally have been in B, (ii) the product is formed with the number in C, the top 20 digits of it (starting with the sign digit) are <u>added to the number that was originally in A</u>, and the bottom 19 digits of the product (whose weightings run from 2^{-20} to 2^{-38}) are substituted into the 19 top positions of B; the bottom (p_1) position of B is left with 0 in it. Moreover the digits of the product all have positive weighting (as in the standard representation) except the sign-digit in A, which will represent −1 if the two factors are of opposite signs. Command 0 is required in order that the number that is in A when command 2 is obeyed may be zero.

If we desire the product correct to 19 binary places the command

$$3 \qquad (R) \xrightarrow{+} A$$

is added. This adds a 1 or 0 to (A) in the p_1 position (representing 2^{-19}) according as the top digit in B is 1 or 0, and

(1)

the error in the approximation (A)' to the product is then at most 2^{-20}.

The product (A) x (C) can be found by the one command

$$0 \quad c(A) \xrightarrow{x} B$$

where the source c(A) (code no.9) has the effect of clearing A as the number in it is read out. For (p M) x (q M) an alternative procedure is

$$\begin{array}{ll} 0 & (q\ M) \longrightarrow A \\ 1 & (p\ M) \longrightarrow C \\ 2 & c(A) \xrightarrow{x} B \end{array}$$

Ex.6. Put half the number in C into D_0. In the binary representation a positive number (on any convention as to the position of the point) is halved by right-shifting its set of digits one place in relation to the binary point, so if (C) > 0, $\frac{1}{2}$(C) = r(C). If (C) is a negative number x in the standard representation, put x = - 1 + y. Then

$$\tfrac{1}{2}x = -1 + \tfrac{1}{2}(1 + y), \qquad (1)$$

and $\frac{1}{2}(1 + y)$ = r(C); e.g. y = 0.101 gives x = (C) = $\bar{1}$.101 and the right shift of the pulse-pattern 11010... is 011010..., which represents $\frac{1}{2}(1 + y)$. The -1 which is to be added on the right of (1) coincides with the sign-digit s(C) of (C). Hence

$$\tfrac{1}{2}(C) = r(C) + s(C), \qquad (2)$$

and this is true also when (C) > 0 since then s(C) = 0. Hence the required commands are

$$\begin{array}{ll} 0 & r(C) \longrightarrow D_0 \\ 1 & s(C) \xrightarrow{+} D_0\,. \end{array}$$

When the binary point is taken to be in some position other than the standard one we can still represent a negative number x on the convention that the top digit only has negative weighting, provided |x| does not exceed the weight 2^a attached to the top position. A little consideration will show that, on this convention, formula (2) remains valid, i.e. if (C) gives x on the convention then r(C) + s(C) gives $\frac{1}{2}$x on the same convention.

For a number x in A the source $\frac{1}{2}$(A), no. 6, gives $\frac{1}{2}$x in one step, irrespective of the sign of x.

If the bottom digit p_1(C) of (C) is 1, $\frac{1}{2}$(C) as given by (2) is rounded down, i.e. the machine answer is less than the true answer by $\frac{1}{2}p_1$(C). An answer that is rounded up is given by

$$\begin{array}{ll} 0 & (C) \longrightarrow D_0 \\ 1 & r(C) \xrightarrow{-} D_0 \\ 2 & s(C) \xrightarrow{+} D_0\,. \end{array}$$

If (C) had been arrived at as an unrounded product the latter procedure would give $\frac{1}{2}$(C) with the smaller probable error.

Ex.7. If (A), (C) represent numbers x,y on the standard convention, find 2xy, supposing 2xy < 1. If A contains a number x on the standard convention, and $-\frac{1}{2} \leq x < \frac{1}{2}$, the order (A) $\xrightarrow{+}$ A gives (A)' = 2x. Moreover the digit pattern for 2x is the left-shift of the pattern for x. (For $0 \leq x < \frac{1}{2}$ this is evident. For $-\frac{1}{2} \leq x < 0$ put x = $\bar{1} + \frac{1}{2}$y, where $\bar{1} \leq y < 2$ so that

Then $2x = \overline{2} + y = \overline{1}.abc\ldots$, and the pulse pattern of this is the left-shift of that of x, since $x = \overline{1} + \frac{1}{2}y = 1.1\,abc \ldots$.)

Thus left-shifting and doubling are equivalent, provided the doubling can be done within register capacity; and in all cases, $A \longrightarrow A$ gives a left-shift.

If (A, B), as the result of a multiplication, represents a number at double length, the left-shift destination L (no.13) gives a doubling of this number with minimum loss of accuracy, since the top digit of (B) moves into the place vacated by the bottom digit of (A); and similarly for multiple left shifts up to the maximum number of them, eight, that can be called. The best solution of the problem is thus

```
0      c(A) —x→ B     ;   (A,B)' = xy
1      p20 —→ L1      :   (A,B)' = 2xy
2      (R) —+→ A
```

There is a note on the left-shift operation at the end of Chapter 1. The coding of commands calling for it is:

for 1 left shift	(0,0,31,13) ,	written	$p_{20} \longrightarrow L_1$
	or(16,0,26,13),	''	$16,0(K) \longrightarrow L_1$
for 2 left shifts	(16,2,26,13),	''	$16,2(K) \longrightarrow L_2$
for 3 left shifts	(16,4,26,13),	''	$16,4(K) \longrightarrow L_3$
. . . .	. . . .	. . .	. . . .
for 8 left shifts	(16,14,26,13),	''	$16,14(K) \longrightarrow L_8$

Ex.8. If x,y are integers represented as $(A) = xp_1$, $(C) = yp_1$, represent the product xy in the form xyp_1, assuming $|xy| < 2^{19}$. Or otherwise expressed,if x,y are represented in the machine on the convention that the binary point is on the extreme right, find their product on the same convention, assuming that it is within register capacity. (This is the usual convention for representing integers other than cell-numbers.)

Since $p_1 = 2^{-19}p_{20}$ we have on the standard convention $(A) = 2^{-19}x$, $(C) = 2^{-19}y$, so the operation $c(A) \xrightarrow{x} B$ gives $(A,B)' = 2^{-38}xy$ on this convention. Hence xy is given by (A,B)' provided we suppose that the binary point is 38 places to the right of the standard position, which puts it between the p_2 and p_1 positions of register B; and one right shift of the digit pattern in A,B will bring the binary point to the right of the bottom position p_1, as assumed in the representation of x and y.

If xy is positive, the assumption $xy < 2^{19}$ secures that the whole product appears in B (the part in A being entirely zero), so the problem is solved by

```
0      c(A) —x→ B
1      r(B) —→ A   :   (A)' = xyp1.
```

If xy is negative the top part of the machine product is a string of 1's, and on the assumption $|xy| < 2^{19}$ the whole of A is filled with 1's; the significant part of the product is all in B, and the right shift of this with the sign digit $\overline{1}$ added on the left is what is required. In both cases therefore the answer is s(A) + r(B).
Since command 1 above erases the sign digit of (A), the solution

```
0    c(A) ──x B
1    r(B) ── C
2    s(A) ──± C        (C)' = xyp₁
```

$c' = xyp_{20}$ [illegible] and $|xy| < 1$

2.3. Formation of constants.

Constants are often required as data for a calculation or for making alterations in the sequence register. The constants p_{20}, p_{11}, p_1, which on the standard convention represent -1, 2^{-9}, 2^{-19} respectively are directly available from sources 31, 24, 25. More generally, any constant which, as a digit-train, has a symbol of the form (x,y,0,0) may be read out from the interpreter-source (K), no. 26. Thus the command 5,17 (K) $\longrightarrow$ A or 5,17 (K) $\xrightarrow{\pm}$ A will substitute or add the pulse-train (5,17,0,0) into register A.

A constant whose lower half is not zero cannot be directly generated in this way, but it can be formed with the help of the H-register. Let X be any cell containing the word (a,b,c,d). Then

(X) $\longrightarrow H_u$ gives in H the word (a,b,a,b)
(X) $\longrightarrow H_1$ '' '' '' (c,d,c,d)

and for (H) = (x,y,x,y) the command

$(H_u) \longrightarrow$ Y gives in Y the word (x,y,0,0)
$(H_1) \longrightarrow$ Y '' '' '' (0,0,x,y)

By means of the H-register we can therefore perform shifts of half word-length.

<u>Ex.9. Place the pulse-train (0,0,0,10) in C.</u>

```
0    0,10(K) ── Hu
1       (H1) ── C
```

<u>Ex.10. Place the number $(\frac{1}{2} + 2^{-14})p_{20}$ in cell 5,17</u>

0	8,0(K) $\longrightarrow$ A	: $\frac{1}{2}p_{20}$ = (8,0,0,0)
1	1,0(K) $\longrightarrow H_u$	: $2^{-4}p_{20}$ = (1,0,0,0)
2	$(H_1) \xrightarrow{+}$ A	: adds (0,0,1,0) to A
3	(A) $\longrightarrow$ 5,17 M	

Another way of having constants available is to store them in memory-cells at the time when the programme is entered in the machine, and for a four-term constant this is more economical than to generate it as in Ex.10.

<u>Ex.11. Evaluate $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$, where the numbers xp_{20}, a_0p_{20}, a_1p_{20}, a_2p_{20}, a_3p_{20} are stored in cells 95,96,97,98,99.</u> The polynomial is evaluated in the form $((a_3x + a_2)x + a_1)x + a_0$.

0	(3,2 M) $\longrightarrow$ A	$(A)' = a_2$
1	(2,31 M) $\longrightarrow$ C	$(C)' = x$
2	(3,3 M) $\xrightarrow{x}$ B	$(A)' = a_2 + a_3x$

```
5        c(A) ──x── B
6   (3,0 M) ──+── A        (A)' = f(x)
7        p_20 ──── T
```

We have here omitted round-offs, and have added command 7 to stop the machine. In practice if f(x) were the desired end point the programme would proceed to send (A)' to the punch or teleprinter.

The procedure of this example is used for finding circular and exponential functions from suitable polynomial approximations.

2.4. Operations on the sequence register.

The c(S) destination is used for the conditional skipping of an order, mainly on the criterion of the sign digit of some number. The S and +S destinations are used for an unconditional break in the sequential selection of commands.

Ex. 12. Given $(A) = xp_{20}$, get $(A)' = |x|\, p_{20}$.

$$\begin{array}{lll} 1,0 & s(A) \xrightarrow{c} S & \\ 1,1 & c(A) \xrightarrow{-} A & \\ 1,2 & c(A) \xrightarrow{-} A & : \quad (A)' = |x| p_{20} \end{array}$$

At the start of the operation the sequence register S is supposed to contain the number 1,0 (in p_{11} units) of the first command (i.e. the cell number where the command is stored). At stage (iii) of the first computer-cycle (S) becomes 1,1, and at stage (iv) the first command is obeyed. This command converts the sign digit of (A) into the same digit in the p_{11} position and adds this to (S); hence if $x < 0$ (S) becomes 1,2 and if $x \geq 0$ (S) remains at 1,1. Accordingly the next command to be obeyed is drawn from cell 1,1 or cell 1,2 according as $x \geq 0$ or $x < 0$, and in the former case the command 1,2 will be subsequently taken. The effect is that (A) has its sign changed twice if $x \geq 0$ but once only if $x < 0$, so in both cases we finish with $(A)' = |x|$ - except when $x = -1$, and then we finish with $(A)' = -1$, on the standard convention(1).

Ex.13. Calculation of $\sin \pi x$, $\cos \pi x$ for $-1 \leq x < 1$

The argument is taken in the form πx so that the range of x that can be represented on the standard convention corresponds to the period 2π of the functions. The calculation is made from the formula

$$\frac{\sin \pi x}{4x} = 0.7853974 - 1.291842x^2 + 0.635901x^4 - 0.139599x^6$$

which gives $\sin \pi x$ with an error not exceeding 10^{-6} for $|x| \leq \frac{1}{2}$. For x outside this range we use $\sin \pi x = \sin \pi(1 - x) = \sin \pi(-1-x)$. Note (1) how the multiplier 1.291842 is handled, (2) that overcarries at commands 0,3 are irrelevant since to change x by 2 is without effect on $\sin \pi x$, $\cos \pi x$. It is supposed that $(A) = x$, at the start.

The significance of commands 21,22 will be shown in Chapter 3.

(1) The machine subtraction of the digit train (16,0,0,0) from (0,0,0,0) gives the digit train (16,0,0,0) - the same as though the

0	$8,0 \xrightarrow{-} A$	$(A)' = x - \frac{1}{2} \pmod 2$
1	$s(A) \xrightarrow{c} S$	
2	$c(A) \xrightarrow{-} A$	$(A)' = \frac{1}{2} - x \pmod 2$ if x is not in $(-\frac{1}{2},\frac{1}{2})$
3	$8,0 \xrightarrow{+} A$	$(A)' = \left.\begin{matrix} 1-x & (\text{if } \frac{1}{2} \leq x < 1, \\ x & (\text{if } -\frac{1}{2} < x < \frac{1}{2}, \\ -1-x & (\text{if } -1 \leq x \leq -\frac{1}{2}) \end{matrix}\right\} = u$
4	$(A) \longrightarrow D_0$	$(D_0)' = u$
5	$(A) \longrightarrow C$	
6	$c(A) \xrightarrow{x} B$	
7	$(R) \xrightarrow{+} A$	
8	$c(A) \longrightarrow C$	$(C)' = u^2$
9	$(0,26\ M) \xrightarrow{x} B$	
10	$(0,25\ M) \xrightarrow{+} A$	
11	$c(A) \xrightarrow{x} B$	
12	$(0,24\ M) \xrightarrow{+} A$	
13	$c(A) \xrightarrow{x} B$	
14	$(R) \xrightarrow{+} A$	
15	$(0,23\ M) \xrightarrow{+} A$	
16	$(C) \xrightarrow{-} A$	$(A)' = \frac{\sin \pi u}{4u}$
17	$(D_0) \longrightarrow C$	
18	$c(A) \xrightarrow{x} B$	
19	$16,2 \longrightarrow L_2$	
20	$(R) \xrightarrow{+} A$	$(A)' = \sin \pi x$
21	$p_{11} \xrightarrow{+} D_{15}$	
22	$(D_{15}) \longrightarrow S$	
23	< 12,18,3,30 >	0.785 3974
24	< 27,10,18,15 >	-0.2918420
25	< 10,5,18,19 >	0.635901
26	< 29,24,16,26 >	-0.139599

It is easy to see that register capacity will not be exceeded up to command 16, and thereafter the only chance of trouble is for $|x|$ near $\frac{1}{2}$, when $|\sin \pi x|$ is near 1. The approximation has been carefully chosen so that here it is slightly in defect; it gives in fact $\sin \frac{1}{2}\pi = 1 - 2^{-18}$.

Suppose now that initially $(A) = x$ and we require $\cos \pi x$. Since $\cos \pi x = \sin \pi(x + \frac{1}{2})$, a programme with $8,0 \xrightarrow{+} A$ preceding command 0 would give finally $(A)' = \cos \pi x$. Since this initial command would cancel command 0, we shall obtain $\cos \pi x$ if we enter the programme at command 1 with $(A) = x$.

