

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

August, 1959

Reformatted into a more modern style by
Bill Purvis
February, 2021

CHAPTER I. INTRODUCTION.

1.1 Background.

CSIRAC is a general purpose automatic electronic digital computer. The word electronic means that it uses electronic circuits and valves. 'Digital' means that numbers are represented in the machine by discrete entities (actually by trains of electrical pulses), similar to the familiar representation of decimal digits; in contrast there are 'analogue' machines in which each number is represented by a single physical quantity, such as 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 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 and many logical operations also, provided the programme and its data can be fitted into the stores of the machine.

To 'programme the calculation' means to draw up a list 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 will need to know which commands the machine can perform and how to organize these operations into a complete programme. It is sometimes helpful, but not necessary, to understand the physical details of machine organization, some of which are treated in the appendices.

In many programmes, most of the commands are copied from previously tested programmes, which have been organized in a 'library' of routines. A 'Description of Routines in CSIRAC Library' can be obtained from the computation Laboratory. Since efficient library routines are available for input and output of numbers in various forms, for division, square root and so on, the programmer will need to know which routines are available and may regard them as an extension of the basic operations of the machine.

1.2 Preparation of a Programme.

This falls into a number of stages that are relatively distinct:-

1. Method selection. Compound interest may be calculated directly or by repeated simple interest calculations. Solution of an algebraic equation could involve Newton's method or trial and error. The questions involved here belong largely to mathematics rather than to computing techniques, but the choice of method may be influenced by the facilities peculiar to an automatic computer. Many repetitions of a simple process that can be readily programmed may be preferred, 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 where minor variations are possible. For example, in a money calculation one could use pounds and fractions or an integral number of pence, or again the number of significant figures required may vary. There are many decisions of this sort to be made, some larger and some smaller; the larger ones merging into questions of method.

3. Drafting the programme. This consists of breaking up the calculation into a series of steps, each of which can be performed by a library routine or by a simple list of commands. The draft is conveniently represented in a 'flow diagram' exemplified below:-

```

      1      STOP (to insert date tape)    <-+
+--> 2      READ NUMBER FROM TAPE        |
      |      IF POSITIVE                   IF NEGATIVE |
      |      FIND SQUARE ROOT             RETURN TO 1 -+
      |      PRINT SQUARE ROOT
+----- REPEAT FROM 2.

```

4. Coding the programme. The Master programme, which organizes interconnections between the library routines, is written in a symbolism introduced in section 1.5 and punched on paper tape using a keyboard marked in the command symbolism. It may be necessary to code and punch other routines, not provided by the library. These special routines, the library routines and the master programme are assembled onto one tape using copying equipment associated with the keyboard punch.

5. Testing and operating the programme. Generally the master programme and special routines are printed by the computer using Tape Symbol Punch library routine and subsequently checked. The programme is then read into the machine's store and a few check calculations made to ensure that the programme is as desired. When satisfactorily tested, the programme is operated automatically to complete the desired computation.

1.3 Representation of Numbers

Data are prepared for insertion into the computer by typing on the electric typewriter-punch, which produces a punched paper tape as well as the typed copy. When this tape is read into the computer the punch-codes are converted into the computer's code by standard library routines. Similar library output routines convert numbers in the computer into output codes, which may be punch codes for paper tape (to be printed later on the electric type-writer) or codes for the monitor printer attached to the computer. Since all conversions between the user's decimal representation and the computer's representation are handled by library routines and ancillary equipment, the user is rarely concerned with the details of number representation in the computer or on paper tape.

It is convenient, however, to discuss the basic operations of the computer in terms of representation of numbers as a pulse train in which the occurrence of a pulse is represented by '1' and a gap in the train by '0'. Numbers can be represented by a string of digits, which may be '0' or '1' in the 'binary' system, whereas the conventional 'decimal' system uses the digits 0,1,2,3,4,5,6,7,8,9. In the decimal system 5029.6 stands for

$$5 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 9 \times 10^0 + 6 \times 10^{-1}$$

and similarly in the binary system, 10110.01 represents the number

$$1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2}$$

which equals $16 + 0 + 4 + 2 + 0 + 0 + 1/4 = 22.25$ in the decimal system

The user should gradually become familiar with the binary symbols for the integers 0,1,2,...,30,31, which are listed in Appendix 1, but these can be worked out mentally by partitioning the integer into such of the components 16,8,4,2,1 as it contains. Thus $21 = 16 + 4 + 1 = 10101$ (binary). It is useful to understand the principles underlying addition and subtraction in the binary system. The addition of corresponding digits follows the table:-

0 + 0	gives	0	with	0	to carry
0 + 1	"	1	"	0	"
1 + 0	"	1	"	0	"
1 + 1	"	0	"	1	"

Thus : -

1010	1101	1100	11010
<u>+1100</u>	<u>+1101</u>	<u>-1010</u>	<u>-01101</u>
10110	11010	0010	01101

where in the case of subtraction the digit is borrowed rather than carried.

In CSIRAC the registers hold pulse trains of standard length corresponding to 20 binary digits, which are denoted by P20,P19,P18,... P2 ,P1. The decimal number 21 is represented by the twenty binary digit array 00000 00000 00000 10101. When this is subtracted from zero using the rules above:-

00000 00000 00000 00000
-00000 00000 00000 10101
 ..11)11111 11111 11111 01011

only the lower twenty digits are registered in the representation of $-21 = 1048555 = 2^{20}-21$. It may be verified that subtraction of this binary representation from zero produces the twenty binary digits of $+21 = 00000 00000 00000 10101$. For integers represented by 19 binary digits the top digit, P20, is 0 for positive numbers and 1 for negative numbers. The circuits of the computer's multiplication unit are based on this convention so that the product of the binary representation for -21 (with P20 = 1) with the binary representation of +21 (with P20 = 0) will be -441 in binary form (with P20 = 1). The convention that P20 is a sign indicator requires integers to lie in the range -524288 to +524287, i.e. -2^{19} to $2^{19} - 1$.

With decimal desk machines, the decimal point may be considered to lie anywhere in the number and similarly with CSIRAC the 'binary' point may be regarded as occurring between any two binary digits. The simplest conventions to maintain throughout a programme are (a) that the binary point lies to the right of P1, so that all numbers are integers as above, or (b) that it lies between P20 and P19, whereupon all numbers are fractions in the range $-1 \leq x < +1$. In the fraction convention products lie in the same range and no shifting is required to 'line up' the binary point as would be required by any other position of the binary point. Thus $+1/2 = 0.1000 00000 00000 00000$ and $-1/2 = 1.1000 00000 00000 00000$ sum to zero according to the rules for fractions and the multiplication circuits are designed so that their product will be $1.1100 00000 00000 00000$, which represents $-1/4 = -0.0100 00000 00000 00000$.

If it is desired to have the binary point between P19 and P18, the numbers of this example represent +1 and -1 and the product would represent -1/2, so that all products would have to be doubled to re-align the binary point. Because -1×-1 produces a result +1 which is outside the range of representation in the standard 'fraction' convention, the machine product $10000 00000 00000 00000$ will be regarded as -1, so that this exceptional case may have to be specially treated in a programme.

A convenient representation of a twenty binary digit numbers is one in which each group of five binary digits is replaced by the equivalent decimal integer, producing the '32 scale' representation. Thus the binary number 01001 10011 00110 11011 has the symbol 9,19,6,27 which, considered as an integer, equals $9 \times 32^3 + 19 \times 32^2 + 6 \times 32^1 + 27 \times 32^0$. In 32-scale representation the P20 digit is treated as positive, so that 10101 00000 00000 00001 has the symbol 21,0,0,1. To convert a decimal fraction to 32-scale, it is multiplied by 16 and the integral part of the result recorded. The fractional remainder is multiplied by 32 and the integral part of the product again recorded. Again

the fractional remainder is multiplied by 32 and so on. Consider the conversion of the decimal fraction 0.1 to scale of 32. We have:-

$$0.1 \times 32 = 3.2 = 3 + 0.2$$

$$0.2 \times 32 = 6.4 = 6 + 0.4$$

$$0.4 \times 32 = 12.8 = 12 + 0.8$$

$$0.8 \times 32 = 25.6 = 25 + 0.6$$

In this fashion 0.1 (decimal) = 3,6,12,25 (32-scale) is converted to 0.0001 10011 00110 01101 (binary). For negative numbers such as -0.1, subtract 3,6,12,25 from 32,0,0,0 using 32 for 'borrowing' to yield 30,12,25,19 = 1.1110 01100 11001 10011.

1.4 Organization of CSIRAC

CSIRAC is designed to have a main store capacity of 1024 'words', each 'word' being 20 binary digits stored in one of 1024 'cells' or store locations numbered serially 0 to 1023. During any calculation some of the words represent numbers but generally most words represent commands. Provision is made for storing larger quantities of data in auxiliary 'magnetic-drum' stores, each having a capacity for 1024 words.

Arithmetic operations are conducted using three main 'registers' A, B and C, and the D-registers denoted by D, D1, D2, ..., D15 and the half-word register, H. Commands take copies of numbers from store to these registers for calculation and return results to store cells when the registers are wanted for other operations. For control of operations the computer uses the 'sequence register', S to determine which command is to be performed next and copies the command into the 'interpreter register', K, which causes the appropriate units of the computer to be connected to perform the operation required by the command. An 'input register', I, is used in reading codes from paper-tape readers and an 'output register' is used when codes are to be printed or punched. Other facilities include hand-set switches, fixed constants, loudspeaker etc.

The normal procedure is to have commands of the programme stored in successively numbered cells of the main store in the order in which they are to be obeyed. The sequence register holds the number of the cell containing the command to be performed next and activates the circuitry which copies the command into the interpreter register, K. While the pulses in K activate circuitry to perform the current command, the number in S is increased by unity so that the next command will be taken from the cell following the cell holding the current command. Thus commands are normally taken in order from serially numbered store cells until the sequence is broken by a command of the programme which replaces the sequence by a new number. For example, if the sequence number is 13, commands will be taken from cell 13, then cell 14, 15, .. until the order from cell 27 (for example) causes the sequence number to be replaced by 139, so that after command 27 is completed the next commands will be taken from cells 139, 140, 141, etc., until another order is encountered, which replaces the 'sequence number' in S.

1.5 Representation of Commands.

Commands in CSIRAC call for a copy of binary digits from a 'source' to be transferred to a 'destination'. The ten least significant binary digits of a command are interpreted as specifying one of 32 sources and one of 32 destinations, which are listed on Appendices 2 and 3. The upper ten binary digits specify a number from 0 to 1023 which may be used as the 'address' of a store cell, Alternatively, the digits P14,P13,P12,P11 of a command may be used as the 'address' of one of the 16 D-registers. Other uses of address digits will be discussed subsequently. The digits of a command are partitioned as:-

P20	P19	P14	P13	P12	P11	P10	P6	P5	P1
---store-address-----						--source---			destination			
----D-address--												

When the 20 binary digits of a command are written in the 32-scale outlined above, the first two symbols represent the store address, the third represents the source code and the fourth represents the destination code. Commands are written with the address in 32-scale codes and the source and destination numbers replaced by mnemonic letter codes listed on pages(i-iv). For example, the binary digits:-

00010 10010 00000 00100

are written in 32-scale as 2,18,0,4 and this command takes a copy from source 0, which is the store (symbolized as M), to destination 4 (which is A), one of the arithmetic registers. It is seen that the store cell involved is number 82 ($2 \times 32 + 18$), thus the above digits represents a command which is written in the form:

2 18 M A

the effect of which is to take a copy of the contents of cell 2,18 (=82) of the main store to the A-register. It is important to remember that when this command is performed by CSIRAC, a copy of the binary digits in cell 2,18 will 'over-write' and replace the previous contents of the register A, leaving the number in cell 2,18 unchanged. Other commands are exemplified below:-

(a)	00010	10010	00100	00000	2 18	A M	2, 18, 4, 0
(b)	00000	00000	01110	00100		C A	0, 0, 14, 4
(c)	00000	00101	10001	10010	5	D PD	0, 5, 17, 18
(d)	01000	10101	00000	10001	8 21	M D	8, 21, 0, 17

Command (a) transfers a copy of digits from A to cell 2,18 replacing the cell's previous contents and leaving the contents of A unchanged. Command (b) copies the register contents of C into register A. Command (c) transfers a copy from D5 and adds it to the contents of D5, i.e., doubles the number by adding it to itself. In command(d) the address of the store cell is 'compatible' with D5 because the lower four digits of the address of the cell correspond to the address of the D-register. When the address digits are irrelevant as in command (b), by convention they are made zero.

It is often convenient to write commands with 'relative' addresses, denoted by the symbols '1A', '2A', '3A', etc. The latter indicate that in the course of reading the orders of the programme into the computer, various constants which will be determined by the computer itself or by the programmer will be added to the respective orders. This is used frequently in writing commands referring to a block of data to be placed in the main store cells, whose numbering: is to be decided later. The block is marked with the symbol '3S' (for example) and a command written in the form:-

3A 7 M C

will call for a copy of the contents of the seventh cell of the block to be transferred to register C. Similarly the command

3A 2 18 M A

will call for a copy from the eighty-second cell of the block to register A. The addresses are thus 'relative' to the head of the block, and when the command is read into the machine in a programme, the address will be made absolute by addition of the correct address of the block to the address in the command.

The examples of the next chapter will make the use of the command symbolism clearer. In explanatory comments the label of a register will be used to denote the digits or the number contained in the register and the contents of a register after an operation will be denoted by using a ' after the register symbol; e.g., comment on order (c) above could be "D5 = x, D5' = 2x"

CHAPTER 2. ARITHMETIC OPERATIONS

2.1 Most arithmetic operations occur either as a number emerges from a source or as it enters a destination. The combination of source and destination codes considered in this chapter perform the basic operations of addition, subtraction and multiplication.

2.2 Addition. A number may be added to the contents of registers A, C or any one of the D registers. Addition is not possible in store cells or in the B and H registers.

Two numbers held, for example, in cells 1,17 and 1,18 of the main store can be added using the A-register by the commands

```
0          1 17 M  A  Copy first number into A
1          1 18 M  PA Add second number into A
```

Since addition is not possible in storage cells one of the numbers is copied into a suitable arithmetic register and the sum is formed by adding the second number.

The sum in C is obtained by the commands

```
0          1 17 M  C  Copy into C
1          1 18 M  PC Add into C
```

The main arithmetic registers A,B,C and H do not involve address digits, whereas each D-register is specified by a four digit address. A number may be added from a main register to any D-register by a command such as

```
5  A  PD  Add the content of A to D5.
```

It is possible to copy or add numbers from a store cell to a D-register only if the lower four binary digits of the cell address are the same as the D-register address. If the cell address and D-register address are not 'compatible' in this way, an intermediary register such as B must be used. The following commands illustrate the direct and indirect use of D-registers, forming the sum in D1 and placing the result in 3,1 and 3,2.

```
0  1 17 M  D  Copy first number in D1
1  1 18 M  B  Copy second number into B
2          1  B PD  add it to D1
3          1  D  B  Copy sum into B and
4  3  2  B  M  hence to cell 3,2
5  3  1  D  M  Copy from D1 to cell 3,1
```

As 1,17 = 00001, 10001 and 3,1 = 00011, 00001 both are 'compatible' with D1, orders 0 and 5 do not involve B.

Addition of numbers from the store is performed in A or C whenever possible as less commands are required. D-registers are used only for 'compatible' cells or when A and C are already in use.

(Note: The serial numbering of the commands on the left is for the convenience of the programmer and does not necessarily denote the location of the command when stored in the machine.)

2.3 Subtraction. Subtraction is possible in A,C and D registers but not in B or H. The destinations SA, SC and SD (numbered 6,16,19 respectively) are used as shown below in examples corresponding to those of the last paragraph.