<u>Ex.14.</u> <u>Get $(A)' = (A) + 9(B)$, assuming that the addition does not overcarry.</u>

This could be done by nine consecutive $(B) \xrightarrow{+} A$ commands, but a more powerful solution is

1,0	$0,8(K) \longrightarrow C$	: set count
1,1	$(B) \xrightarrow{+} A$	
1,2	$0,1(K) \xrightarrow{-} C$	count
1,3	$s(C) \xrightarrow{c} S$	

$A \rightarrow C$
$B \rightarrow A$
$2A \rightarrow +A$ | $A' = 3B$
$2A \rightarrow +A$ | $A' = 9B$

The commands 1,1 - 1,4 form a cycle on loop, for when 1,4 is obeyed it puts the number 1,1 into the sequence register, so that the following command will be drawn from cell 1,1 and then will follow 1,2 and 1,3. A new repetition of the loop will take place whenever 1,4 is obeyed; and 1,3 secures that it will be obeyed so long as the number (C) remains ≥ 0. Command 1,2, however, diminishes (C) by 0,1 for every traverse of the loop, so ultimately (C) becomes negative; and on the first occasion when this is so 1,4 is skipped and we emerge from the loop to the stop command 1,5.

By command 1,0 we place initially in C the number that will secure that the loop is traversed 9 times: to check this in detail:

after 1st (B) $\xrightarrow{+}$ A, (C) becomes 0,7 and 7 + 1 = 8
" 2nd " " " 0,6 and 6 + 2 = 8
" 9th " " " 0,-1^{x} and 9 + (-1) = 8.

The number in C effectively counts the number of times the loop has been traversed, and by a proper 'setting of the count' (command 1,0) we secure the desired number of repetitions.

An alternative solution, using a 'count up' instead of a 'count down' and showing other variants, is

$$\begin{array}{rll}
 & 1,0 & 31,23\ (K) \longrightarrow C \\
\hline
1,5 \longrightarrow & 1 & (B) \xrightarrow{+} A \\
 & 2 & p_{11} \xrightarrow{+} C \\
 & 3 & s(C) \xrightarrow{c} S \\
\ulcorner & 4 & p_{11} \xrightarrow{+} S \\
 & 5 & 31,27\ (K) \xrightarrow{+} S \\
\hline
\llcorner\!\!\longrightarrow & 6 & \text{next command}
\end{array}$$

Command 1,0 would have the same effect as 0,9(K) $\xrightarrow{-}$ C applied when C is initially clear; but as we cannot assume C to be initially clear we use a substitution command having the desired effect. At 1,2 we have used the p_{11} source (no.24) instead of the K source for the constant (0,1,0,0) which is p_{11}. Regarding 1,4 and 1,5 it is to be remembered that when either of these commands is obeyed (at stage (iv) of the computer-cycle) the automatic addition of 0,1 to S has just taken place. Thus 1,4 will cause 1,6 to be taken as the next command, while 1,5 will give (S)' = 1,6 + 31,27, equivalent to (S)' = 1,1 on machine addition; the command has the effect of subtracting 0,5 from (S), but cannot be programmed 0,5 $\longrightarrow$ S because subtraction from (S) is not a machine operation.

The reading of a programme is facilitated by inserting lines to mark off the loops which the programme contains and by indicating points where entry is not always from the preceding command; two methods of doing this are illustrated above.

The following part-programme, or a variant of it, is frequently required. In addition to devices that have already been exemplified it uses the teleprinter destination O_t (no.2).

Ex.15. Register A contains, on the standard convention, a number x that is not -1. Express it in decimal notation and print it.

The procedure is first to find and print the sign of x; and then to find $|x|$ and print successively its decimal digits. If $|x| = 10^{-1}a_1 + 10^{-2}a_2 + \dots$ where $0 \leq a_1, a_2, \dots \leq 9$ we have $10|x| = a_1 + 10^{-1}a_2 + \dots$, and here a_1 is the integral part which has to be isolated. To get a machine product that is within register capacity we multiply $|x|p_{20}$ by $10p_1$, whose machine-product is $2^{-19} \times 10|x|p_{20}$; and this will represent $10|x|$ if we take the binary point 19 places to the right of the standard position, i.e. between the p_1 position of register A and the p_{20} position of B. Hence the integral part of the product will occur in A, in p_1 units, and the fractional part in B with the point one place to the left of standard position.

The programme should now be intelligible; we give it for a 6-decimal print, but since $2^{-19} \approx 2 \times 10^{-6}$ the last decimal will not be fully significant.

	1,0	$0,27(K) \longrightarrow O_t$	: signals 'figure-shift' to teleprinter
	1	$s(A) \xrightarrow{c} S$	
	2	$0,3 \xrightarrow{+} S$	: if $x \geq 0$, skip to 1,6
	3	$c(A) \xrightarrow{-} A$	)
	4	$0,11(K) \longrightarrow O_t$	) if $x < 0$, get $(A)' = \|x\|$, print − , and skip 1,6
	5	$p_{11} \xrightarrow{+} S$	)
	6	$0,10(K) \longrightarrow O_t$	print +
	7	$0,12(K) \longrightarrow O_t$	print dec.point
	8	$0,10(K) \longrightarrow H_u$	
	9	$(H_1) \longrightarrow C$	set multiplier $(C)' = 10p_1$
	10	$5,0(K) \longrightarrow D_0$	set count for 6-decimal print
1,16 →	11	$c(A) \xrightarrow{\times} B$	
	12	$c(A) \longrightarrow O_t$	print integral part of product
	13	$r(B) \longrightarrow A$	set fractional part into A on standard convention, ready for 1,11 on next cycle
	14	$1,0 \xrightarrow{-} D_0$	count
	15	$s(D_0) \xrightarrow{c} S$	test for completion
	16	$31,26(K) \xrightarrow{+} S$	gives return to 1,11
	17	$0,31(K) \longrightarrow O_t$	signals 'space' to teleprinter (to leave a gap before another number is printed).
	18	next command.	

Note: The teleprinter takes its signals from the 'logical sum' or disjunction of the p_{15}-p_{11} and p_5-p_1 positions of the output register O_t. Normally one of these sets of digits is zero and the other then determines entirely the signal that is sent; in 1,0 and 1,4 etc. the operative group is $p_{15} - p_{11}$, while in 1,12 it is $p_5 - p_1$. If $(O_t) = 0$ a signal is sent whereby 'A' (on letter

The following is a loop in which the number of repetitions is not explicitly set in advance, but is implicitly determined by the 'data'.

<u>Ex.16.</u> <u>Registers A,B contain a positive number $\underline{x}$ (the result of a multiplication, say) on the convention that the binary point lies between the p_1 position of A and the p_{20} position of B. Represent this number in the machine in the form $2^n y$ where $0 \leq y < 1$.</u>

The index n will be represented in D_0 with p_1 as unit. To right-shift (A,B) one place is equivalent to dividing x by 2, so in compensation we must increase the index (D_0) by p_1. This right-shift is not a machine operation, and it must be done by stages, one of which is to 'move' the bottom digit of (A) into the top position of another register, which is conveniently taken as D_1 (1). The operation is complete when the integral part in A becomes zero, and this is tested by use of the source $\overline{z}(A)$, no.10, which reads out zero if (A) = 0 and p_1 otherwise. The source $p_1(A)$, no.8, also is used.

D SD	0	$(D_0) \xrightarrow{-} D_0$	clear D_0 to receive the index
1 RB D	1	$r(B) \longrightarrow D_1$	moves fractional part of x into D_1 with point in standard position
ZA CS	2	$\overline{z}(A) \xrightarrow{c} S$	exit test
7 K PS	3	$0,7(K) \xrightarrow{+} S$	
1 PS PD	4	$p_{20} \xrightarrow{+} D_1$	adds $1.p_{20}$ or $0.p_{20}$ to D_1
LA CS	5	$p_1(A) \xrightarrow{c} S$	according as $p_1(A)$ is 1 or 0
1 PS PD	6	$p_{20} \xrightarrow{+} D_1$	
1 SD D	7	$r(D_1) \longrightarrow D_1$	right-shift (A,D_1); after 7,
HA A	8	$\frac{1}{2}(A) \longrightarrow A$	$p_{20}(D_1)$ becomes 0, and on 8 the former bottom digit of (A) is lost
PL PD	9	$p_1 \xrightarrow{+} D_0$	increase index to compensate right-shift
3123 K PS	10	$31,23(K) \xrightarrow{+} S$	return to 2
	$3 \longrightarrow 11$	next command	$(x) = 2^{(D_0)'}(D_1)'$

Note (1) that we must apply the exit test before starting to right-shift, because x may be less than 1 at the start. (2) On each right-shift of D_1 a digit is lost from its bottom end; it is assumed that we are interested only in the 19 most significant digits of x. (3) The effect of 5 is to convert a 1 in the p_1 position into 1 in the p_{11} position before adding it to (S), whereas in previous examples the 'counted' digit has been in the p_{20} position. A command $(X) \xrightarrow{c} S$ where (X) has non-zero digits in both the $p_1 - p_{11}$ and $p_{15} - p_{20}$ groups would add $2p_{11}$ to (S), but occasions for using this facility are very rare.

(1) We could use C instead of D_0 or D_1 ; but in an actual programme

2.5. Stop commands.

Any non-zero pulse-train sent to destination T (no.31) operates a switch which stops the machine. If such a command is stored in cell n the machine in obeying this command will stop; if then re-started (by a manual switch) it will proceed with the commands stored in cells n + 1, n+ 2, ...

The standard stop command is $p_{20} \longrightarrow T$: primarily because its coded form (0,0,31,31) is very easy to recognize on programme tapes.

A running stop is effected by the command 31,31(K) $\xrightarrow{+}$ S; its effect is to subtract 1 from (S) at stage (iv) of each computer cycle, which cancels the 1 added at stage (iii), so this same command is repeated indefinitely. This device has no useful function, but the 'hoot stop'

```
cell m      31,31(K) ——  P
cell m+1    31,30(K) —+— S
```

can be used as a signal of distress (e.g. if the machine is asked to divide by 0)(1) or of triumph (to summon an attendant when a job is finished); the two commands form a loop that is repeated indefinitely. The digit-train 31,31 has been chosen to send to the loudspeaker P (destination 10) because it is the heaviest that is available from source K.

Stops are inserted into programmes at points where it is desired to examine the contents of registers on the monitors of the machine or to read data from the hand-set registers N_1, N_2 (sources 2,3). Successive stops can be distinguished from each other by using commands 0,1(K) —— T, 0,2(K) —— T etc. ; the addresses 0,1 etc. will be visible on the monitor lights after the commands have been obeyed.

2.~~5~~6. Notation.

In writing programmes, a pair of commands such as

(5,17 M) —— A , 5,17(K) —— A

are distinguished from each other by the fact that the former has '5,17' inside the brackets that denote 'content of' while the latter has '5,17' outside them. It is customary therefore to omit the symbols M,K and write simply

(5,17) —— A 5,17 —— A ;

but when it comes to punching such commands on tape the fact that brackets imply source M and absence of brackets imply source K is of course vital. For a transfer to a store-cell we similarly omit the symbol M from written programmes, and write e.g. (A) —— 5,17 in place of (A) —— 5,17 M.

For the four parts of the magnetic disc store the suffices a,b,c,d are appended to the numerical symbols.

---

(1) If (A) is the proposed divisor the division part of the programme could start with $\overline{z}(A) \xrightarrow{c}$ S, m(K) —— S, leading to the hoot stop

CHAPTER 3. Programming Technique

3.1 Variation of commands by the +K procedure.

In Exx.14-16 we have shown programmes containing loops: sets of commands that are repeated a certain number of times. However it is often desired that the repetition should not be exact, but should involve each time slight variations. Suppose we wish to add together the numbers in a large set of cells m, m+1, m+2, ... The compact way to do it would be by means of a loop of which the kernel was the variable command $(m+r) \xrightarrow{+} A$.

Two ways are available for securing, in effect, such a variable command; we say 'in effect' because the actual set of commands used is quite determinate, and the variability occurs only in their net effect.

The first way - and the one that is usually adopted - is to use the destination +K (no.26), and the common form of command is

$$(X) \xrightarrow{+} K \quad .$$

The effect of this command is (i) that the pulse train (x,say) that is in the register or cell X is read out on to the digit trunk and thence into an adding unit attached to the Interpreter register; and (ii) that as the pulse-train representing the following command enters the Interpreter, in the next computer cycle, the pulse-train x is machine-added to it, and it is the resulting pulse-train which is then de-coded and taken as the command(1). For example let x be the pulse-train (5,17,0,0), and let the following command be $(0) \xrightarrow{+} A$, represented by the pulse-train (0,0,0,5). These two pulse-trains when added give (5,17,0,5) which represents the command $(5,17) \xrightarrow{+} A$. Thus the effect of

$$\begin{array}{lll} n & (X) \xrightarrow{+} K & (X) = (5,17,0,0) \\ n+1 & (0) \xrightarrow{+} A & \end{array}$$

is that the contents of cell 5,17 are added into A.

If therefore we have a loop which includes the commands

$$\begin{array}{ll} n & (D_0) \xrightarrow{+} K \\ n+1 & (0) \xrightarrow{+} A \\ n+2 & p_{11} \xrightarrow{+} D_0 \ , \end{array}$$

so that the number in D_0 is increased by p_{11} on each traverse of the loop; then if D_0 contains originally yp_{11} the contents of cells y, y+1, y+2, ... will be added into A on successive traverses.

To complete the construction of the loop we need a return-to-the-start command, and must make provision that the loop be traversed the desired number of times, by including a count and an exit test. And since the number in D_0 increases regularly during the performance of the programme it has the character of a count-number and can be used as such; in doing this we achieve an economy of programming - the use of a register for two or more purposes simultaneously - akin to artistry.

For the execution of these ideas there are a number of variants in detail; the choice of the detailed tactics will be governed by the particular context.

(1) The corresponding facility on the Manchester machine is called

Ex.17. Add the numbers in cells 97,98, ... 132 into register A.

A loop with a count down is one command shorter than a loop with a count up (cf. Ex.14), so in order to use a count-down number in the +K command we take the cells in a decreasing sequence. Since

$$\left.\begin{array}{l} 132 = 4 \times 32 + 4 = (4,4) \\ 97 = 3 \times 32 + 1 = (3,1) \end{array}\right\}, \text{ with difference } (1,3)\text{, a}$$

suitable count-number will start at (1,3) and be diminished by (0,1) after each traverse of the loop, so that it will become negative after (1,4) traverses, which is the correct number. Since 1,3 can be set directly into D_3 we use D_3 for the count-number.

```
              n      $1,3 \longrightarrow D_3$          set count
                     ____________________
n+5 $\longrightarrow$ n+1      $(D_3) \xrightarrow{+} K$
              n+2    $(3,1) \xrightarrow{+} A$   becomes $(4,4) \xrightarrow{+} A$ on 1st traverse
                                                  $(4,3) \longrightarrow A$ on 2nd traverse
                                                  . . . . . . . . . . . . .
              n+3    $p_{11} \xrightarrow{-} D_3$                 $(3,1) \xrightarrow{+} A$ on last traverse
              n+4    $s(D_3) \xrightarrow{c} S$
              n+5    $31,27 \xrightarrow{+} S$
                     ____________________

              n+6    next command
```

It is to be noted that the block of cells n - (n+6) where these commands are stored must not overlap with the block (3,1)-(4,4) containing the numerical operands.

Ex.18. Elimination of a variable between a pair of linear equations.

The solution of linear equations by elementary algebraic processes is based upon the repeated elimination of unknowns. The full programming of the solution is quite complicated, and we shall consider here only an operation which represents, in a somewhat simplified form, the typical elimination step.

Let the equations be

$$a_{r1}x_1 + a_{r2}x_2 + \dots + a_{rn}x_n + a_{r,n+1} = 0 \quad (r = 1,2, \dots n).$$

Suppose the first equation to be retained and used to eliminate x_1 from the rth equation $(r > 1)$. The first equation has to be multiplied by $-a_{r1}/a_{11}$ and added to the rth, so that the rth equation is replaced by a new one in which the coefficient of x_s is $a'_{rs} = a_{rs} + a_{1s}(-a_{r1}/a_{11})$. We shall assume that the multiplier $(-a_{r1}/a_{11})$ has been found and placed in register C, and shall give a programme for extracting the element a_{rs} from store, forming a'_{rs}, and storing it in the cell formerly occupied by a_{rs} (which is of no further interest).

We suppose that initially the coefficients (represented on the standard convention) are stored sequentially according to the scheme

cell	m	m+1	..	m+n	m+n+1	..	m+2n+1	m+2n+2	...
content	a_{11}	a_{12}	..	$a_{1,n+1}$	a_{21}	..	$a_{2,n+1}$	a_{31}	...

A generally useful programme must be valid for any value of n so none of its orders must contain n explicitly; the value of n for any particular set of equations will be stored at the same time as the values of the coefficients are stored, in D_5 say.

For the same reason the programme must not refer explicitly to r. It is plain that the useful datum involving r is the number $m + (r-1)(n+1)$ of the cell where a_{r1} is stored, and we shall suppose that this number is stored in D_6; it would have been arrived at by successive additions of $(n+1)$ to m. The other key cell number is m, which we suppose to be stored in D_7. Finally (D_8) will indicate the value of the second suffix s; we conveniently take (D_8) to be $s - 1$ rather than s. Since these four numbers are either cell numbers or are closely related thereto they will be represented in p_{11} units: initially

$$(C) = (-\frac{a_{r1}}{a_{11}})p_{20}, \quad (D_5) = np_{11}, \quad (D_6) = (m+(r-1)(n+1))p_{11},$$

$$(D_7) = mp_{11}.$$

	t	$(D_8) \overset{-}{\text{——}} D_8$	
t+19 ⟶	t+1	$(D_7) \text{ —— } A$	
	t+2	$(D_8) \overset{+}{\text{——}} A$	entered generally with $(D_8)=(s-1)p_{11}$;
	t+3	$(A) \text{ —— } B$	gives $(B)' = (m+s-1)p_{11}$ = cell number for a_{1s} = u, say
	t+4	$(D_6) \text{ —— } A$	
	t+5	$(D_8) \overset{+}{\text{——}} A$	$(A)' =$
	t+6	$(A) \text{ —— } D_o$	$(D_o)' = (m+s-1+(r-1)(n+1)p_{11}$ = cell number for a_{rs} = v, say
	t+7	$(A) \overset{+}{\text{——}} K$	
	t+8	$(O) \text{ —— } A$	becomes $(v) \text{ —— } A$ and gives $(A)'=a_{rs}$
	t+9	$(B) \overset{+}{\text{——}} K$	
	t+10	$(O) \overset{x}{\text{——}} B$	becomes $(u) \overset{x}{\text{——}} B$ and adds $a_{1s}(-\frac{a_{r1}}{a_{11}})$ to (A)
	t+11	$(R) \overset{+}{\text{——}} A$	
	t+12	$(D_o) \overset{+}{\text{——}} K$	
	t+13	$c(A) \text{ —— } O$	becomes $c(A) \text{ —— } v$ and overwrites a_{rs} by a'_{rs}
	t+14	$(D_8) \text{ —— } A$	completion test.
	t+15	$(D_5) \overset{-}{\text{——}} A$	$(A)'$ is negative for $(D_8)=0,1,..n-1$,
	t+16	$s(A) \overset{c}{\text{——}} S$	but zero for $(D_8)=n$, giving exit from loop
┌	t+17	$0,2 \overset{+}{\text{——}} S$	
	t+18	$p_{11} \overset{+}{\text{——}} D_8$	adjust for dealing with next pair of elements
	t+19	$t+1 \text{ —— } S$	
↳	t+20	next command	

Note (1) at command t+3 we have used B for temporary storage, (2) at t+6 we have stored in D_0 a cell number that is later required; (3) instead of counting towards 0 we have counted away from 0, with a suitable completion test for repetition of the loop; (4) a definite value of t would be chosen so that the stored programme did not overlap with the stored coefficients.

In Exx.17,18 the +K facility has been used to modify the address-part of commands. It can also be used to modify the source and destination parts, but the occasions for doing this are infrequent. Here is one:

<u>Ex.19</u>. <u>Add the modulus of (B) into A.</u>

t	$(R) \xrightarrow{+} K$
t+1	$(B) \xrightarrow{+} A$.

The R source reads out a string of zeros if the p_{20} digit of B is 0, or the string 0 .... 01 if $p_{20}(B) = 1$. The command $(B) \xrightarrow{+} A$ is coded (0,0,11,5); and when this enters the interpreter it finds there and is added to, in consequence of the command $(R) \xrightarrow{+} K$, the digit-train

$$(0\ 0\ 0\ 0) \text{ if } (B) \geq 0, \qquad (0\ 0\ 0\ 1) \text{ if } (B) < 0.$$

Hence if $(B) \geq 0$ command t+1 is executed unchanged, but if $(B) < 0$ it is changed before execution to (0,0,11,6) which is $(B) \xrightarrow{-} A$; so in both cases $|(B)|$ is added to A, unless of course $(B) = -1$. The success of the manoeuvre depends on the fact that the code number for subtraction (6) is next to that for addition (5).

3.2 <u>Variation of commands by explicit operation on them.</u>

Any command is represented in the machine by a pulse-train, and we can subject <u>this</u> pulse-train to 'arithmetical' operations such as are usually applied only to pulse-trains that represent numbers. For example if to the pulse train (3,1,0,5) representing the command $(3,1) \xrightarrow{+} A$ we add the pulse train (0,1,0,0) it becomes (3,2,0,5) and now represents the command $(3,2) \xrightarrow{+} A$. An alternative solution of the problem of Ex. 17 is accordingly

<u>Ex.20</u>.	n	$1,3 \longrightarrow D_3$
n+7 →	n+1	$(n+9) \longrightarrow C$
	n+2	$(D_3) \xrightarrow{+} C$
	n+3	$(C) \longrightarrow n+4$
	n+4	$[p_{20} \longrightarrow T]$
	n+5	$p_{11} \xrightarrow{-} D_3$
	n+6	$s(D_3) \xrightarrow{c} S$
	n+7	$31,25 \xrightarrow{+} S$
	n+8	$p_{11} \xrightarrow{+} S$
	n+9	$< (3,1) \xrightarrow{+} A >$

At command n+1 we put the contents of cell n+9, viz (3,1,0,5) into C, at n+2 we add the current contents of D_3 to this - initially (1,3,0,0), at n+3 we plant the resulting digit-train into cell n+4, and the command represented by this digit train (which in general is $(3,1 + (D_3)) \xrightarrow{+} A)$ is there obeyed, since it is in the cell n+4 traverse of the loop, and it does not matter what it was originally. The original content is usually taken to be p_{20} —— T, a stop signal; if the programme goes as is intended this will be altered before it is obeyed, and the choice of a stop signal will lead to an immediate indication if some error in the execution of the programme should occur. The square bracket notation indicates that it is intended to be overwritten.

The digit-train in cell n+9 serves only as a component in the formation of the variable command n+4, and is not itself ever to be obeyed as a command. It is called a pseudo-command, and in writing the programme this nature is indicated by the angle brackets. Command n+8 is inserted so that the pseudo-command n+9 will be skipped on emergence from the loop.

The procedure of Ex.20 is longer than that of Ex.19, and it is naturally used only when it is unavoidable. An important use of this 'making the machine construct its own commands' occurs in the Control Routine of Chapter 5.

3.3 Routines.

There are many standard operations which are often required but which require a sequence of machine operations for their execution; for example division, square root, calculation of circular and other transcendental functions, printing a number in decimal notation. Many such operations have been programmed once and for all(1) and the sub-programmes are available in the CSIRAC library, whence they can be copied into any programme where they are required. They go under the general name routines(2). It is convenient to leave the usage of this term somewhat vague because there is no sharp dividing line between 'standard operations' and 'programme devices' such as those of Exx. 12,19; but on the whole our usage will be fairly clear-cut.

It often happens that a standard operation occurs once only in a programme - though if it is in a loop it will be performed more than once when the programme is run. In this case an appropriate command-sequence for the operation will naturally be copied at the proper place into the programme. But if the operation occurs more than once we shall save store-space if we have a device whereby we need write only one set of commands for the operation; since the operation is required at two or more different places in the main programme, the device must be such that return to different points in the main programme can be made after the several executions of the operation. This can be achieved by incorporating a certain device in the sub-programme for the operation, and it is sub-programmes containing this device which we call par excellence routines(3).

(1) This is an over-statement. For almost any standard operation there are a number of different but reasonable programmes, the choice between which may be guided by the context of the moment. Experience shows, moreover, that one must be wary of asserting that such and such a programme is the 'best possible' for its purposes.

(2) Often called 'sub-routines', in antithesis to 'master routine' which is the main programme.

(3) These are often called 'closed sub-routines'; the antithesis

The device is (1) to record in a suitable register, say D_{15}, the cell-number of the command (a sequence-change) whereby the main programme is left and the routine is entered (this is called 'planting the link'), (2) to place at the end of the routine a command $(D_{15}) \longrightarrow S$ which when obeyed will cause re-entry to the main programme at the proper point provided it is preceded by $p_{11} \xrightarrow{\pm} D_{15}$ (this is called the 'link command'). The way this device works will be made clear by an example.

Suppose that we have stored in the machine, in cells 0 - 26, the sine-cosine routine given in Ex.13 (§2.4), and that the main programme requires on two occasions that the sine of an angle be calculated. To suit the routine we must use 2-right-angles as the unit of angle, and arrange that the measure x of the angle (mod 2) be placed in register A. Suppose that the commands which finally secure this on the two occasions are placed in cells 3,5 and 4,29, and that on the first occasion we require actually the square of the sine. The main programme then runs

	3,5	command securing $(A)' = x$	
$3,8 \to H_n$	3,6	$(S) \longrightarrow D_{15}$:	gives $(D_{15})' = 3,7$
$0 \to S$	3,7	$0 \longrightarrow S$:	gives $(A)' = \sin \pi x$ on re-entry at 3,8
Routine finishes with $(H_n) \to S$	3,8	$(A) \longrightarrow C$	
	3,9	$c(A) \xrightarrow{\times} B$	
	3,10	$(R) \xrightarrow{+} A$	$(A)' = \sin^2 \pi x$
	. . . .		
	4,29	command securing $(A)' = y$	
	4,30	$(S) \longrightarrow D_{15}$	
	4,31	$0 \longrightarrow S$	
	5,0	. . . .	proceed with $(A)' = \sin \pi y$

By the time 3,6 is obeyed (S) has been increased to 3,7, so it is this value which is planted in D_{15}. At 3,7 the machine switches to the set of commands 0-22 given in Ex.13, so at 21 (D_{15}) is increased to 3,8 and at 22 this number 3,8 is substituted into the sequence register. Hence the command following 22 is taken from cell 3,8, and thence the main programme is pursued. On the second occasion the happenings are similar; at 4,30 we get $(D_{15})' = 4,31$, and after 22 we return to 5,0.

It will be observed that the person making the main programme does not need to know <u>how</u> the routine works in forming $\sin \pi x$. All he needs to know about this routine is (1) that it will finish with $(A)' = \sin \pi x$ provided it is entered with $(A) = x$, (2) that its first command lies in cell 0, (3) that it has been written with D_{15} as link-register, (4) - for purposes of overall design of store-space - that it occupies cells 0 - 26.

If at 4,29 we had desired to find $\cos \pi y$ we would have replaced 4,31 by $1 \longrightarrow S$.

The internal structure of the routine of Ex. 13 depends, at commands 9,10,12,15, on the fact that the routine is written for storage in cells 0 - 26. When incorporated into a specific programme it will, almost always, need to be put into some different block of cells, so all cell-numbers in it will need to be increased can be made to do this 'bookkeeping'.

Consider now in general terms a routine for calculating sin πx with double precision, i.e. to 38 binary places. All numbers will need two registers for their representation, and both addition and multiplication of two such numbers need to be executed by sequences of operations. The relevant thing for the present purpose is not the detail of these operations, but the fact that the sine routine will have the general structure of Ex.13 and will call several times for double-length addition and multiplication. Hence we shall do best to make routines for these operations, which will be 'subordinate' to the sine-routine in the sense that they will be called in during the operation of the sine-routine. The important point now is that, having used D_{15} for the link between the sine-routine and the main programme, its content must remain undisturbed during the whole operation of the sine routine. Hence the link between this routine and the subordinate routines must be placed elsewhere, in D_{14} say, and the sub-routines must be written with the concluding commands $p_{11} \xrightarrow{+} D_{14}$, $(D_{14}) \longrightarrow S$; and if these sub-routines have to be called directly during the main programme the plant command must be $(S) \longrightarrow D_{14}$. Similar considerations apply when there are three or more encapsuled routines.

It is of course mere convention to use D_{15}, D_{14}, ... as link registers.

A routine is a general-purpose tool, and in a particular context may not have the maximum efficiency.

3.4 <u>Loops</u>.

Nearly all programmes contain loops, which are the means whereby a programme occupying say 100 cells may in its execution involve the obeying of thousands or even millions of individual commands. Apart from stop-loops, every loop must have an exit which is reached after some finite number of repetitions, and a test for attainment of the exit. In Exx. 14,15, the number of repetitions is determined by an explicitly set count; in Exx.16,18 it is implicitly determined by what the content of certain registers may be when the loop is entered. Another example of implicit determination is the following routine for division.

<u>Ex.21. Find x/y, where $(A) = xp_{20}$, $(C) = yp_{20}$ and it is assumed that $-1 \leq x/y < 1$.</u>

If y is positive we change the signs of both x and y. Then, if $y = -(1 - c_0)$ we have

$$\frac{x}{y} = -x(1+c_0+c_0^2+c_0^3+\ldots) = -x(1+c_0)(1+c_0^2)(1+c_0^4)\ldots .$$

If $c_{n+1} = c_n^2$ and $a_{n+1} = a_n(1+c_n)$, with $a_0 = x$ we have then $a_{n+1} \longrightarrow -x/y$ as $n \longrightarrow \infty$ (provided $c_0 < 1$, i.e. $y \neq 0$). Also a_{n+1} approaches its limit with $|a_{n+1}|$ increasing, so register capacity will never be exceeded.(1) When $c_n = 0$ <u>as given to single length in the machine</u> there is no further change in a_n, to single length, and the quotient is 'attained'.