	<u>A-register</u>	<u>C-register</u>	<u>D-register</u>
0	1 17 M A	1 17 M C	1 17 M D
1	1 18 M SA	1 18 M SC	1 18 M B
2			1 B SD
3			1 D B
4			3 2 B M
5			3 1 D M

2.4 Clearing. When the number in a storage cell or register is transmitted to a destination, the number generally remains unchanged in the source. A, C and each D-register can be 'cleared to zero' by subtracting the number in the register from itself; for example:-

A SA C SC 5 D SD

The source CA (source code 9) may be used to clear the A-register, the command CA C copying the contents of A into C and clearing A to zero.

The source Z (source code 20) transmits zero and thus may be used to clear any register or store cell. Thus:-

2	0	Z	M	Replaces the number in 2,0 by zero
	5	Z	D	'Clears' D5
		Z	B	'Clears' B

Cells of the auxiliary stores MA (code 27), MB (code 28), etc., may be cleared using commands such as

3 5 Z MA 'Clears' cell 3,5 of MA

The entire main store, all the D-registers and each of the registers A,B,C,H may be cleared by pressing buttons on the switchboard of the console. The method of clearing the auxiliary store from the switchboard is outlined in a later chapter.

Registers cannot be assumed to be zero in a calculation. They must be cleared initially when subsequently used for addition and subtraction as below.

2.5 The complement of a number. The complement of a number when added to the number itself gives zero. For example, the binary complement of the number 22

00000,00000,00000,10110 in binary
is 11111,11111,11111,01010.

The complement of the number in D5 may be formed in (a) the C-register or (b) the B-register using the commands

(a)	0		C	SC	Clear C	
	1		5	D	SC	Subtract number in D5 from zero.

(b)	0		A	SA	Clear A
-----	---	--	---	----	---------

1	5	D	SA	Form complement in A
2		A	B	Copy into B.

The complement has to be formed in an intermediary register since B has no subtraction facilities.

2.6 Multiplication. The registers A, B and C are used in multiplication. Firstly the multiplicand is placed in C. The multiplier is transmitted to destination XB (destination code 12) forming the product in registers A and B, Thus the product of the numbers in cells 1,17 and 1,18 is obtained by the commands:-

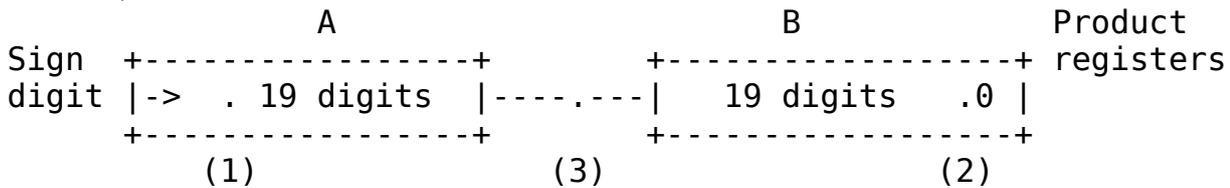
(a)	0		A	SA	Clear A initially
	1	1, 17	M	C	Set multiplicand
	2	1, 18	M	XB	Form product

OR

(b)	0	1, 17	M	C	Set multiplicand
	1	1, 18	M	A	Multiplier in A
	2		CA	XB	Clear A and multiply .

The multiplication unit forms the correct sign digit for the product from the sign digits of the two numbers multiplied. The remaining 19 binary digits of each number are multiplied to form a 38 binary digit product, The sign digit and the 19 most significant digits are added to the contents of A. Thus it is important to remember to clear A as in (a) command 0 or (b) command 2.

The least significant 19 binary digits of the product replace digits P20-P2 in B with the P1 digit zero as below,



- (1) Position of binary point in the product of two fractions
- (2) Position of binary point in the product of two integers
- (3) Position when an integer and a fraction are multiplied where representation is in the standard conventions (Sections 2,7, 2.8).

The number in C is unchanged after multiplication and can be used again if required, for instance x^5 , when the fraction x is in cell 1,17 can be formed by the orders

0	1, 17	M	C	x placed in C
1		C	A	x placed in A
2		CA	XB	Form x^2
3		CA	XB	x^3
4		CA	XB	x^4
5		CA	XB	x^5

The lower 19 digits in the B register are not used in forming the values of x^3 , x^4 and x^5 . The error produced in x^2 by neglecting the contents of B may be as great as 2^{-19} . This error is then multiplied by the fraction x so that it becomes less than the error due to neglecting B after the final product.

The error produced in a product when only the A-register is considered, may be reduced to 2^{-20} maximum error by rounding the product with the command

R PA

This adds 1 to the least significant position in A if the number in B is greater than or equal to 2^{-20} , (half a P1 digit in A), that is if the P20 digit of B is one.

2.7 Integer Products. When the product of two fractions with the binary point between P20 and P19 is formed, the result represents a fraction with the binary point between P20 and P19 of A (position 1 in the diagram in Section 2.6).

If two integers with the decimal point to the right of the P1 digit are multiplied the binary point of the product is between the P2 and P1 (position 2) digit of B. For positive products less than $524288=2^{20}$ the result is represented by the 19 digits of B but requires a right shift to return the binary point to the conventional position at the right of the P1 digit. The source RB (code 13) in a command such as

RB C

places the product in C with the binary point in the correct position, and with PS = 0. For negative products within register capacity the sign digit should be 1 but in right shifting to C the sign digit will be zero. This may be corrected by adding the sign digit from A, giving the correct result for both positive and negative products, by the commands

0	CA	XB	Form product
1	RB	C	Right shift to C
2	SA	PC	Add sign digit

provided the result is less than 524288 in absolute magnitude.

2.5 Multiplication of an integer by a fraction. If a fraction and an integer are represented according to the binary point conventions outlined above, the binary point of the product lies between the integral part in A and the fractional part in B (position 3 in the diagram of Section 2.6). This is of use in scale conversions as in the following example where a fraction of a pound in A is converted to shillings and pence in D3 and D4. It is assumed the cell 1, 0 holds the integer 20 and cell 1, 1 the integer 12.

0	1	0	M	C	C' = 20
1			CA	XB	A' = integer shillings
2			RB	C	C' = fraction of a shilling
3		3	CA	D	D3' = shillings
4	1	1	M	XB	A' = integer pence
5		4	CA	D	D4' = integer pence:
6		4	R	PD	

2.9 Halving. The source HA (code 6) is used to halve the contents of A. If A contains $\frac{1}{2} = 0.1000\dots$ in binary, the command

HA A

right shifts the contents of A one place so that $A = \frac{1}{4}$. If A contains a negative number, for example $-\frac{1}{2} - A = 1.1000\dots$ in the standard convention. The same command then places $1.11000\dots$ in A, the machine representation of $-\frac{1}{4}$.

The one command thus right shifts the contents of A and adds the original sign digit.

A positive number may be halved in B, C or a D-register by using the sources RB, RC, RD (codes 13,16,19) in an order such as

RC C

If C had contained $1.1000\dots$ the result in C is $0.1100\dots$ so that $-\frac{1}{2}$ becomes $+\frac{3}{4}$ rather than $-\frac{1}{2}$ because the sign digit of the result is zero.

The sign digit has to be adjusted if those registers are used for halving negative numbers. Commands for placing half the positive or negative number in C into the D-register are:-

0	RC	D	Right shift C to D, PS = 0
1	SC	PD	Add original sign digit

In shifting, the least significant digit of C is lost. Commands giving an answer which is rounded up according to the lost digit are

0	C	D
1	RC	SD
2	SC	PD

2.10 Doubling A single left shift can be used to multiply a number by two provided the result is within the capacity of a register. The fractions $\frac{3}{8} = 0.0110 \dots$ or $-\frac{3}{8} = 1.1010\dots$ are doubled correctly to $+\frac{3}{4} = 0.110 \dots$ and $-\frac{3}{4} = 1.0100 \dots$ as in both cases the original P20 digit equalled the P19 digit. If, however, these fractions are doubled again by a single left shift, the results $1.1000 \dots$ and $0.1000 \dots$ respectively, exceed register capacity and the significant sign digit is lost.