(1) The question whether this conclusion remains valid when rounding errors are taken into consideration is here left aside.

0	$(C) \longrightarrow D_0$	
1	$(D_0) \xrightarrow{+} D_0$	
2	$s(C) \xrightarrow{c} S$	If $y = (C) > 0$ replace $(A),(C)$ by their
3	$c(A) \xrightarrow{-} A$	negatives, equivalent to supposing
4	$s(C) \xrightarrow{c} S$	$y < 0$
5	$(D_0) \xrightarrow{-} C$	
6	$p_{20} \xrightarrow{+} C$	$(C)' = y + 1 = c_0$
7	$(A) \xrightarrow{x} B$	taken in general with $(C) = c_n$, $(A) = a_n$,
8	$(R) \xrightarrow{+} A$	giving $(A)' = a_n + a_n c_n = a_{n+1}$
9	$c(A) \longrightarrow D_0$	$= (D_0)'$
10	$(C) \xrightarrow{x} B$	
11	$(R) \xrightarrow{+} A$	$(A)' = c_n^2 = c_{n+1}$
12	$\bar{z}(A) \xrightarrow{c} S$	
13	$0,3 \xrightarrow{+} S$	exit if $c_{n+1} = 0$, with $(D_0) = a_{n+1} = -x/y$
14	$(A) \longrightarrow C$	$(C)' = c_{n+1}$
15	$(D_0) \longrightarrow A$	$(A)' = a_{n+1}$
16	$7 \longrightarrow S$	
17	$(D_0) \xrightarrow{-} A$	gives $(A)' = + x/y$ because entered
18	$p_{11} \xrightarrow{+} D_{15}$	with $(A) = 0$; we finish also with
19	$(D_{15}) \longrightarrow S$	$(D_0) = - x/y$.

Notice that possible overcarries at commands 1,5,6 are irrelevant; we certainly finish at 6 with $(C)' = c_0 (\mathrm{mod}\ 2)$, and the result must be c_0 because $-1 < c_0 < 1$.

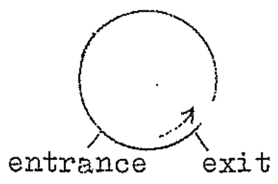
In exx. 14,18 have been shown three methods of counting and testing for completion, and further varieties are possible. For example, suppose that in Ex.18 n had been stored in a cell whose number is congruent to 8(mod 16), say 15,24. Then in place of (t+14) - (t+19) we could proceed

t+14	$(15,24) \xrightarrow{-} D_8$	$(D_8)' \begin{matrix} < 0 & \text{before completion} \\ = 0 & \text{at completion} \end{matrix}$
t+15	$s(D_8) \xrightarrow{c} S$	
t+16	$0,3 \xrightarrow{+} S$	
t+17	$(15,24) \xrightarrow{+} D_8$	
t+18	$p_{11} \xrightarrow{+} D_8$	
t+19	$t+1 \longrightarrow S$	
→t+20	next command	

In certain contexts this procedure would have the advantage of not disturbing the contents of A or C, and of leaving $(D_8)' = 0$ on exit from the loop.

To make a programme containing a loop it is usually best to start by constructing the loop, making sure that it has a valid exit, and then to see what preparatory commands (such as setting of a count) are needed. The allotment of cell numbers to the commands must often be tentative at first.

(e.g. command 16 in Ex.21) would be unnecessary: Essentially a loop has a circular structure, to be cycled the desired number of times, with prescribed entrance and exit points. To adapt this to the linear store of the machine the circle has to be cut at some point and a sequence shift command must be inserted at this point; and the point of section can be chosen at pleasure. It is usually so chosen that either the entrance point is at the top or the exit point is at the bottom; as the preceding examples show, we can not always secure both. On occasion, a judicious alteration in the point of section will save a command; the routine of Ex.21 can in fact be shortened by a cyclic change and slight modification in the loop along with a change of tactics in the preparatory commands; the reader may try his hand at this.



Programmes often contain loops within loops. It is usually best to construct the inside loop first, and thenwork outwards. As an example let us return to the problem of solving linear equations. In Ex.18 we have shown the elimination of x_1 from the r^{th} equation by means of the 1^{st}. This has to be done in succession for $r = 2,3,\ldots n$ and we shall amplify the programme so as to secure this. The loop in cells (t+1) - (t+19) will be the inner loop; for present purposes the relevant facts about it are that it involves r only via the contents of C and D_6. The outer loop must arrange that r starts at 2 and finishes at n, and it must also secure that for each r the correct multiplier $(-a_{r_1}/a_{11})$ is taken. This requires a division for which we shall call in the routine of Ex.21, supposed stored in cells 0-19. The typical adjustment of (D_6) is to increase it by $(n+1)p_{11}$ for a unit increase in r. It will be best to use a new register for the r-count; (D_6) is not convenient for this purpose because it involves r in somewhat complicated fashion. The detail of the r-count can be arranged in a number of ways, of which one (which is at any rate close to the optimum) has been chosen. The cell numbers for the commands are of course assigned at the last stage.

<u>Ex. 18 (cont.)</u>: <u>Preliminary group</u>

```
      t-16      $(D_5) \longrightarrow A$
      t-15      $0,2 \xrightarrow{-} A$
      t-14      $(A) \longrightarrow D_9$        set $(D_9)$ = r-count at n-2
      t-13      $(D_7) \longrightarrow A$
      t-12      $(A) \longrightarrow D_6$        set $(D_6)$ initially at m
```

<u>Outer loop</u>

```
t+21 →t-11      $p_{11} \xrightarrow{-} D_9$       r-count, initially n-3
      t-10      $(D_5) \longrightarrow A$  }
      t-9       $p_{11} \xrightarrow{+} A$     }  sets n+1 into (A)
      t-8       $(A) \xrightarrow{+} D_6$        re-set $(D_6)$ for current value of r:
                                   cell no. for $a_{r1}$
      t-7       $(D_6) \xrightarrow{+} K$                   )
      t-6       $(0) \longrightarrow A$        $(A)' \doteq a_{r1}$  }set factors for division
      t-5       $(D_7) \xrightarrow{+} K$                   )         routine
      t-4       $(0) \longrightarrow C$        $(C)' = a$        )
```

t-3	$(S) \text{——} D_{15}$	calls for division; return to main programme with $(D_o) = - a_{r1}/a_{11}$
t-2	$0 \text{——} S$	
t-1	$(D_o) \text{——} C$	(C)' = multiplier for current value of r.
t	$(D_8) \overset{-}{\text{——}} D_8$	set count for inner loop
(t+1) to (t+19)	inner loop	
t+20	$s(D_9) \overset{c}{\text{——}} S$	
t+21	$t-11 \text{——} S$	

It is most important to check the counting. On the first traverse of the outer loop $(D_9) = n-3$, corresponding to $r = 2$, and for each traverse (D_9) is diminished by 1. Hence $r = n$ corresponds to the traverse with $(D_9) = -1$ and exit from the loop occurs after command t + 20.

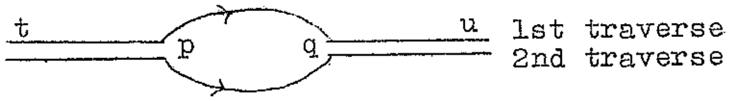
This programme will of course be correct only if, for each r and s,

$$- 1 \leq - \frac{a_{r1}}{a_{11}} < 1 \quad \text{and} \quad -1 \leq a_{rs} + a_{1s}(- a_{r1}/a_{11}) < 1 \;;$$

the manoeuvres that will secure this are not here considered.

3.5 Switches.

It sometimes happens that two sets of commands to be executed are largely identical, but have certain differences. We can then write a programme in which the identical sequences occur once only and in which at the end of such a sequence the machine is appropriately directed by means of a switch: the logical layout may be represented thus:



The switch is constituted by the sign digit of a D-register, say D_2, and a suitable scheme is

t-1	$p_{20} \text{——} D_2$	set switch negative for 1st traverse
t	. . . .	identical group
	. . . .	
p	$s(D_2) \overset{c}{\text{——}} S$	
p+1	$v \text{——} S$	taken on 1st traverse
	. . . .	
v-1	$q \text{——} S$	
→ v	. . . .	taken on 2nd traverse
	. . . .	
→ q	. . . .	identical group
	. . . .	
u	. . . .	
u+1	$p_{20} \overset{+}{\text{——}} D_2$	$(D_2)'$ = 0 (1st time), - 1 (2nd time)
u+2	$s(D_2) \overset{c}{\text{——}} S$	
u+3	$t \text{——} S$	return for 2nd traverse

It will be observed that $(D_2)' = -1$ after the 2nd traverse, so if a similar switching procedure is required later in the programme a command corresponding to t-1 will be unnecessary. The switching procedure involves 7 commands, and near-identical sequences of length less than 7 are therefore best written separately.

Sometimes a sequence is to be traversed twice without any variation. The preceding scheme then has an evident simplification, and the switching procedure is equivalent to a count, for two traverses of a loop, which is <u>self-resetting</u>. This is called a 'binary switch'. Remembering the mod 2 character of machine addition it will be seen that a self-resetting 'ternary switch', for 3 traverses of a loop, can be based on repeated additions or subtractions of the number 2/3; but since the binary representation of 2/3 in the machine involves a rounding error, this switch will break down if the number of traverses of the loop or loops exceeds about 330,000.

3.6 <u>Strobes.</u>

A <u>strobe</u> is a single 1 digit placed in a register and for some purpose moved systematically to the right or to the left. Such a digit can be used for a count of 19 or 20 (by a sign-test or zero test), but nothing is thereby gained over the conventional procedure unless the strobe is serving some other purpose also. In Ex.22 we show such a double use, and in Ex.23 we show a strobe-count.

<u>Ex.22. Given $(A) = x$ with $0 < x < 1$, find $\log_2 x$.</u>

If $x \geq \frac{1}{2}$ the characteristic is $\bar{1}$ and the logarithm can be exhibited in a single register; if $x < \frac{1}{2}$ separate registers are required for the characteristic and mantissa. We first express x in the form $x = 2^{-n}(A)$ where $\frac{1}{2} \leq (A) < 1$ (commands 1-7) and then build up the mantissa digit-by-digit. A left-shifting strobe is used, which becomes finally the characteristic of the logarithm in the case $x \geq \frac{1}{2}$.

	0	$p_1 \longrightarrow D_1$	set strobe
	1	$(D_0) \xrightarrow{-} D_0$	for characteristic
	2	$p_1 \xrightarrow{-} D_0$	normalization loop; exit when $(A)' \geq 1$
	3	$2(A) \longrightarrow A$	
	4	$s(A) \xrightarrow{c} S$	
	5	$31,28 \xrightarrow{+} S$	
	6	$\frac{1}{2}(A) \longrightarrow A$	reverse the overcarry which gave $(A) \geq 1$, giving $(A)' = 2^n x$ where $(D_0)' = -(n+1)p_1$ and $\frac{1}{2} \leq 2^n x < 1$
	7	$p_{20} \xrightarrow{\dotplus} A$	
$17 \longrightarrow$	8	$(A) \longrightarrow C$	Let $(C)=(A)=2^{-1+\frac{c_1}{2}+\frac{c_2}{2^2}+\cdots}$, $(c_1, c_2, \ldots 0 \text{ or } 1)$
	9	$(D_1) \xrightarrow{+} D_1$	Left-shift strobe (and digit train $c_1c_2\ldots$ to its right)
	10	$c(A) \xrightarrow{x} B$	$(A)' = 2^{-2+c_1+\frac{c_2}{2}+\cdots}$
	11	$2(A) \longrightarrow C$	$(C)' = 2^{-1+c_1+\frac{c_2}{2}+\cdots} \begin{cases} \geq 1 \text{ if } c_1 = 1 \\ < 1 \text{ if } c_1 = 0 \end{cases}$
	12	$s(C) \xrightarrow{c} S$	

$-1+\frac{c_2}{2}+\cdots$

15	$(R) \xrightarrow{+} D_1$	adds c_1p_1 to (D_1)
16	$s(D_1) \xrightarrow{c} S$	exit when strobe reaches top of D_1
17	$31,22 \xrightarrow{+} S$	
18	$(D_1) \longrightarrow A$	$(D_1)=(A)'=\log_2(2^n x)=\bar{1}\cdot c_1c_2\cdot\cdot(=\log_2 x$ if $n = 0)$
19	$p_{20} \xrightarrow{+} D_1$	cancel sign digit from D_1, giving
20	$p_{11} \xrightarrow{+} D_{15}$	$\log_2 x$ with characteristic (a
21	$(D_{15}) \longrightarrow S$	negative integer) in D_0 and mantissa (a positive fraction) in D_1.

Ex.23. Solution of an equation by trial process. Let $f(x)$ be any function which increases monotonically from a negative value at $x = 0$ to a positive value at $x = 1$. The single root of $f(x) = 0$ between $x = 0, 1$ will be constructed digit-by-digit, with the help of a right-shifting strobe in D_0. It is supposed that there is a routine, stored in cells t, t+1, ..., written for linking in D_{15}, which when entered with $(A) = x$ yields finally $(A)' = f(x)$. Counting is done by a left-shifting strobe in D_3. The programme is arranged so that nothing significant is left in any registers other than D_0, D_3 , D_4 during the calculation of $f(x)$; in certain cases (e.g. calculation of square and higher roots) more economical arrangements are possible.

0	$8,0 \longrightarrow D_0$	set digit-strobe at $\frac{1}{2}$
1	$(D_4) \xrightarrow{-} D_4$	for current approximation to x
2	$p_1 \longrightarrow D_3$	set count strobe
3	$(D_4) \longrightarrow A$	
4	$(D_0) \xrightarrow{+} A$	add digit to form new trial value of x
5	$(A) \longrightarrow D_4$	$(D_4)' = (A)' =$ trial value for x
6	$(S) \longrightarrow D_{15}$	
7	$t \longrightarrow S$	$(A)' = f(x)$
8	$(D_0) \longrightarrow B$	
9	$s(A) \xrightarrow{c} S$	
10	$(B) \xrightarrow{-} D_4$	if $f(x) \geq 0$, decrease trial x by removing the digit added at 4
11	$r(D_0) \longrightarrow D_0$	right-shift digit-strobe
12	$(D_3) \xrightarrow{+} D_3$	left-shift count-strobe
13	$s(D_3) \xrightarrow{c} S$	
14	$3 \longrightarrow S$	
15	$p_1 \xrightarrow{+} D_4$	round off, to suit case where the root is exact
16	$(D_4) \longrightarrow A$ (1)	$(A)' =$ required root

(1) This command has no logical force, and is inserted merely in accordance with a general convention that results are exhibited

CHAPTER 4. Input and Output.