A fraction x in the range $-\frac{1}{2} \leq x < \frac{1}{2}$ may be doubled by a single left shift leaving the P1 digit zero.

A single left shift in any register may be obtained by adding it to itself by a command such as

C PC

The source TA (code 7) transmits double the number in A. In particular the order

TA PA

trebles the contents of A provided the answer is within register capacity.

2.11 Multiple Left Shifts. The digits in the registers A and B can be shifted to the left jointly by calling upon the destination L (code 13). This operation corresponds to multiplying the 40 digit number in A and B by a power of 2.

The two registers are joined during the shift so that the upper digits of B move into the lower positions of A. The upper digits of A enter the lower position of B as shown below.

Initially	A	abc.....t	B	abg.....t
After 1 left shift		bcd.....ta		bgd.....ta
After 2 left shifts		cd..tab		gd.....taa
After 3 left shifts		d.... ..tabg		d.....taab

so that the initial P20 digit of A is repeated in B and the last digit emerging from the top of A is lost.

Left shifting occurs when a non-zero P20 digit is sent to L. The source K (code 26) may be used to supply the required PS digit. This source K transmits the address in the command in which it appears as the upper half of a number. For instance, the command

16 0 K L

transmits the digit train 16, 0, 0, 0 to L and since the P20 digit is one, causes a left shift.

Up to seven left shifts can be called.. The number required is specified by the lower four binary address, digits of the command, For n left shifts, ($n \leq 7$) the address must be 'compatible' with $2n - 2$ so that

16	0	K	L	gives one left shift
16	2	K	L	gives two left shifts
16	4	K	L	gives three left shifts

16	12	K	L	gives seven left shifts

Again for a left shift, the source PS (code 31) which transmits the digit train 16, 0 0 0 may be used, An alternative command calling seven left shifts is

0 12 PS L

The use of the left shift operation is illustrated by the following commands which multiply a fraction smaller than 1/10 in A by 10 and set the fractional result in A

0	10	0	K	C	C' = 10/16 = address of command
1			CA	XB	Places 10/16 fraction in A and B
2	16	6	K	L	Multiplies by 16 in 4 left shifts.

Here the result will not exceed register capacity if the original number is less than 0.1. Furthermore four of the more significant digits from B move into A.

Provided B is cleared initially, a number may be placed in A alone and left shifted. To multiply a number in A by 32 the commands below are used,

		Z	B	'Clear' B
16	8	K	L	Left shift 5 times, multiplying by 32.

2.12 Use of the H-register. The H-register may be used to shift digits ten places left or right or to select upper or lower halves of words. The source or destination HU (code 22) transmits or accepts upper half digits P20, P19,... P11, and the source or destination HL (Code 21) transmits or accepts lower half digits P10, P9 ... P1. This is illustrated by the following commands, where cell 2,3 contains the number 3, 4, 6, 1 (32-scale).

0	2	3	M	HU	H' = 3,4
1			HL	A	A' = 0,0,3,4
2	2	3	M	HL	H' = 6,1
3			HU	PA	A' = 6,1,3,4

In combination with source K, the H-register can be used to assemble constants from address digits of commands. The integer 10, and the constant $1/\pi = 0.3183099 = 5,2,31,6$ (scale 32) are assembled from the address digits by the commands:-

0	0	10	K	HU	Giving H' = 0,10
1			HL	C	C' = 0,0,0,10
2	5	2	K	A	A' = 5,2,0,0
3	31	6	K	HU	H' = 31,6
4			HL	PA	A' = $1/\pi = 5,2,31,6$

2.13 Logical Operations. When binary digits are transmitted to destinations CA,DA,NA (codes 7,8,9) they are combined with the digits already in register A according to the rules:-

	AND	OR	XOR
Digits transmitted	1100	1100	1100
Digits held in A	1010	1010	1010
Result of operation	CA=1000	DA=1110	NA=0110

For CA (conjunction or collation) the result digit is 1 only if the corresponding digit in both original numbers is 1, for DA (disjunction) if either is 1 and for NA (negation) if the digits differ. CA is

used to collate or separate out groups of digits of a number. The 'D-address' digits P14,P13,P12,P11 of a number in A are isolated by the command 0, 15 K CA, since digits of A will be retained only where they coincide with 1's in the digit train 00000 01111 00000 00000. Further uses of these operations will be illustrated subsequently.

CHAPTER 3. PROGRAMMING TECHNIQUES

3.1 Introduction. The last chapter illustrated commands used to perform arithmetic operations Programming techniques used to control the sequence in which the commands are executed or to vary a command before it is obeyed are introduced in the following sections.

3.2 Sequence Control Commands. The sequence register S holds the address of the next command to be executed (cf. 1.4). As each command is obeyed, the address in the sequence register is increased by unity so that commands are normally selected from successively numbered cells of the main store

This sequence may be broken by commands which replace or add to the sequence number in S. To achieve this, a number is transmitted to one of the destinations S (code 23), PS (code 24) or CS (code 25).

The programmer may determine the sequence unconditionally as the programme is written or it may be determined by contents of the registers as the programme is performed as follows :

A command of form 1,20 K S replaces the contents of S by the address 1, 20 so that the next command is taken from that address.

A command of type 0, 6 K PS adds six to S, causing a total increase of seven since a unit is automatically added as the command is executed. For example, if S = 0, 3 before this command, it is increased to 0, 10 so that the next command is taken from cell 0, 10.

Conditional

When the destination CS receives bits in either the P1-P11 or P15-P20 digits, an extra unit is added to S. Thus the computer 'skips' the next command if the either range is non-zero but executes it if both ranges are zero. If bits in both ranges are received a further unit is added to S thereby skipping a second command.

Sources SA, SC and SD¹. (codes, 5, 15, 18) transmit the sign digit of A, C or D respectively. The source R (code 12) transmits a P1 digit having the same value as the P20 digit of B.

By using commands of the form

	SA	CS
	SC	CS
	SD	CS
or	R	CS

the computer executes or 'skips' the next command according as the P20 of the source register is 0 or 1, that is as the number in the register is positive or negative.

The source ZA (code 10) transmits 0 in the P1 position if all the digits of A are zero or 1 if any digit of A is one. The command ZA CS is thus a 'zero-test' because the next command is executed if A is zero, by-passed if A is non zero.

¹ Some source and destinations have the same alphabetic code. For instance, the source SA = sign of A (code 5) and the destination SA = subtract from A (code 6) are used but are rarely confused in practice as they occur in different columns in written and printed programmes.

The source LA (code 8) transmits the 'least digit of A'. If the number in A is an integer, the least digit will be 0 if it is even, 1 if it is odd. The command LA CS makes the computer execute or 'skip' the next command according as the integer in A is even or odd respectively.

The sources PS (code 31; P-Sign), PE (code 14; P-Eleven) and PL (code 25; P-Least); transmit one in the P20, P11 and P1 positions respectively. The commands

PS	CS
PE	CS and
PL	CS

cause the machine to skip the next command unconditionally. For example, if A contains a positive or negative number the following commands will place the modulus or 'positive' value of A into A

0	SA CS	Test if A is positive or negative
1	PE PS	If positive, skip the next order
2	TA SA	$A' = A - 2A = -A$
3	next command	

3.3 Programme Loops. The number in B may be added to the number in A nine times by nine repeated commands B PA. A 'loop', requiring less storage, uses sequence control operations to repeat the same command nine times, counting the repetitions. A 'count down' loop is illustrated by the following commands.

7,0	<u>0</u> <u>8</u> <u>K</u> <u>C</u>	Set C' = 8 = n-1 to count 9=n
7,1	B PA	Add B to A
7,2	PE SC	Subtract 1 from count
7,3	SC CS	Skip when the count is negative
7,4	7 1 K S	Repeat from 7,1 while count is positive
7,5	Next command	

One is subtracted from the count in each cycle. As long as the number in C is positive, the command 7,4 takes the machine back to repeat the loop. After nine repetitions, C becomes negative and the conditional skip command 7,3 causes the machine to exit from the loop to 7,5.