4.1 The programme tape and punch.

Input of programmes for CSIRAC is from punched paper tape of capacity 12 holes per row(1). The tape is read by a photo-electric reader, whence a pulse-pattern is set into the input register corresponding to the pattern of the tape-row: to a tape-hole corresponds a 1 (pulse) and to a tape-blank corresponds a 0 (gap). To the 12 tape-positions correspond the positions p_1 to p_{10} and p_{19}, p_{20} in the registers of the machine, so we can designate the tape-positions by these symbols. But in normal practice the programme-data are confined to the p_1 to p_{10} positions, and holes (or their absence) in the p_{19} and p_{20} positions are used as 'cues' or 'tags' or 'control designations' regarding the handling of this data, in a way that will be shown. Hence for the p_{19} and p_{20} positions the distinctive symbols X and Y, respectively, are used, as with Hollerith-type punched cards.

The keyboard from which the tape-punch is actuated has 32 keys, each of which carries one of the numbers 0,1,...31 and also the symbols for the source and destination that are coded by this number. There are also X and Y keys, an erase key and a 'punch' key. Depression of the key numbered n causes the binary symbol for n (as a hole-blank pattern) to be set up; on the first such depression the symbol is set up in the p_{10}-p_6 positions, as for calling a source, and on the second depression in the p_5-p_1 positions, as for calling a destination. An X or Y key (or both) causes similarly a setting for the X or Y position (or both). When the set-up is complete, depression of the 'punch' key actuates the punch; and an error in setting detected before this key is depressed can be corrected by use of the erase-key.

A 20-digit command-word has to be assembled, by a process that will be shown, from two 10-hole tape rows. In normal practice the top or address-half of the word is punched first, and the bottom half (indicating the source and destination) is punched on the next tape-row; the latter is accompanied by an X-punch but the former is not. For command words having the top half (0,0) - (address zero) - the bottom half only, with the X designation, is punched. Since the symbols in which programmes are written are shown on the keyboard, punching can proceed without an intermediary translation from programme symbolism to numerical code.

Since a Y-punched hold is read as a 1 digit in the p_{20} position it has the potentialities of a 'switch' in the sense of §3.5.

4.2 Assembling of words by the Primary Routine.

Suppose that a punched tape is in the tape reader with a word y under the reading head. When a command which calls the input-register I as a source is obeyed, the word x currently in I (which is shown on monitor lights) is read out to whatever destination may be called, the word y under the reading head goes into I and overwrites x, and the tape is stepped forward so that the word z following y on it comes under the reading head. The tape reader is thus a feeder for the input register. Any programme containing commands which call on I must thus be designed in relation to what is on the tape, or rather the two must be designed together.

Suppose now that a tape carrying, in punched form, the programme for some calculation is positioned in the reader with the first row of holes under the reading head. The words constituting the programme have to be stored sequentially in the proper cells, and many of these words will have to be assembled from two half-words punched on a pair of tape-rows as stated in §4.1. This assembling and storing is done by the obeying (by the machine) of a set of commands called the Primary Routine or primary. But before this is possible the primary must of course be stored in the machine, and this is done by a special process analogous to forced feeding. The full scheme is

(i) storing the primary, by forced feeding,
(ii) storing the programme for a calculation, by obeying the commands of the primary,
(iii) performing the calculation, by obeying the commands of the programme,
(iv) outputting the results of the calculation, usually by obeying commands of the programme.

In previous chapters we have considered (iii) and Ex. 15 has related to (iv). We shall consider (ii) now, and (i) at the end of this section.

Ex. 24. The primary routine (first version). This is stored always in cells 0-17, and is

	0	$(D_0) \longrightarrow H_1$	isolates the half-word (a,b)
	1	$(D_0) \xrightarrow{+} D_0$	gives $s(D_0) = 1$ if there was an X-punch
	2	$s(D_0) \xrightarrow{c} S$	
	3	$(S) \xrightarrow{+} S$	gives $(S)' = 8$ if there was no X-punch
	4	$(H_1) \xrightarrow{+} A$	if there was an X-punch adds (0,0,a,b) to (A)
	5	$(C) \xrightarrow{+} K$ }	and stores (A) in cell whose number is currently in C and increases storage-cell number
	6	$c(A) \longrightarrow 0$ }	
	7	$p_{11} \xrightarrow{+} C$	
	8	$s(D_0) \xrightarrow{c} S$	
	9	$(H_u) \xrightarrow{+} A$	adds (a,b,0,0) to (A) if there was no X-punch
logical start of loop	10	$(I) \longrightarrow D_0$	reads into D_0 a half-word (a,b) from tape-positions p_1-p_{10} along with possible $X(p_{19})$ and $Y(p_{20})$ punches
	11	$s(D_0) \xrightarrow{c} S$	leads to 13 if there was a Y punch; otherwise to 12 and 0
	12	$0 \longrightarrow S$	
	13	$p_{20} \longrightarrow T$	
later over-written	14	$(D_0) \longrightarrow H_1$ }	stores (0,14,0,0) in C
	15	$(H_u) \longrightarrow C$ }	
	16	$(D_0) \xrightarrow{-} D_0$	clears D_0 so that the obeying of 0,1,2,3, 8,9 on the first traverse shall be vacuous
	17	$0 \longrightarrow S$	

The tape which is to be fed through the reader under the

row 1	0,14 Y
row 2	upper half of 1st word to be stored
row 3	lower '' '' '' '' '' '' '' , with X punch
row 4	upper '' '' 2nd '' '' '' ''
row 5	lower '' '' '' '' '' '' '' , with X punch
. . . .	
row n	lower half of last word to be stored
row (n+1)	Y

Suppose that row 1 is in position under the reading head at a time when the registers I,D_0,H,A,C,S are all clear, with the machine switched on. Since (S) = 0 the first command obeyed is 0, and then follow 1,2,3,8,9, which are all vacuous, i.e. they alter nothing except (S). At 10 the zero content of I is read into D_0, row 1 of the tape is read into I and row 2 comes under the reading head; and then follow 11,12,0,... back to 10, which now puts row 2 into I and gives $(D_0)' = (16,0,0,14)$, where the '16' stands for the 1 in the top position originating from the Y of row 1. Since now $s(D_0) = 1$ we skip from 11 to 13; and on re-starting the machine after the stop we come to 17 and thence 0 with (C)' = (0,14,0,0) and D_0 clear.

On returning again to 10, row 2 goes into D_0 and row 3 into I. Since row 2 has neither X nor Y punch, the sequence goes 11,12,0,1,2,3,8,9,10, and at 9 row 2 is added into the upper half of A - which is equivalent to substituting it into A since A started clear.

On the next cycle row 3 goes into D_0, and the X-punch of this leads to the sequence 11,12,0,1,2,4,5,6,7,8,10. At 4 the p_1-p_{10} part of the row is added into the lower half of A so that the desired word is assembled, at 6 this word is stored in cell no.14, at 7 (C) is increased to (0,15,0,0), and at 10 we start a new cycle with A clear.

Similar cycles are now repeated, with the X-punch always as a storage cue and with successive assembled words stored sequentially. The first four storages overwrite what was originally in cells 14-17.

Finally row n+1 with its Y punch is read via I into D_0; the Y punch forces an exit from the cycle to command 13 and the machine stops, with the programme for the desired calculation ready for operation. And when the machine is re-started the programme will run, starting at command 14.

It follows that a programme which is to be stored under the control of this primary routine must be written with its first command in cell 14.

If the tape contains a number of consecutive rows without X punches they will be added cumulatively into the upper half of A. Hence blank tape-rows are vacuous. In particular, the primary will operate correctly if, at the start when I,D_0,H,A,C,S are all clear, there are a number of blank rows ahead of row 1, one of which is under the reading head.

The stop command 13 plays no logical part but is often a convenience(1). If it is omitted the first tape row must be 0,13 Y.

The forced feeding of the primary is done by switching the machine to an abnormal mode of operation in which, at stage (ii) of each computer-cycle, the logical sum (disjunction) of the contents of the hand-set register N_1 and the sequence register is transmitted to the Interpreter, but the other stages take place normally, so that in particular (S) increases by 1 in each cycle. The word (I) —— M (coded 0,0,1,0) is set on N_1. It is supposed that the primary has previously been punched at the head of the programme tape, with a number of blank rows between its last word no.17 and the programme-rows starting with '0,14 Y'. The tape is stepped by single-shot through the reader until the first word '(D_0) —— H_1' appears on the monitor lights of the input register. Then all registers and memory-cells are cleared, including S, and the tape is stepped or run continuously through the reader. Since S increases by 1 at each computer-cycle the commands successively obeyed have the codes (0,0,1,0), (0,1,1,0), (0,2,1,0), .. and in ordinary notation are (I) —— 0, (I) —— 1, (I) —— 2, etc.; so the successive words of the primary are stored in cells 0,1,2,...17. When they are all in, at any stage before the row '0,14 Y' reaches the reading head, the machine is switched to normal operation and S is cleared to 0. The machine thus starts obeying the commands of the primary, and the first non-vacuous one is 0,14 Y, whose effect we have already considered.

It will be noticed (i) that the primary consists entirely of lower-half words, such as can be punched directly onto tape, and (ii) that these words are all in themselves complete commands. Because of (ii) it is a programme, and because of (i) the forced feeding of it as just described is possible. The commands of the primary are punched without X-tags.

4.3 Input of numerical data.

If the numerical data for a calculation are to be punched on the programme tape they must be punched in binary representation as a pair of half-words. Normally this is done only for 'absolute constants', such as the coefficients in the polynomial of Ex.13. This is because programmes are normally designed to handle any calculation of some specific type e.g. solution of linear equations, and the programme tape will contain references to the numerical data but not the specific data of a particular case. If the specific data are few in number they may be converted by hand to binary representation and input by hand-setting on N_1, N_2 and $I^{(1)}$; for this purpose the programme must contain a stop during which the data are hand-set, and this will be followed by commands calling some or all of N_1,N_2 and I as sources.

When the data are numerous it is best to punch them in decimal representation on tape according to some scheme, and to include in the programme a routine for reading the numbers, converting them to binary representation, and appropriately storing them. The storage arrangements will depend on the particular context, but the punching and conversion can be standardized, and we shall show one scheme for this.

Ex.25. Conversion to binary representation of a number x in the range $-1 < x < 1$, for which the first 6 decimal digits of x are punched in successive rows in the $p_4 - p_1$ positions, and the last of these rows contains a Y punch and $\begin{cases}\text{an X punch}\\\text{no X punch}\end{cases}$ if the number is $\begin{cases}\text{negative}\\\text{positive}\end{cases}$.

Let $x = 10^{-1}a_1 + 10^{-2}a_2 + \dots + 10^{-6}a_6$ where $0 \le a_r \le 9$.
Then
$$x = 10^{-6}\left[\,..\,(10(10a_1 + a_2) + a_3) + ..\right] = 10^{-6}N, \text{ say.}$$
The integer N is built up in register A in the representation with p_1 as unit, the multiplication by 10 being performed by using the factor $\frac{10}{16}$ followed by four left shifts. This gives
$(A)' = Np_1 = N \times 2^{-19}p_{20} = y$ say; so $10^{-6}Np_{20}$ – which is what we require – is equal to $10^{-6}2^{19}y$.

Since $0 < N < 10^6$ and $2^{19} = 524288$ we have $0 < y < 2$, and the operations on A will not have caused any overcarry, but the top digit of (A)' may be a 1, representing +1, not −1. To allow for this the final multiplication is done in the form

$$10^{-6}2^{19}y = 10^{-6}2^{19}(y - 1) + 10^{-6}2^{19},$$

where the final answer will be properly represented in the machine since $x < 1$.

The routine converts −0.999999 punched into −1 in the machine (in A after command 17), while 0.999997 becomes $1-2^{-19}$, which is a reasonable approximation to +1.

0	$A \xrightarrow{-} A$	required since 6 initially calls +A
1	$10,0 \longrightarrow C$	$(C)' = \frac{10}{16}\, p_{20}$
2	$c(A) \xrightarrow{x} B$	multiply current contents of A by 10; vacuous on first entry.
3	$16,6 \longrightarrow L_4$	
4	$(I) \longrightarrow D_0$	
5	$(D_0) \longrightarrow H_1$	isolates the $p_{10} - p_1$ half of I
6	$(H_1) \xrightarrow{+} A$	add next decimal digit
7	$s(D_0) \xrightarrow{c} S$	exit when the Y-punch has been read, with $(A)' = yp_{20}$
8	$31,25 \xrightarrow{+} S$	
9	$p_{20} \xrightarrow{+} A$	$(A)' = (y - 1)p_{20}$
10	$(0,20) \longrightarrow C$	
11	$c(A) \xrightarrow{x} B$	
12	$(R) \xrightarrow{+} A$	
13	$c(A) \xrightarrow{+} C$	$(C)' = \lvert x \rvert$
14	$(D_0) \xrightarrow{+} D_0$	gives $s(D_0) = 1$ if there was an X-tag
15	$(C) \xrightarrow{-} A$	gives $(A)' = \begin{pmatrix} \lvert x\rvert \text{ if no X-tag} \\ -\lvert x\rvert \text{ if X-tag} \end{pmatrix} = x$
16	$s(D_0) \xrightarrow{c} S$	
17	$(C) \longrightarrow A$	
18	$p_{11} \xrightarrow{+} D_{15}$	
19	$(D_{15}) \longrightarrow S$	
20	$< 8,12,13,30 > \; = 2^{19}10^{-6}$	

For punching the decimal digits of a number it is more convenient to put two of them (rather than one) on each tape-row: the 'compact punching' scheme. The conversion routine must then provide for dissecting the row, as read, into its two constituents, and is consequently longer. Conversion routines on this plan are

A data-tape must of course be fed through the tape-reader under the control of a programme already stored in the machine, and two problems here arise. (i) The tape must be stepped into the position where the first punched tape-row is under the reading head. This stepping can take place only by the machine obeying a command in which the input register I is called, and we call I by switching to an abnormal mode of operation in which commands are taken from N_1, and setting an appropriate command, calling I, on the N_1-switches. The safest one to choose is (I) —— I (code $0,0,1,1$) or (I) —— Z (code $0,0,1,20$), since both these send the word $(0,0,0,0)$ read from blank tape-rows to 'nowhere'; (I) —— M(code $0,0,1,0$) will do provided the substitution of zero into cell 0 (i.e. the clearing of this cell) does not spoil the later operation of the programme. When the first punched row is in position, switch the machine to normal operation. (ii) We have to secure the proper synchronization of the obeying of commands with the progress of the tape through the reader. The simplest way to do this is to store the reading-and-conversion routine in cells 14,15,... and to punch Y on the tape in the row preceding the first datum row; and to clear (S) to zero on the appearance of this Y on the input-register lights.

What happens now is this: we start with $(S) = 0$, with D_0 containing the Y (i.e. p_{20}) which was read as the last tape-row of the programme tape, previously stored. The machine proceeds to obey the commands of the primary routine, and at 1 the p_{20} is lost by overcarry and (D_0) becomes zero. Hence the sequence follows 2,3,8,10. At 10 the Y which heads the data-tape goes into D_0 and the first data-row goes into I, and thence follow 11 and the stop at 13. On re-starting the machine commands 14,15,... will be taken, and the first of them which calls I will concern the first data-row. Subsequent calls to I will automatically concern the later data-rows, in order.