An equivalent 'count up' loop generally requires one more command, as below:-

7,0	<u>31</u>	<u>23</u>	K	C	Set C' = -9 = 32,0 - 0,9 = n		
1	+ ->		B	PA	Add B to A		
2			PE	PC	Add one to count		
3			SC	CS	Skip if count negative		
4			PE	PS	- +	Exit when C is positive	
5	+ -	<u>31</u>	<u>27</u>	K	PS		S' = 7,5+1+31,27 = 7,1
6		Next command				< - +	

Command 7,0 places a 'negative' address as count. To clear C and subtract 9 would require two commands instead of the one used. Since S has no subtraction facilities a 'negative' address is used in 7,5 to return to 7,1

Another method of counting is to use a 'strobe'. A pre-set digit is shifted progressively until it alters the effect of a conditional 'skip' command. For instance:-

7,0			PE	C	Set C'=P11=P(20-n) as a strobe digit			
7,1	+ ->		B	PA	Add B to A			
7,2			C	PC	Shift 'strobe' left each cycle			
7,3			SC	CS	- +	Skip when strobe is in position		
7,4	+ -	<u>3</u>	<u>1</u>	<u>28</u>	K	PS		S' = 7,4+1+31,28 = 7,1
7,5		Next command					< - +	

When the 'strobe' reaches the P20 position the 'loop' will have been repeated nine times. A strobe count is inconvenient for more than 20 repetitions and is rarely used unless the strobe digit serves some other purpose.

SUMMARY

- (1) For a COUNT DOWN loop, to be repeated n times, the initial count number set is n-1; a unit is subtracted in each cycle and exit occurs when the number becomes negative.
- (2) A COUNT UP loop usually requires one more command; the count number for n cycles is initially set as one is added each time and exit occurs when the number becomes zero.
- (3) A STROBE loop is less frequently used. For n cycles, the strobe digit initially set is P(20-n) if the strobe is left shifted each time. Exit occurs when the strobe reaches the P20 position.

CHECK each loop in a programme to avoid the most frequent mistakes, ensuring that:

- (1) The correct count is set for the required number of repetitions.
- (2) The loop is entered at the correct command.
- (3) The count is changed correctly in each cycle.
- (4) The exit test command is correct
- (5) The address of the command returning the machine to the start of the loop is correct.

Checking is aided by drawing lines to indicate the loop and exit as shown in the example. The 'negative' address in a sequence change command may be checked by counting from the 'target' command (7,1 in the examples above), adding one to the negative address until 31,31 is reached at the command changing the sequence.

The loops above are more than four times slower but occupy less store space than equivalent sequences of repeated commands.

3.4 Stop Commands. The computer is stopped by transmitting non-zero digits to destination T (code 31).

Commands to stop the computer are commonly required:-

- (1) After a programme tape has been read into the machine
- (2) To allow a data tape to be placed in the reader.
- (3) To allow switch settings to be changed.
- (4) When it is desired to inspect the contents of registers.
- (5) At the end of a calculation.

Stop commands such as

		PS	T,
		PE	T,
		PL	T,
0	1	K	T etc.,

can mark stages in programme since each has a distinctive display pattern on the computer lights.

A conditional stop command such as

SA	T
----	---

may, for instance, precede commands for calculating a square root and will stop the machine if A is negative. Conditional stops using the 20-digit hand-set switch sources NA and NB (codes 2 and 3) are useful when testing programmes. The stops are removed when testing is complete by setting the switches to zero.

If the computer is restarted from the console switchboard the programme continues from the command following the stop-command. In the event of invalid conditions arising in a calculation, the 'dynamic stop' 31 31 K PS, which repeats itself indefinitely, may be used to prevent accidental restarting.

The loud speaker destination P (code 10), used in the 'hoot' stop:-

31	31	K	P
31	30	K	PS

produces an audible sound to attract attention, provided the loud speaker switch is on.

3.5 Modification of commands entering the computer. If a command is preceded by a relative address, symbolised as 1A, 2A etc. (cf. sec. 1.5) a constant is added as it enters the computer. The constant can be the address of the cell in which a previous command was stored. In this case the earlier command is preceded by the 'designation' symbolised as 1S, 2S etc. A programme can thus be written so that commands are not completely determined until the programme is stored in the machine. In the following example, the first command is preceded by the designation '1S' so that, no matter where the group of commands is stored, the address of the first command will be added to each command preceded by '1A'. The resultant addresses will then refer to the correct cells in the computer store. In this example, the fraction x in register A is replaced by $A' = 0.2 + 0.3x + 0.4x^2$ and then the machine is stopped.

			1S	
0		CA	C	Set multiplier = x
1	1A	6	M XB	$A' = 0.4x$
2	1A	7	M PA	$A' = 0.3 + 0.4x$

3			CA	XB	$A' = 0.3x + 0.4x^2$
4	1A	8	M	PA	$A' = 0.2 + 0.3x + 0.4x^2$
5			PS	T	Stop with result in A
6	6	12	25	19	0.4)
7	5	10	21	11	0.3) in 32 scale
8	3	6	12	26	0:2) (rounded off)

This programme would be stored from cell 2,11 or 3,15 (for example) onwards in the form:-

2,11			CA	C	or	3,15			CA	C
2,12	2	17	M	XB		3,16	3	21	M	XB
2,13	2	15	M	PA		3,17	3	22	M	PA
2,14			CA	XB		3,18			CA	XB
2,15	2	19	M	PA		3,19	3	23	M	PA
2,16			PS	T		3,20			PS	T
2,17	6	12	25	19		3,21	6	12	25	19
2,18	5	10	21	11		3,22	5	10	21	11
2,19	3	6	12	26		3,23	3	6	12	26

3.6 Modification of a command before executing it The destination PK (code 26) is used to add a number to a command before it is performed. The command held in the store is not changed. A number transmitted to the destination PK is held and the next command is added to it. The sum is then executed as a command. For example, if $H' = 0,3$, the pair of commands

HU PK
D PA

will cause the second command to be executed as 0, 3 D PA.

This technique is often used to vary a command in a loop. The sum of the nine numbers held in cells 2,1; 2,2; 2,3; ... 2,9; can be formed by the 'count down' loop:-

0			A	SA	clear A	
1		8	K	C	Set count in C	
2	+ ->		C	PK	Vary next command	
3		2	1	M	PA	Adds from 2,9 the first time through
4			PE	SC	Reduce count	
5			SC	CS	-+ Test for end	
6	+ -	<u>31</u>	<u>27</u>	K	PS	
7		next command			<-+	

The command 2 adds 8 to the address of the next command so that it becomes 2,9 M PA the first time through the loop. In the second cycle $C' = 7$ so that the number in cell 2,8 is added to A and so on. It is often possible to use the count number to vary commands in this way.

If the first cell of the block holding the nine numbers had been designated with '4S' (for example), command 3 could have been written as 4A M PA; thus a command may be modified as it enters and also in the process of the calculation.

The source and destination codes can be varied by a preceding 'PK' command. For example, if $D = 0,0,0,1$, the second commands of the pair

D PK

C PA

will be executed as

C SA.

Only the address of a command referring to auxiliary stores MA, MB, MC and MD may be modified by a preceding PK command. **Commands calling on input or output equipment cannot be modified by a PK command.**

3.7 Modification of commands in the store. A command may be transferred to an arithmetic register, modified and replaced in the store to be executed later. This procedure may be used to sum the nine numbers of the last example.

7, 0			A SA	Clear A
7, 1		8	K D	Set 'compatible' count in D8
7, 2	+ -> 7	10	M C	Extract basic command
7, 3		8	D PC	Add count number
7, 4	7	5	C M	Plant constructed command
7, 5			PS T	Overwritten by varying command
7, 6		8	PE SD	Reduce count
7, 7		8	SD CS	->-+
7, 8	+ - -31	25	K PS	
7, 9			PE PS	->-+
7,10	2	1	M PA	Basic command
7,11	next	command		<-+

The command formed in C overwrites commands 7,5 by 2,9 M PA on the first traverse; then by 2,8 M PA the second time and so on.