If for any reason the reading-and-conversion routine is not stored in cells 14, ... cell 14 must contain the appropriate sequence-shift.

4.4 Output to teleprinter.

In Ex. 15 we have given a routine for binary-to-decimal conversion and printing, and all that need be added is a few words about the layout of the printed page. We may achieve whatever may be desired by including in the programme commands to the teleprinter for space, line-feed, carriage-return, figure shift and letter shift according to the code shown at the end of Chap.1; but it should be noted that there is no back-space facility. The commands $0,29$ —— O_t, $0,30$ —— O_t for line-feed and carriage-return (which if given in this order cancel the 'space' accompanying the line-feed) must be given when or before the line being printed is full. This is done by using a register to count the number of prints per line - usually via the complete numbers printed rather than the number of separate print commands - and forming a loop in the programme the exit from which is based on this count.

4.5 Output to punch.

The punch can be disconnected from the keyboard used in punching programme and data tapes, and connected to the machine. A command of the form (X) —— O_p will then cause the p_1-p_{10} and p_{19},p_{20} digits of the word in X to be punched in their standard positions on one row, and the tape will be moved forward ready for the next punching. The routine for punching a complete word involves its dissection into upper and lower halves, and X or Y

Experience hasshown that output to punch is less subject to machine errors than is output to teleprinter, and there are occasions therefore on which it is best to make the primary output by punching and use a supplementary programme for decimal conversion and printing: punched output will naturally be used when the results are to be data for a later calculation.

Output to both punch and teleprinter can be checked by having the machine form the sum (mod 2) of a set of numbers to be output and then punching or printing this sum, suitably tagged to indicate its significance.

Note. The operation of tape reader, teleprinter and punch is slower than that of the machine in obeying commands, and there is a switch in the machine which automatically inhibits the execution of a command calling any of these mechanical units until the last such commanded operation has been completed: the machine waits until the mechanical unit is ready. The ready-signal interfers with the correct operation of a command to destination +K(no.26), and such a command must never be followed immediately by a command calling I, O_t or O_p.

CHAPTER 5. Controlled input of programmes containing library routines Machine operation.

5.1 Introduction: Control designations.

It is often desired to minimise the time taken to write and punch a programme, which may be done by making the greatest possible use of the CSIRAC library of routines for such standard operations as are required by the programme. The complete programme will comprise these routines, along with a master routine or part-programme which is specific to the problem in hand. The master will call in each routine where it is required, and the command which calls it in has the form n —— S, where n is the number of the cell in which the first command (or occasionally a later one) of the routine is stored.

Now on the one hand most routines contain references, in certain of their commands, to the cells wherein they are to be stored (e.g. command 10 of Ex.25), and on the other hand we cannot hope to reserve a fixed set of cells for the storage of each routine on all occasions when it is required. Hence if we use the input procedure of Chapter 4 we are obliged , for each programme, to re-write the required routines so as to suit the numbers of the cells where they are to be stored. There is however a cunningly devised input procedure whereby this re-writing is avoided but the routines are stored as though they had been re-written. The procedure uses a primary routine slightly modified from Ex.24 together with a control routine.

Suppose for definiteness that for some problem the programme includes the conversion routine of Ex.25, the sine routine of Ex.13, and others, and that we decide to store the full set of words (commands) sequentially, starting with the head word of the conversion routine in cell n_1. Then the last word of this routine will be stored in cell n_1+20, and word 10 must be replaced by $(0,20+n_1)$ —— C. The first word of the sine routine will then be stored in cell n_1+21, and to the cell numbers contained in its words nos. 9,10,12,15 we shall have to add n_1+21. And so on.

The control routine is a set of commands which in their execution by the machine make these adjustments automatically.

For this to be possible it is clearly necessary that those tape-rows which represent words requiring adjustment must be accompanied, on the tape, by some physical indication of what adjustment is required. This physical indication consists of a tape-row (or sometimes two rows) which includes a Y punch. The Y-punch when read, acts as a switch out of the primary into the control routine; and the pattern of holes that accompanies the Y determines the point at which the control routine is entered and thence the effect of the control-commands that are obeyed.

These Y-tagged control-rows are represented on the written programme by appropriate symbols accompanying the words that are to be adjusted; each symbol is a direction to the person punching the tape that he is to punch the corresponding control-pattern with a Y-tag before he punches the word that is to be adjusted.

Opposite the first word (to the left of it in standard practice) of the conversion routine, Ex.25, we write '1 S', this symbol is a mnemonic for 'store the number n_1 of the cell which will hold the first word of routine no.1'. Opposite the first word of the sine routine we write '2 S', standing for store the number n_2 of the cell which will hold the first word of the routine no.2'.

mS consists of two rows:

0, m

0, 1 Y ,

where the '1' in the second row is the code for 'S' or 'store'.

Opposite word no. 10 of the conversion routine we write '1 A', a mnemonic for '<u>add</u> the number n_1 to the address-part of the word which follows on the tape'. Similarly, opposite words nos. 9,10,12,15 of the sine routine we write '2 A', with corresponding significance. The pattern on the tape corresponding to the symbol m A consists of two rows:

0 m

0,2 Y

where the '2' in the second row is the code for 'A' or 'add'.

The symbols mS, mA are called control-designations. There are three other control designations, as follows

'm, q T', meaning '<u>transfer</u> the number (m,q) to register C', with the tape pattern (two rows) m, q
0, 0 Y ;

'R' meaning '<u>repeat</u> the control operation last executed', with the tape pattern (one row) 4 Y ;

'D' meaning '<u>do</u> the command represented by the word next following on the tape', with the tape pattern (one row)

6 Y .

5.2 <u>Primary and Control routines</u>.

The primary-and-control routine which, by the obeying of its commands, secures the performance of these control operations is

<u>Ex. 26</u>. <u>Primary routine (second version) and control routine</u>.

<u>Primary</u>	0	$(D_0) \xrightarrow{+} D_0$	
	1	$s(D_0) \xrightarrow{c} S$	
	2	$(H_u) \xrightarrow{+} A$	
	3	$s(D_0) \xrightarrow{c} S$	
	4	$(S) \xrightarrow{+} S$	
	5	$(H_1) \xrightarrow{+} A$	
	6	$(B) \xrightarrow{+} S$	gives (S)' = 0,18 if preceded by command 21
	7	$(C) \xrightarrow{+} K$	
	8	$c(A) \longrightarrow O$	
	9	$p_{11} \xrightarrow{+} C$	
	10	$(I) \longrightarrow D_0$	
	11	$(D_0) \longrightarrow H_1$	
	12	$s(D_0) \xrightarrow{c} S$	
	13	$0 \longrightarrow S$	
<u>Control</u>	14	$(H_u) \xrightarrow{+} S$	
	15	$(0,24) \xrightarrow{+} A$	entry point called by m,q; OY, for Transfer
	16	$(0,23) \xrightarrow{+} A$	entry point called by o,m; 1Y,

18	c(A) —— 0,19	
19	[p_{20} —— T]	becomes the executive command, in virtue of command 18; called by 4Y, for Repeat
20	31,21 $\overset{+}{\text{——}}$ K	
21	0,11 —— B	entry point called by 6Y, for Do
22	0,10 —— S	
23	< 0,0,13,27 >	
24	< 31,8,12,14 >	

The primary part of this is, apart from command 6, merely a rearrangement of Ex.24, whose working has been explained. To see how the ingenious control routine works we note first that when command 10 is obeyed with a Y(p_{20}) in I that has come from the tape-row 0, nY, command 13 will be skipped on account of the Y, at 14 we shall have (H_u) = 0,n, and the next command will be 15,16,17,19 or 21 according as n = 0,1,2,4 or 6.

Suppose now that five consecutive tape-rows are

(1) a lower half-word accompanied by X-tag
(2) 0, m
(3) 0, 1 Y
(4) an upper half-word
(5) a lower half-word accompanied by X-tag,

and let the initial state be that B is clear and row (1) is in I, with command 10 about to be obeyed. This will lead to a traverse of the primary in which A is cleared and (C) finishes at the value r, say. Then on command 10 the half-word (0, m) goes into D_0 and on the succeeding traverse is put into the upper half of A, with (C) remaining at r. On command 10 again the word (0, 1 Y) goes into D_0, and we arrive at 14 with (H_u) = 0,1, so the next command taken is 16, entered with (A) = (0,m,0,0).

Command 16 now adds into A the word (0,0,13,27) from cell 23, and the following command adds into A the word (0,24,0,5) from cell 15. This gives

$$\begin{array}{rrrrrl} (A)' = & 0 & m & 0 & 0 & = (0,m+24,14,0), \\ & +0 & 0 & 13 & 27 & \\ & +0 & 24 & 0 & 5 & \end{array}$$

which is the pulse pattern for the command (C) —— m+24. At command 18 this pulse-pattern is substituted into cell 19, so command 19 becomes (C) —— m+24, <u>which is obeyed</u>. Hence the number r in C is stored in cell $\overline{(m+24)}$, which is the <u>m</u>th after the last cell occupied by the control programme.

Next, commands 20 and 21 are obeyed, and 20 causes 21 to become 0,0 —— B, which leaves B clear. Then on command 22 we return to 10 and tape-row (4) is sent into D_0. There follow two traverses of the primary, entered with A clear (on account of the preceding command 18), and on the second one the word assembled from tape-rows (4), (5) is stored in cell r.

The effect therefore is that the number r of the cell where this word is stored has itself been stored in cell (m+24).

Suppose now that at any later stage a pair of tape-rows

0, m

command 17 will be entered with (A) = (0,m,0,0); and this command adds (0,24,0,5) to (A), giving (A)' = (0,m+24,0,5) which represents the command

$$(m+24) \xrightarrow{+} A .$$

At 18 this is put into cell 19 and A is cleared, and at 19 the command is obeyed, which gives (A)' = r since (m+24) = r. Hence the following traverse of the primary, (which as before is entered via commands 20,21,22,10) is made with (A) starting at the value r, and the word which is assembled during this traverse and the following one will have r added to the upper half as read from the tape. In consequence, if t is the upper half (i.e. an address) punched on the tape, the word as stored will have the upper half r + t; and this is the effect which was desired to be secured by the control tape-rows 0, m; 0, 2 Y (which the programmer punched in response to the designation m A on the written programme).

In the same way it will be found that, when entry at command 15 is called by the tape-rows m, q; 0 Y, the pulse-pattern put into cell 19 is (m,q,26,14) representing the command m, q —— C. The obeying of this command puts the number m,q into C, so the command next assembled by the primary will be stored in cell m,q. The code m,q ; C Y accordingly calls for a change in the storage-cell number, and we thus have a control operation whereby we can start the storage of words (read from tape) in any desired cell, or break the sequential storage at any desired point.

If 'Do' is called by the code 6Y the control routine is entered at 21, so we get (B)' = 0, 11. Then follows the assembly of the next word from its tape-rows, and on the X-tag we arrive at command 6, whose effect is to give (S)' = 0,18. Hence the word already assembled in A is placed in cell 19, and the command which its represents is forthwith done; and then 20, 21 give (B)' = 0. The commonest use of this facility is to transfer control to any desired point of a programme that has just been read and stored; at the end of the tape we punch, say,

6 Y

k —— S ;

the command is obeyed, so the number k is put into S, and the following command to be obeyed will be the one in cell k.

Finally the code 4 Y leads to entry to the control routine at command 19, so whatever command was placed there by the preceding control operation (and then obeyed) will forthwith be obeyed again.

5.3 Programme assembly and tape layout.

Consider a complete programme consisting of a number of routines and a master. The master will at some stage call each of the routines by a command containing the number of the cell where the head-word of the routine is stored; and some routines may similarly refer to others. Also each routine may contain words referring to the cells where it itself is stored, e.g. word 10 of Ex. 25. The routines now are to be arranged in such an order that each of them refers only to itself or to those that come earlier in the sequence (which nearly always can be done), and last will come the master because it refers to all the routines.

To the routines and master, in this order, are attached the control-designations 1S, 2S, 3S, ... Each of them is written as though for storage in a set of cells starting at cell 0, but their head-words will actually be stored in cells n_1, n_2, n_3, ..., say. Then for the cross-references or internal references to be correct it will be necessary that certain words have their address-this adjustment the control designations 1A, 2A, etc. are appended. The tape is punched with the control cues corresponding to 1S, 1A, etc. in the proper places, and is read into the machine via the primary and control routines.

The tape is headed by the primary (words 0-13 of Ex.26). Immediately following this are the words

14	$(H_u) \text{ —— } C$
15	$(D_o) \overset{-}{\text{——}} D_o$
16	$(Z) \text{ —— } H_u$
17	$0 \text{ —— } S$

which are later overwritten. This is forced-fed into the machine as described in §4.2. Then follow

	blank rows	
	0,14 Y	
14	$(H_u) \overset{+}{\text{——}} S$	) control routine, stored in
	. . . .	) cells 14-24 by operation of
24	< 31,8,12,14 >	) the primary
	m, n	} cue for m,n T meaning 'store
	0, 0 Y	} the next following word in cell no. m,n
	0, 1	} cue for 1S : store head cell
	0, 1 Y	} number of routine 1
	1st word of routine 1 (two halves)	
	. . . .	
	0, 1	} cue for 1A: adjust address-
	0, 2 Y	} part of the following word
	. . . .	
	0, 2	} cue for 2S: store head cell
	0, 1 Y	} number of routine 2
	. . . .	
	. . . .	) master, designated say by kS
	. . . .	)
	0, k	} cue for kA
	0, 2 Y	}
	0, 6 Y	cue for Do
	q —— S :	next command obeyed will be the (q+1)st of the master, numbered q as the master was written.

5.4 <u>A modified assembly scheme</u>.

If the only cross-references are between the master and the routines, i.e. if each routine makes no reference to other routines - as is often the case - there is a modified assembly scheme which is logically more complicated but physically simpler than that of §5.3. <u>Each</u> routine is written with the head control-designation 1S and its self-references designated by 1A, and is correspondingly punched. But the routines, taken in any order, are also numbered 2, 3, ..., and references to them <u>in the master</u> carry the control-designations 2A, 3A, ... The master carries the head designation 1S and its self-references are designated by iA.

In punching the tape, the routine numbered 2 is preceded by the cues for both head designations 2S and 1S; and similarly for the other routines, so that the layout of the tape is (symbolically)

```
primary and control
 n, n T
 2S
 1S
routine no. 2
 3S
 1S
routine no.3
. . . . . .
 1S
master
 1 A D q —— S
```

The reading of the cues 2S, 1S will cause the head cell-number for routine 2 to be stored in both the cells 24+1 and 24+2; for the self-references of the routine the storage in 24+1 will be operative (via the designation 1A) but for the references to the routine by the master the storage in 24+2 will be operative (via the designation 2A). Then when the cues 3S, 1S are read the head cell-number of routine 3 will be stored in cell 24+1 (overwriting its previous content) and in cell 24+3; and so on, until finally cell 24+1 contains the head cell-number of the master.