This programme would be written as follows if relative addresses were used:-

			1S		
0			A	SA	
1		8	K	D	
2	1A	10	M	C	
3		8	D	PC	
4	1A	5	C	M	
5			PS	T	
6		8	PE	SD	
7		8	SD	CS	
8	1A	2	K	S	Relative address with sequence change
9			PE	PS	
10	4A		M	PA	Data block marked '4S'
11					next command

This method requires more commands than the use of 'PK', but is occasionally unavoidable. If a command changed in the store is required later in its original form, it must be reset before use.

5.8 "Switches", On each traverse of a loop it is possible to 'switch' between alternate sequences of command by using the contents of a register to determine which sequence to perform. The numbers of the last example may be alternately added or subtracted using a 'switch' in D12 as follows:-

			1S			
0			12	PS	D	Set switch initially
1				A	SA	
2			8	K	D	Set count
3	+->		12	PS	PD	Resets switch
4			12	SD	CS	
5			3	K	PS	->+
6			8	D	PK	
7		4A		M	SA	D12 negative, subtract
8			2	K	PS	
9			8	D	PK	<-+
10		4A		M	PA	D12 positive, add
11			8	PE	SD	
12			8	SD	CS	<-+
13	+-	1A	3	K	S	

As D12 alternates between 0 and P20, the sequence switches between commands 9,10 and 6,7,8, alternately adding and subtracting.

A three way switch is possible since

12 D CS

skips the next command if D12 holds P20 or P1 and skips two commands if D12 holds both P20 and P1. (With destination CS, the command is skipped if non-zero digits are transmitted in either of the groups P1-P11 or P15-P20; two commands are skipped for non zero digits in both groups; digits in the group P12-P14 have no effect).

3.9 Loops within loops. Each traverse of a loop may involve a repeated 'inner' loop. The sum of eleventh powers of the nine numbers of the last example is obtained in D by the following commands

			1S	
0			D SD	Clear D for later summing
1		8	K D	Set outer count
2	+->	8	D PK	Pick value
3	4A		M A	Data block
4			A C	C' = A' = x
5		9	K D	Set inner count
6	+->		CA XB	Form x ² ,x ³ ,x ⁴ ,x ⁵ etc.
7		9	PE SD	
8		9	SD CS	
9	+- 31 28		K PS	
10			CA PD	Add x ¹¹
11		8	PE SD	
12		8	SD CS	
13	+- -1A	2	K S	

In complex programmes care is needed to ensure that registers involved in outer loops are not affected by inner loops. Note that over 400 commands are executed in performing the above programme, which could itself be portion of an even larger loop.

3.10 Directory. A directory is a stored list of numbers used to modify operations on other numbers or commands. If the nine numbers involved in the previous few examples are scattered irregularly throughout the store, a 'directory' of addresses may be used to refer to them. If the nine addresses are stored in a block of cells designated by '5S', the sum of the numbers is formed as follows:-

0			A SA	
1		8	K C	
2	+->		C PK	Select a directory entry
3	5A		M PK	Combine with next command
4			M PA	Add from cell specified by directory
5			PE SC	reduce count
6			SC CS	
7	+- -	31 26	K PS	

The example in section 3.9 may be generalised to form the sum of arbitrary powers of the nine numbers by including arbitrary count numbers as the lower half of directory entries. If, for example, the third number in cell 23,17 is to be raised to the 7-th power, the third item in the directory is the

32-scale number 23,17,0,5; where the lower half is 2 less than the power required. A directory in this form may be utilised by the commands:-

0		D	SD	
1		8	K	D
2		8	D	PK
3	5A	M	HL	extract 'count' from directory
4		9	HU	D
				Set count in D9
5		8	D	PK
6	5A	M	HU	extract 'address' from directory
7		HU	PK)
8		M	A)
				extract number from store
9		A	C	
<hr/>				
10	+->	CA	XB	Raise number to power
11		9	PE	SD
				specified by initial count
12		9	SD	CS
				number plus two
13	+ -	31	28	K
				PS

and so on as in previous section.

A directory can concentrate the 'specifications' of similar operations, providing a powerful programming technique for convenient control of quite involved operations.

3.11 Standard Routines. Groups of commands for standard operations have been designed, tested and incorporated into a library of routines associated with the computer. Most of these routines are preceded by the designation 1S so that the relative address '1A' can be used in the routine to store it correctly in any desired group of cells. In addition, each routine is given a designation number as a programme is written and is also preceded by the designation nS, where n is the number of the routine.

The routines are constructed so the main programme may 'link' into the routine from a number of points in the calculation.

After the routine has been executed it is usual to resume the sequence of commands in the main programme. This is achieved by using the source S (code 23) to transmit the address of the next command to be performed to a D-register, where it is stored for the duration of the routine.

Generally D15 is used and two commands, known as 'plant and cue' are included in the main programme. For instance, a routine designated 7S is called in by the commands

n	15	S	D	'plant' address n+1 in D15
n+1	7A	K	S	'cue' to first command of 7S
n+2	next command			

12			6	PE	SD	Reduce D6 count
13			6	SD	CS	
14		+--1A	7	K	S	Repeat inner loop until D6 negative
15			5	PE	SD	Reduce D5 count
16			5	SD	CS	
17		+-----1A	3	K	S	Repeat outer loop until D5 negative
18			15	PE	PD	Adjust link address
19			15	D	S	Link to main programme

This type of routine is used for computing sums of products for statistical purposes with commands

	2	D	PK
3A		M	PA

preceding command 9 in the routine above.

THIS ROUTINE EVALUATES $A' = \sin pA$, FOR A IN THE RANGE -1 TO 0.999998, FROM THE FORMULA,

$$\sin pU/2 = 1 - .570795 U + 0.645921 U^3 - 0.079458 U^5 + 0.004362 U^7$$

IF THE ANGLE DENOTED BY A IS IN THE FIRST OR FOURTH QUADRANT, WE PUT $U = A$, IN THE RANGE .5 TO -.5. IF IT IS IN THE SECOND QUADRANT, $U = 1-A$, AN ANGLE IN THE FIRST QUADRANT WITH THE SAME SINE. IF IT IS IN THE THIRD QUADRANT $U = -1-A$, AN ANGLE IN THE FOURTH WITH SAME SINE. SO, IN EVERY CASE, U IS IN THE RANGE .5 TO -.5.

THE POLYNOMIAL IS EVALUATED AS FOLLOWS,

$$V = 2 U \times U$$

$$R = -0.034900 V + 0.317951$$

$$S = RV - 0.291842$$

$$T = SV + 0.570795 - V$$

$$\sin pU = 2(UT + U)$$

```

1S
0      8      K SA  ENTER FOR SINE
1      SA CS  ENTER FOR COSINE*
2      CA SA  ORDERS 0-3 GIVE A' = U, EQUIVALENT
3      8      K PA  ANGLE IN QUADRANT 1 OR 4
4      A D   D' = U
5      A C
6      CA XB
7      PS L
8      R PA
9      CA C   C' = 2 UxU = V
10 1A     27   M XB
11 1A     26   M PA  A' = R
12      CA XB
13 1A     25   M PA  A' = S
14      CA XB
15 1A     24   M PA
16      C SA  A' = T
17      D C   C' = U
18      CA XB A' = UT
19      C PA  A' = UT + U
20      PS L
21      R PA  A' = SIN pU
22      15   PE PD
23      15   D S  LINK
24      9   4   7 29 +0.570795
25     27 10 18 15 -0.291842
26      5   2   25 10 +0.317951

```

27 31 14 4 6 +0.034900
D PS T OPTIONAL STOP

* OMISSION OF ORDER 0 IS EQUIVALENT TO ADDING $\rho/2$ TO THE ANGLE.

=====

THE FOLLOWING ROUTINE FORMS $A' = A/C$; WHERE THE QUOTIENT A' IS IN THE RANGE -1 TO 0.999998 , THE LARGEST FRACTION THAT CAN BE REPRESENTED IN A SINGLE REGISTER.