The virtue of this scheme is that those routines which are self-contained can be punched once and for all, and by the CSIRAC editing equipment can be copied from library tapes on to the tape for the desired calculation. In this way punching is reduced to a minimum and punching errors are minimized.

The library routine-tapes for CSIRAC are all punched with the control-designations 1S, 1A.

In designing a programme which is to be written and assembled with use of the control routine we must of course remember that cells 1-24 are earmarked for the primary and control and that a certain number of cells 25, 26, ... will contain the head cell numbers of the routines and the master. The head cell for the specific programme must be chosen so as not to overlap with this preliminary block. Once the specific programme has been stored, however, this preliminary block has fulfilled its function and it can be overwritten by numerical data or otherwise as directed by the programme proper (unless it is later to be used for reading a continuation of the tape).

5.5 Tape editing procedure.

Tapes can be copied as follows: The punch is connected to the machine, the tape to be copied is inserted in the reader, the machine is switched so as to take its commands from N_1, and on N_1 the command (I) —— O_p (code 0,0,1,3) is hand-set. When the machine is then run, continuously or single-shot, a duplicate of the tape will be punched. The next word to be punched is the one shown on the input monitor lights.

To interpolate new tape-rows on the copy the machine is stopped at the appropriate point and the setting of N_1 is changed to (N_2) —— O_p (0,0,3,3). The desired tape-row is then set up on N_2 and punched by single-shot. The next desired tape-row is then hand-set on N_2 and punched, and so on.

To discard a tape row that is on the original but not desired on the copy, the setting of N_1 is changed to (I) —— I (0,0,1,1) send it to limbo and the next-tape-row will be positioned in the reader.

This procedure is used for making minor amendments to programme tapes.

5.6 Machine operation.

Incidental reference has been made, in this and the preceding Chapter, to some of the machine operation procedure, and we may conveniently mention here the chief switch-board facilities:

Master stop-start switch
Stop-start button
Single-shot switch
'Take command from N_1' switch
'Take command from N_1 and S' switch
Clear-buttons for various registers
Switches for hand-setting N_1, N_2 and I
Switch for normal speed or speed up (continuous running)
Switch and control knob for rapid single-shotting
Switches for displaying contents of selected blocks of mercury-store cells
Switches for inhibiting 'add 1 to S' and tape-read
Trigger stop switches; by appropriate setting of these the machine can be made to stop after obeying the command in any desired cell. A common use is to make the machine stop with any desired number in S.

It may be added that the machine is reasonably fool-proof.

CHAPTER 6. Miscellaneous.

6.1 Errors.

Experience has shown that it is very difficult to make a programme that is free from all error. It is wise to 'reason through' the draft a number of times, at intervals, and to have someone else read it critically. An error may cause a programme to run wild - which will be evident - or it may merely cause arithmetical mistakes; to detect these it is wise to check specimen results against those found by hand computation. An error may lie in the written programme or in the punched tape; as regards the latter the tape should always be read against the written programme, and in practising this the binary symbols and the code numbers for the sources and destinations will be learnt. If the programme is run on the machine by single-shot the successive commands as obeyed are shown on monitor lights.

It is impossible to catalogue all possible sorts of error in programme writing, but some common ones can be mentioned:

1. Adding or subtracting into a register which ought to be but is not explicitly clear. Multiplying when A is not initially clear.

2. Overwriting a number which will later be required; or destroying a number later to be required by operating on it, e.g. in counting.

3. Incorrect counting.

The more subtle forms of these arise when the operations are interior to a loop. A loop has different sequences of commands preceding (1) its first entry and (2) later arrivals at the entry point, and it must be checked that at all the entries the initial settings of such registers as are assumed in the loop are correct.

4. Errors in programming encapsuled loops: for example there may be failure to re-set the count for an inner loop.

5. Failure to provide for limiting or special cases. The special values of variables most likely to cause trouble are 0 and -1. An obvious case is that machine multiplication gives $(-1)(-1) = ?\,-1$; either one must be certain that this case of multiplication can never arise, or one must provide in the programme for its detection and appropriate treatment (see Ex.24 below). A more subtle case is the criterion '$|a| \geq |b|$ if the sign digit of $|a| - |b|$ is zero', which is important in solving linear equations. For $a = 0$ and $b = -1$ the machine gives $|-1| - |0| = ?(-1) - 0 = -1$; and working with the minus-the-modulus is no better: $-|0| - (-|-1|) = 0 - (-1) = ? - 1$.

6. Failure to allow for rounding errors. For example (1) a variable may theoretically tend to zero, and the exit-test from a loop may be based on this fact; but the variable is not represented in the machine exactly, and it may happen that the machine value, having become reduced say to 2^{-19}, remains there or oscillates between $\pm 2^{-19}$. (2) A rounding error may cause a number which is theoretically within (-1, 1) to lie beyond this range.

7. Correcting an error and thereby making a different one.

8. When constructing the programme, altering some detail and thereby introducing inconsistency with other parts of the programme.

9. Sheer blunders, arising from inattention, or from disregarding the obvious because attention has been directed towards avoiding the more subtle traps.

6.2 Keeping within machine capacity

Most computations concern numbers that are capable of quasi-continuous variation rather than numbers, like integers, that can vary only by multiples of a finite unit, and it is this sort of computation that we shall consider. If the binary point is taken in the standard position, special steps will usually have to be taken either to secure that no number x outside $-1 \leq x < 1$ occurs, or to detect any number outside this range and deal suitably with it.

By preliminary hand computing regarding a specific problem it is sometimes possible to determine in advance the range of values that will be encountered. If this spreads outside $(-1, 1)$ two courses are open. The first is to scale down the problem-data so that the range is brought within $(-1, 1)$, and finally scale up the answers obtained for the modified problem; this final up-scaling may be left to hand-computing, or done on the machine by a suitably programmed print-routine. The other course is to adopt a non-standard position for the binary point; machine addition remains valid, but each machine multiplication must be followed by an appropriate left-shift for the proper representation of the product on the adopted convention. For extracting the answers a special point-routine will of course be necessary.

Sometimes the scaling can be a programmed one, and moreover not the same for all values of the variables. For example, suppose the value is required of

$$f(\theta) = \frac{a(c - \cos\theta)^2 + b\sin^2\theta}{1 - k\cos\theta}\,, \qquad (1)$$

for a succession of values of θ from 0 to π, regularly spaced; here a, b, c, k are data-constants with $0 < a, b < \frac{1}{2}$, $0 < c, k < 1$. For $0 \leq \theta < \frac{1}{2}\pi$ the numerator and denominator can be handled on the standard convention, while for $\frac{1}{2}\pi \leq \theta \leq \pi$ we can handle

$$\tfrac{1}{2}f(\theta) = \frac{a(\frac{c}{2} - \frac{\cos\theta}{2})^2 + b(\frac{\sin\theta}{2})^2}{\frac{1}{2} - k\frac{\cos\theta}{2}}\,. \qquad (2)$$

The programme is essentially a loop in which θ is suitably increased after each traverse. The loop starts with a test whether $\theta \geq \frac{1}{2}\pi$, and a switch (say D_2 as in §3.5) is set to be 0 if this is so and -1 otherwise. We then programme so that (1) or (2) is calculated according to the state of the switch; for example if $(A) = \sin\theta$ the commands

$$s(D_2) \xrightarrow{c} S$$
$$\tfrac{1}{2}(A) \longrightarrow A$$

will give $(A)' = \sin\theta$ in case (1) but $\frac{1}{2}\sin\theta$ in case (2). The programme must similarly provide for doubling the quotient when $(D_2) = 0$, so that (2) may give $f(\theta)$. (The idea in making the dichotomy is to conserve accuracy for θ near 0; in the present case there is not much so gained, but it would be important if, say, the fourth power of $f(\theta)$ were required.)

If the range of the variables cannot be foreseen in advance it will be necessary to adopt some plan for representing in the machine numbers that are far outside the range $(-1, 1)$. There are two possibilities. One is to use two registers or cells to hold a number: one for the integral part (with sign) and the other for the (positive) fractional part; this is called <u>double-length</u> representation. The other is to represent a number in the form $2^n x$ where $-1 \leq x < 1$ and n is an integer; then one register is used for x and another for n, in p_1-units say. This is called a floating-point or <u>floating</u> representation. By the first method it will often be a reasonable assumption that all numbers that occur satisfy this condition. The second method provides for enormously larger numbers. Both methods require that the simplest operations be performed by routines. For example let $x = (A.B)$ and $y = (D_0.D_1)$; the representation is supposed such that the p_{19}-digit of (B) and (D_1) has the weighting 2^{-1}, that the p_{20}-digits of these registers are zero, and that the p_1-digits of (A), (D_0) have the weighting 2^0. Then:

Ex.27. Double length addition: $x+y$ is given by $(D_0.D_1)'$, where

0	$(A) \xrightarrow{+} D_0$	add integral parts
1	$(B) \xrightarrow{+} D_1$	add fractional parts
2	$(D_1) \longrightarrow B$	add carry digit, if any, into D_0
3	$(R) \xrightarrow{+} D_0$	
4	$s(D_1) \xrightarrow{+} D_1$	and cancel it from D_1

In what may be called 'fully-floating arithmetic' each number $2^n x$ is represented with its own index n, and the restriction that $\frac{1}{2} \leq x < 1$ is often made. There is also possible a 'block-floating' representation, in which all numbers are represented with the same index n, which at any stage will be the largest that is required for any individual number; this representation may conveniently be used in matrix operations, and has the advantage of conserving store-space.

In particular problems a judicious mixture of single length, double length and floating representations will suggest itself. Thus in calculating $f(\theta)$ above, one would use a floating representation of the quotient and convert this to double-length representation for the convenient printing of $f(\theta)$.

Ex. 28. Addition in block-floating representation.

The numbers to be added are $2^n x$, $2^n y$, where $(D_0) = np_1$, $(A) = xp_{20}$, $(C) = yp_{20}$, so that x,y are fractions (or -1) with the standard representation. If x,y have opposite signs, or if either is 0, their sum is in $-1 \leq x+y < 1$. If they are both negative their addition overcarries(1) if $x+y < -1$; the sign digit of the machine sum is 0, it represents -2, and is different from that of x or y (as in the other case).

To deal with the overcarry we have to halve the true sum and add 1 to the index n. In the positive case the true sum 1.abc .. has the true half 0.1abc .., but the machine half is $\bar{1}$.1abc .. . In the negative case the true sum $\bar{2}$ + 0.abc .. has the true half $\bar{1}$·0abc .. but the machine sum 0.abc .. has the machine half 0.0abc .. . In both cases machine addition of p_{20} to the machine half gives the true half. Hence:

0	$s(A) \longrightarrow D_1$	
1	$s(C) \xrightarrow{+} D_1$	$(D_1)' = s(x) + s(y) = \begin{cases} 0 \text{ for like signs} \\ 1 \text{ for unlike signs} \end{cases}$
2	$(C) \xrightarrow{+} A$	$(A)'$ = machine sum $x+y$
3	$s(A) \xrightarrow{+} C$	$(C)' = y + s(x+y)$
4	$s(D_1) \xrightarrow{c} S$	$s(C)' = \begin{cases} 0 \text{ if } y, x+y \text{ have same sign digit} \\ 1 \text{ '' '' '' opp. '' ''} \end{cases}$

(1)

SC CS	5	$s(C) \xrightarrow{c} S$	
4 K PS	6	$0,4 \xrightarrow{+} S$	taken if $s(D_1) = 1$ or $s(D_1)$, $s(C)$ are both 0; no overcarry
HA A	7	$\frac{1}{2}(A) \longrightarrow A$	overcarried; correct half-machine sum
PS PA	8	$p_{20} \xrightarrow{+} A$	
PL PD	9	$p_1 \xrightarrow{+} D_0$	increase index
1 PS D	10	$p_{20} \longrightarrow D_1$	set overcarry indicator
	11	next command	

The programme would now have to provide for halving the fractional parts of all the numbers that have previously been represented with the index n. We shall not discuss how this is to be programmed beyond remarking that some indication must be provided that the index has been increased, and this is the purpose of command 10.

In fully-floating representation each number has its own index, and the above commands for addition must be preceded by a sequence which determines which of the two addends has the greater index and then 'up-floats' the number with smaller index.

6.3 Multiple precision representation.

Sometimes numerical data cannot be represented with adequate accuracy by single words, and a multiple-precision representation must be used. For the double-precision representation of a number x in $-1 \leq x < 1$ two registers are used; the most significant or upper half of the number is in one register with the standard weightings of the digit-positions, and the lower half is in another register with the p_{20} position left blank with the weight 2^{-20} attached to the p_{19} position; the digits in this register all have positive weighting(1). This double precision representation resembles the double-length representation referred to in §6.2, and involves similar programming for the fundamental operations: the distinction is that in the context of §6.2 it was supposed that the data had only single-length precision (equivalent to about 6 decimals), and double-length representation was introduced to deal with overcarries.

The CSIRAC library includes routines for performing the fundamental operations in fully-floating and double-precision arithmetic. Each routine covers the four operations of addition, subtraction, multiplication and division; the particular operation desired is 'called' by entering the routine at a stated point. Such multi-purpose routines are sometimes called function-blocks.

6.4 Style. Dodges.

In learning programming it is well worth while to strive for efficiency, i.e. for doing the required job without wasting commands; if good habits in this matter are formed, dividends will be reaped on occasions when store-space is tight. There are general points, such as making the most use of a given multiplier once it has been set, or using a certain register-content for two different purposes, and there are points that are special to CSIRAC (or whatever the machine may be) so as to make the best use of its individual facilities. Minimal programming is of course especially desirable in the fundamental routines, and it does not at all follow that because these have been drawn up by 'professionals' they are incapable of improvement.

---

(1) This treatment of the lower half is not the only possible one, but is the most convenient.

The examples in this Manual show, incidentally, a number of devices which in the first instance may not have been obvious, and these may be useful in other contexts. Here are a few more.

Ex. 29. To allow for the machine-product (-1)(-1) = ?(-1). It is supposed that a number of products of numbers in standard representation are to be added and that the sum may extend outside the range (-1,1). The sum is accumulated in D_0(integral part) and D_1 (fractional part) and it is supposed that the term (product) currently to be added into $(D_0.D_1)$ is in A. Then s(A) represents -1 unless (A) = (16,0,0,0), and then (the critical case) s(A) represents +1. Hence

0	$(A) \xrightarrow{+} D_1$	
1	$s(A) \xrightarrow{+} D_1$	cancels sign-digit of (A) from D_1
2	$s(A) \longrightarrow B$	
3	$p_{20} \xrightarrow{+} A$	(A)' = 0 in the critical case only
4	$\bar{z}(A) \xrightarrow{+} K$	
5	$(R) \xrightarrow{+} D_0$	adds s(A) to D_0 in the critical case, but in any other case subtracts it
6	$(D_1) \longrightarrow B$	transfers carry digit from D_1 to D_0
7	$(R) \xrightarrow{+} D_0$	
8	$s(D_1) \xrightarrow{+} D_1$	

Ex. 30. A use of ∙A (destination no. 7). One use is to 'isolate' the digits in a set of selected positions of A. For example to isolate the set in positions $p_1 - p_5$ let the word (0,0,0,31) be set into B, this being chosen as having 1's in the selected positions and zeros elsewhere. Then $(B) \xrightarrow{\cdot} A$ will give (A)'with the $p_5 - p_1$ digits unaltered but zeros elsewhere.