THE METHOD IS BASED ON THE IDENTITY, FOR FRACTIONAL Z ,

$$1 = (1-Z)(1+Z)(1+Z^2)(1+Z^4)(1+Z^8) \dots$$

IF THE DIVISOR Y IS NEGATIVE, AS IT IS MADE IN THIS ROUTINE, WE HAVE, ON PUTTING $1+Y$ FOR Z ,

$$X/Y = -X(1+Z)(1+Z^2)(1+Z^4)(1+Z^8) \dots$$

PUTTING $R(N+1) = R(N) \times R(N)$
 AND $Q(N+1) = Q(N) + Q(N) \times R(N)$
 WITH $R(0) = Z, Q(0) = X$

THEN, WHEN N IS SUFFICIENTLY GREAT FOR $R(N)$ TO BE LESS THAN $P1$, NO FURTHER CHANGE WILL OCCUR IN THE REGISTER HOLDING $Q(N)$ AND THE QUOTIENT WILL BE

$$X/Y = -Q(N).$$

		1S	
0		C D	
1		D PD	D = TWICE DIVISOR
2		SC CS	IF C IS POSITIVE,
3		CA SA	CHANGE SIGN OF A
4		SC CS	
5		D SC	AND OF C
6		PS PC	SET C = 1+Y = Z = R(0)
7		A XB	C = R(N), A = Q(N)
8		R PA	A' = A+AC = Q(N+1)
9		CA D	D' = Q(N+1)
10		C XB	
11		R PA	A'=R(N) x R(N) = R(N+1)
12		ZA CS	IF R(N+1) = 0,
13		3 K PS	EXIT WITH D = Q(N+1) = -X/Y
14		A C	C' = R(N+1)
15		D A	A' = Q N+1)
16	1A	7 K S	
17		D SA	A' = X/Y
18		15 PE PD	
19		15 D S	LINK

CHAPTER 4. Input and Output

4.1 Introduction. In the following pages, a description of programming techniques is given for reading numbers into the machine, for punching and printing out results and for storing programme commands. The organization and input of complete programmes is then discussed.

4.2 Input of numbers Numbers are punched on 5-hole tape by means of the "Flexowriter". This tape is then read into the computer via the photo-electric 5-hole tape reader which is selected by a switch on the console. The source I transmits the digits from the input register to any of the destination registers A,B,C,H or D but not other D-registers or memory.

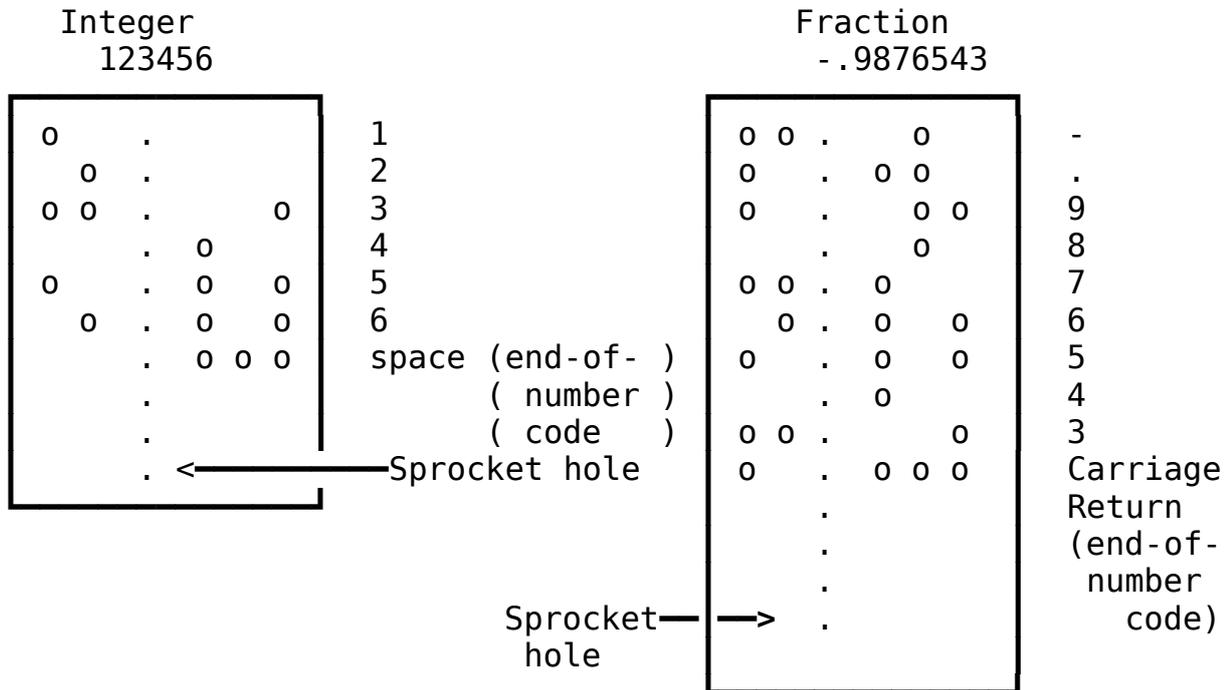
When the code in the input register has been transmitted, the tape is automatically advanced one row and the code is loaded into the input register in place of that transmitted. It should be noted that codes from 5-hole tape enter the input register in the P5-P1 digit positions.

Library routines have been designed for actuating the computer to read in the 5-hole tape and convert the codes for the decimal digits punched thereon to their binary equivalents in the machine.

Various routines are available and cater for positive integers or fractions punched with or without a "+" sign and for negative integers or fractions punched with a "-" sign. Fractions are preceded by a decimal "point", integers have no decimal "point". End of number codes can be space, tab, carriage return (which also gives a line feed to the "Flexowriter") or stop.

Punching errors are erased by depressing the Flexowriter key marked LF thereby punching 5 holes on a tape row. For erasure of a code, the tape row is moved back in the Flexowriter and the row overpunched since library routines ignore a row coded with 5 holes.

The following diagram shows 5-hole tape punched with a 6 digit positive integer and a 6 digit negative fraction:



The integer 123456 is assembled from tape in the form:-

$$((((((1 \times 10) + 2)10 + 3)10 + 4)10 + 5)10 + 6).$$

Each incoming code for a decimal digit, after conversion to its binary equivalent by the library routine, is added into the current product before a multiplication by the binary equivalent of 10 takes place. The routine assembles digits from tape until an end-of-number code is sensed. In this manner a 20 binary digit number is formed in the machine equivalent to the 6 decimal digit integer punched on tape. An example of a library input routine is given at the end of the chapter.

4.3 Number Output. The result of a computation is held in the machine in binary. Further routines are necessary to convert it to decimal form and either punch it on 5-hole tape for printing by the 'Flexowriter' or print it directly by the monitor teleprinter. Library routines punch or print numbers together with sign and decimal point where relevant, but the layout of the numbers on the printed page must be designed by the programmer.

The 5-hole punch is selected and switched on from the console. In addition it may be necessary to switch on the punch motor. A single row is punched on tape by a command calling on destination OP (code 3). The code punched is the logical sum of the group of digits P15, P14 ... P11 and P5, P4 ... P1 of the digit train transmitted to OP.

Tapes to be printed by the Flexowriter must be punched in the codes listed in Appendix 1. The output routine must translate the binary digit

representation of a decimal digit in the computer to the correct code. This is achieved by using a directory. 'End of number' and other layout codes are programmed by such commands as

```

    28 K OP  Punch code to call a 'space'
or   29 K OP  "      "      'line feed and carriage return'
```

The teleprinter operates at a quarter the speed of the punch but is used to produce progress results for 'monitoring' a programme, or to produce check results when programme testing. When in use, the teleprinter is switched on from the console switch board. The character printed corresponds to the logical sum of the groups of digits P15, P14... P11 and P5, P4 ... P1 as above, but the destination is OT (code 2) Note that the teleprinter coding (listed in \ sspendix 4); is different from that interpreted by the Flexowriter. For instance, a space is called by the command 31 K OT and the operation corresponding to the Flexowriter 'CR' requires two commands

```

    29 K OT  line feed and space once to the left
    30 K OT  carriage return without line feed
```

In using either form of output, the programmer must ensure the printing mechanism is set in the correct case. It may be set initially by hand or by including case shift commands in the programme.

```

Flexowriter  Teleprinter
27 K OP      27 K OT  Set lower case for figures, i.e.
```

Shift to figures = SF

```

30 K OP      28 K OT  Set upper case for letters, i.e.
```

Shift to letters = SL

Note that 'TAB' and 'STOP' operate on the Flexowriter only on figure shift.