Ex. 31. A use of vA (destination no. 8). Suppose some process is to be repeated until two independent conditions are both satisfied, e.g. some operation is to be performed on the elements of a matrix, in serial order within each row with the rows taken in serial order, until a 'stated' element in a 'stated' row is reached. For one of the conditions we arrange that $s(D_0)$ shall be 0 if the condition is satisfied but 1 otherwise; and similarly s(A) for the other condition. Then $s(D_0) \xrightarrow{v} A$ will give s(A)'= 0 only if both $s(D_0$ and s(A) were 0, i.e. if both conditions are satisfied.

Ex. 32. A use of ≠ A (destination no. 9). This operation produces from two digit-trains a third digit-train, treating the values 0,1 on the same footing. The following proposal for generating a sequence of 'random' numbers is based on this property, and on the fact that, for almost any machine-product, the upper and lower halves have digit-patterns that are, roughly speaking, unrelated.

0	1S		$(D_0) \longrightarrow C$	
1	1A		$(2) \longrightarrow A$	
2		15,21	$c(A) \xrightarrow{x} B$	the prefixed addresses are relevant for commands 1,4 but not for 2,3.
3		23,22	$(B) \xrightarrow{\neq} A$	
4	1A		$(3) \xrightarrow{\neq} A$	
5			$(A) \longrightarrow D_0$	
6			$p_{11} \xrightarrow{+} D_{15}$	
7			$(D_{15}) \longrightarrow S$	

Ex. 33. Repeated +K commands. A sequence such as

$$\begin{array}{ll} 0 & (D_r) \xrightarrow{+} K \\ 1 & (D_s) \xrightarrow{+} K \\ 2 & (0) \longrightarrow A \end{array}$$

is sometimes useful. If $(D_r) = ap_{11}$ command 2 becomes $(D_{s+a}) \xrightarrow{+} K$, and 3 will transfer to A the word in the cell whose number is in D_{s+a}. In this line of country traps such as

$$\begin{array}{lll} 0 & (D_1) \xrightarrow{+} K & \text{with } (D_1) = mp_{11} \\ 1 & (D_0) \longrightarrow 0 & \end{array}$$

may be noted; the command obeyed at 1 will be $(D_m) \longrightarrow m$, which is unlikely to be what was desired.

Ex. 34. Patching. Sometimes it is found, at a late stage in drafting a programme or after the tape has been punched, that an extra block of commands should be added. To insert this block in the proper place, after command n, say) will involve re-numbering of later commands and hence, in general, alteration in sequence-change commands. To avoid this, put the new block at the end of the programme, starting in cell m say; prefix it by command n; replace command n by the sequence-shift $m \longrightarrow S$; and finish the block with the sequence-shift $n+1 \longrightarrow S$.

CHAPTER 7. Programming Strategy.

The previous chapters have been concerned with the means whereby the machine may be made to perform such detailed operations as may be desired. There remain certain wider questions of choice of method and programme design, which may be called strategic:

(i) choice of mathematical method for solving a specific problem,
(ii) choice of accuracy to be aimed at,
(iii) relative weight to be given to economy in store-space, machine-time and programming-time,
(iv) overall programme design, including
(v) flow diagrams, and
(vi) checks.

These questions are to some extent interlocked. The relevant generalities are fairly obvious, and as the discussion will be confined to these it will be brief. Practical exemplification may be seen by studying the complete programmes and routines in the CSIRAC library.

7.1 Mathematical methods.

For the numerical solution of a given problem there are usually many methods available, whose differences from one another may be anything from quite fundamental to trivial. Information may be sought from books on Numerical Analysis, of which there are many, from certain periodicals, or from the CSIRAC staff. Amongst the different methods the choice may well fall on the one that is most easily programmed, which may not coincide with the one that would be chosen for hand computation. This contrast extends down to programme details such as the evaluation of function-values; the hand computer gets them from tables, but for the automatic machine it is nearly always preferable to calculate them from a definition of the function or a suitable approximation to it; rather than to store a table and programme the interpolatory reading to it. Some recent books pay special attention to the contrast.

7.2 Rounding errors. Checks.

In addition processes absolute errors are relevant, but in multiplication processes proportional errors are relevant. Almost any calculation involves the two processes intermingled, and the growth of absolute errors will be very largely dependent on whether the multipliers involved are large or small. In consequence it is impossible to make any general statement about the propagation of rounding errors. If n numbers, each of which may be in error by β, are added the root mean square error in the sum, taken over a large population of such additions, is on the usual statistical assumptions about $0.3\,\beta\sqrt{n}$; and with considerable reserve this may be taken as indicating the accuracy to be expected in a computation involving n elementary operations, where for numbers represented on the standard convention $\beta = 2^{-20}$. On the whole, then, for a calculation on CSIRAC starting from 6-decimal data one can seldom hope for better than 4 significant decimals in the answers. If better accuracy is needed it will be necessary to work to double precision.

As a check on rounding errors and on machine faults (which can and do occur) the programme should provede for the same sort of checks that are used in hand computation. This is specially important when (as in solving equations, for example) one cannot expect any check from the regularity of the results. A convenient check for the final stage of output has been indicated at the end of §4.5.

In order thet everything may not be lost if a machine-fault occurs during the running of a programme it is wise to arrange that end-results be printed as they are obtained, or perhaps in batches; And if the programme falls into distinct sections the end-results from one section may be punched even though they are not final answers; punched rather than printed so as to facilitate their input if the next section has to be repeated. We have not yet got enough experience to state anything precise as to the average time of fault-free running; it is suggested that results be extracted, if possible, at intervals not exceeding 15 minutes.

7.3 Economy, in time and space.

It is nearly always the case that one cannot have simultaneously minimal store-space and minimal operating time: the sequence-shift commands whereby loops are formed and store-space saved take time to execute. Another consideration is the time taken to construct the programme: for an ephemeral job one may throw together anything that will work, but for a big project or a programme of permanent importance one will take trouble to economize the machine-operation time.

Sometimes, however, the dominant consideration will be that store-space must not be exceeded. To save store-space the following artifices may be possible:

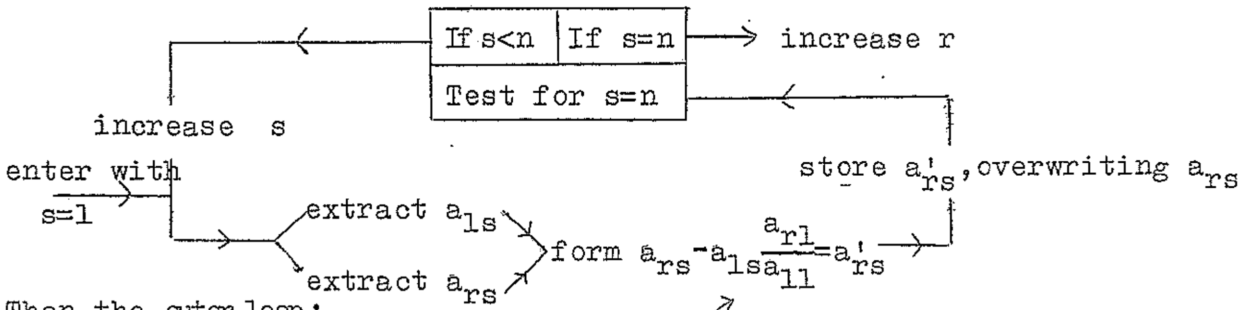
(1) The programme may fall naturally, or may perhaps be arranged, in sections each of which is self-contained. Then at any one time only the current section need be in store; and the programme may direct that when each section is completed the tape carrying the next section be read so as to over-write in the store the previous section. The reading will of course be done by calling in the primary and (if it is used) the control, which will already be in store (and must not have been overwritten); and one must secure that the appropriate registers are in the correct initial state. For example, if the first version of the primary (§4.2) is being used, and if the tape to be read is already positioned (as it will be if it is a physical continuation of the tape previously read, with its first word following without gap after the control Y at the end of the previous tape), the appropriate commands are A SA, 14 K C, 10 K S; but if the tape requires positioning the commands should be A SA, 14 K C, D SD, PS T, and S will be hand-cleared to zero, as for feeding a data-tape (§4.3).

(2) Instead of storing all the data at the start it may be possible to read them from the tape as required - either by blocks or by single items. The programme will then alternate between sets of commands for reading (and perhaps storing also) and for calculating. For example, to form the product of two matrices one of them must be completely in store, but it is just as convenient to read and calculate from the second one row-by-row as to store it all at the start.

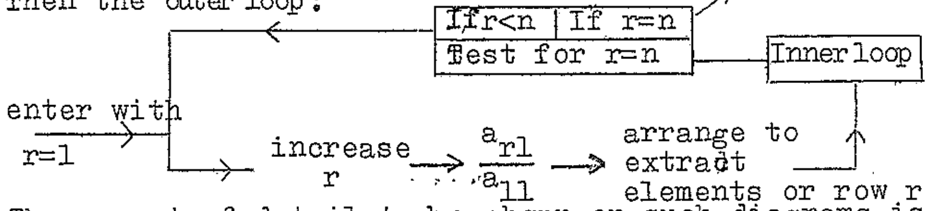
7.4 Crystallizing the design.

A finished programme is the end-product of a process that starts from a general design and proceeds by crystallization down to finer and finer detail. For the crystallization to be correct it is necessary that the programmer should have firmly in mind the logical structure of what must be done: the order in which operations are made, the places at which tests must be made regarding alternative continuations, and so on. For this purpose it is often helpful to construct <u>flow-diagrams</u> showing the logical relations. A flow diagram is an intermediary between the general design and the finished programme, and several such intermediaries, at different levels, may well be desirable.

As an example of flow diagrams the following relate to Ex. 18 (§§ 3.1,3.4). First the inner loop:



Then the outer loop:



The amount of detail to be shown on such diagrams is largely a matter of taste; for example the test for r or s = n involves the setting of a count, or something equivalent, but this has not been shown explicitly. For those who find that they can hold in mind the logical relations, flow diagrams are not necessary; but they are always a considerable help to a person reading someone else's programme.

Select list of library routines.

For most processes there are a number of variants, in each of which certain advantages (e.g. high accuracy) are offset by certain disadvantages (e.g. length). The following list is a select one. It is confined to the operations most commonly required, and as regards these it gives only one or two of the available variants; where two of them are shown, the emphasis is in the one case on shortness and in the other on inclusiveness and accuracy. For further details the library must be consulted. For routines which find some function f(x) of a single variable x it is understood that the routine is entered with (A) = x and finishes with (A)' = f(x), unless the contrary is stated. In each case the control designation is 1S, and the link is in D_{15}. The working space is A,B,C,D_0 except in cases noted.

Function	Initial and final states; notes	Restrictions	L'gth	Extra work'g space	Method
Division	$\frac{x}{y}=\frac{(A)}{(C)}=(A)'$;systematic error is possible	$\lvert x\rvert<y<1$	14	-	Repetitive
ditto	$\frac{x}{y}=\frac{(A)}{(C)}=2^{(D_0)'}(A)'$,where in critical cases (A)' is never -1	$y \neq 0$	29	D_1,D_2	Long division, non-restoring
Reciprocal	$-\frac{1}{2y}=-\frac{1}{2(A)}=2^{(D_0)'}(C)'$; in critical cases,e.g. $y=\frac{1}{2}$, (C)' = -1	$y \neq 0$	24	D_1,D_2	Iterative
Square root	$\sqrt{x}=\sqrt{(A)}=(C)'$	$0<x<1$	15	D_1	Trial and error
Exponential	$x=(A), e^x-1=(A)'$	$-1\leq x<\ln 2$	11	-	Repetitive
ditto	ditto	ditto	19	-	Series[Ж]
Logarithm	$x=(A), \log_2(2x)=(A)'$	$\frac{1}{2}\leq x<1$	16	-	From definition

ditto	$x=(A), \log_2 x=(A)' \equiv (D_0 D_1)'$	if $\frac{1}{2}<x<1$
		if $0<x<1$
			22	-	'' ''

$\sin \frac{1}{2}\pi x$ $\cos \frac{1}{2}\pi x$	gives sin or cos acc. to point of entry	$0\leq x<1$	22	-	Series[Ж]
$\sin \pi x$ $\cos \pi x$	'' '' ''	$-1\leq x<1$	28	-	''
$\tan \frac{1}{2}\pi x$		$-\frac{1}{2}\leq x<\frac{1}{2}$	22	-	''
$\frac{2}{\pi}\text{arc}\sin x$		$\lvert x\rvert\leq 1/\sqrt{2}$	29	-	''
$\frac{2}{\pi}\text{arc}\tan x$		$-1\leq x<1$	29	-	''

[Ж] Curtailed and with appropriately modified coefficients.

Select list of library routines.

Function	Initial or final states;	Restrictions	L'gth	Extra work'g space	Notes
Print integers	$(A) = xp_1$	$\|x\| < 524288$	27	D_1	Prints sign and suppresses initial zeros
Print fractions	$(A) = xp_{20}$	$-1 \leq x < 1$	20	-	Prints -1 as -.999998
Print sterling (money)	$(A) = xp_1$ where amount is x pence	$0 \leq x < 524288$	44	$D_1D_2D_3$	Conventional print layout
Input integers	$(A)' = xp_1$	$\|x\| < 524288$	15		From tape punched with one dec. digit per row, with 7th row to indicate sign and nature
Mixed input	$(A)' = xp_1$ for integers; $(A)' = xp_{20}$ for fractions	$\|x\| < 524288$; $-1 < x < 1$	22	H	From tape punched with one dec. digit per row, with 7th row to indicate sign and nature
Input fractions (compact punching)	$(A)' = xp_{20}$	$-1 < x < 1$	29	HD_1	Sign tag included in 3rd row

Tape symbol print (complete programme, length 161). Reads a programme tape and prints its command-words (including control designations) in conventional symbolism, given on p.17.

Print fractions from store (complete programme, length 50). Prints contents (as fractions) of a block of cells specified by hand-setting on N_1 and N_2. Used in emergency or as post mortem.

CORRIGENDA (omitting a few trivial misprints)

p.2	Fig.1: the amplifier must be between the delay-line and the source-gate.
p.7	par.2, line 4: "3 x 10^6 per second".
p.17	In the blank spaces in col.6, rows 7,8,9, insert "CJ, DJ, NE" respectively.
p.28	par.4, line 5: replace "S" by "T" in two places. Section no. "2.7" should be "2.6".
p.35	The top par. may be a little misleading: standard practice is to use D_{15} for inner routines and D_{14} for the outer one
	line 4 from foot: the second "a_{n+1}" should be enclosed in modulus signs.
p.57	line 17: Place modulus signs round "x".
	Ex.28. In line 4 insert sentence "If they are both positive their addition overcarries$^{(1)}$ if $x+y \geq 1$; the sign digit of the machine sum is 1, it represents +1, and is different from that of x or y."