Conversion from binary to decimal and output of numbers in decimal form is illustrated in the following routine which will print a positive fraction to six decimal places on the teleprinter.

```

1S
0          10 K OT      Print " + "
1          12 K OT      Print " . "
2          5   0 K D      Set count in D
3 1A          12 M C      Set l0P1 as multiplier
4           CA XB      Required digit in A, remainder in B
5           CA OT      Print digit
6           RB A       Reset fraction in A
7          1   0 K SD     Reduce count
8           SD CS      Exit after 6 cycles
9 1A          4 K S
10          15 PE PD
11          15 D S      Link
12          10         Constant multiplier
```

Binary codes for successive digits are formed as integers in A, with a fractional remainder in B. For an equivalent punch routine, command 5 could be replaced by commands to extract the corresponding Flexowriter code from a directory designated by '2S'

e.g.

	CA	HL	For digit n extract nth entry of
	HU	PK	directory of punch codes
2A	M	HL	Print via H at next command as
	HL	OP	punch not permissible after PK command.

Command 3 becomes 1A 15 M C because three extra commands are included. Commands 0, 1 will become 26 K OP and 13 K OP to punch the Flexowriter codes for 'plus' and 'point' respectively.

Input routines and output routines should have compatible conventions for arrangement of characters on tape so that output tapes may be fed into the computer if required.

4.4 Punching of programme Tapes. Commands are punched on '12-hole' tape. A diagram of the keyboard used is included at the beginning of the manual. The neon lights above the keyboard display the contents of the punch register.

In setting the punch register for a single tape row

1. the first key depressed sets a binary code in positions P10, P9 ... P6.
2. the second key depressed sets a binary code in positions P5, P4 ... P1.
3. The key bars X, Y and XY may be used to fill positions P19 and/or P20.

The neon display is then examined and if correct, the unmarked punch bar is pressed to punch the contents of the punch register as one tape row. If the neon display reveals an error, the bar labelled 'CLEAR' can be used to clear the punch register without punching.

In written punching direction, operation of the unmarked key bar is directed by *. The address of a command is punched first as a 32-scale number in one tape row. Zero addresses are omitted. The source and destination codes follow in the next row with an 'end of command signal' added by depressing the X-key. Punching directions for 2,18 M A are written as 2,18 * M, A, X *, while for CA XB the zero address is omitted in CA, XB, X *, appearing on tape as

<table border="0" style="width: 100%; text-align: center;"> <tr><td>o</td></tr> <tr><td>o</td></tr> <tr><td>o</td></tr> <tr><td>o o o o</td></tr> <tr><td>o o o</td></tr> <tr><td>o o o o o</td></tr> <tr><td>o</td></tr> </table>	o	o	o	o o o o	o o o	o o o o o	o	<p>2,18 M A CA XB</p>
o								
o								
o								
o o o o								
o o o								
o o o o o								
o								



where the tape is shown as it is placed in the reader.

Commands punched in this way are assembled from tape are stored serially by a 'PRIMARY' routine. In addition a 'CONTROL' routine is required to interpret 'control designations' (listed with punching directions in Appendix 4.) A 'Primary and Control' is generally copied at the beginning of programme tapes and is followed by other library routines and finally by the main programme. Tapes may be easily copied and corrected using the laboratory editing equipment.

The store cell in which the next command will be placed is specified by the 'transfer' number held in C by the primary routine. The number in C is increased by one as each command is stored so that storage is normally serial. When, however, a control designation is encountered, the control routine is called in by the primary.

The designation 'nS' causes the current value of the transfer number to be copied into the nth place of a control directory. When the designation 'nA' is read, the nth-directory entry is added to the address of the next command and the modified command is stored. The address of the storage cell holding a command preceded by 'nS' is thus added to each command preceded by 'nA'.

Note that punching of a control designation followed by a command may require 14 key operations. For example 3A, 1, 13 K S is punched as 0, 3 * 0, 2, Y * 1, 13 * K, S, X *, taking 4 tape rows.

4.5 Arrangement of Routines on Tape. Several factors determine the order in which routines are copied onto a programme tape. The first copied must be the primary routine and, if control designations are used in the programme, followed immediately by the control. Other routines are copied so that subordinate routines precede the superior routine calling on them; ensuring that the locations of the former are held in the control directory for use in the relative address commands of the superior routines. The main programme or master routine is punched last so that it may call on any preceding routine.

Routines are usually numbered in the order in which they occur on tape. The routine following the 'Primary and Control' is designated 2S, the next routine 3S and so on. For convenience, library routines are generally headed with 1S which is automatically copied along with the sub-routines. The master routine requires only the designation 1S for self reference, since it is not called in by any routine.

The combination 1A D K S, punched as
 0 , 1 * 0,2, Y * 0 , 6 , Y * K, S ,X *

is generally placed at the end of a programme tape. The designation D causes the assembled command 1A K S to be executed rather than stored, thus transferring control from the primary and control routine to the first command of the main programme. If, as is customary, this is a stop command, the machine will stop ready to execute the programme when restarted.

4.6 Data Blocks. An item from data stored in a block of cells is generally referred to using an address relative to the head of the block e.g. 3A 0,7. Experience has shown this provides greater flexibility in programming.

The address of the head cell of the block may be stored in the directory by using the control designation T. For example to store the first item of a data block labelled 3S in cell 8,16; the control combination 8, 16 T; 3S must be coded on tape. This is punched

8, 16 * Y * 0,3 * 0,1, Y *

and must appear on tape before the master routine or any other routine which refers to the data, It is convenient to insert this control combination after the end of the 2S sub-routine and before the next sub-routine, i.e. the 4S sub-routine.

The control combination 8,16 T changes the current transfer number to 8,16 which is stored in the third or 3S cell of the directory. Thus reference is made to the data using 3A.

The original transfer number is restored by preceding the control combination with 1S, punched 0, 1 * 0,1 Y * , and inserting immediately after the control combination 1A T, punched 0,1 * 0,2, Y * Y *. The current transfer number before the control combination is thus stored in 1S, the 1A extracts this number and T invokes it as the transfer number once again.

4.7 Example of Planning and Coding a Master Routine In this section the major steps in designing a programme are followed through for a simple example. The problem selected is to evaluate $\sin \pi x / \pi x$, x in the range (0,1) for 12 values of x, where x is originally given in degrees and minutes.

Planning the Calculation.

The routines which may be used are

INPUT of positive integers, outlined in section *				
Sine/cosine	"	"	"	3.13
Division	"	"	"	3.14
Output of positive fractions	"	"	"	4.3.

These routines use D15 as a link register. The input assembles numbers in register C. The remaining three operate on a number initially in A and place the result in A, or on tape in the last case. The division routine also requires a divisor in C. The sine/cosine routine is valid for

$-1 \leq x < 1$ and the division routine requires the divisor to be larger than the dividend. The steps of the calculation are then:-

- (1) Read in degrees and minutes punched as positive integers on 5-hole tape.
- (2) Convert to a fraction of π by multiplying degrees by $1/180$, minutes by $1/(180 \times 60)$ since 1 degree = 60 minutes.
- (3) Calculate $\sin \pi x$.
- (4) Multiply $\sin \pi x$ by $1/\pi$ so that it is less than x .
- (5) Divide $\sin \pi x/\pi$ by x .
- (6) Punch result on 5-hole tape, to be printed in a single column by the Flexowriter.

Flow Diagram.

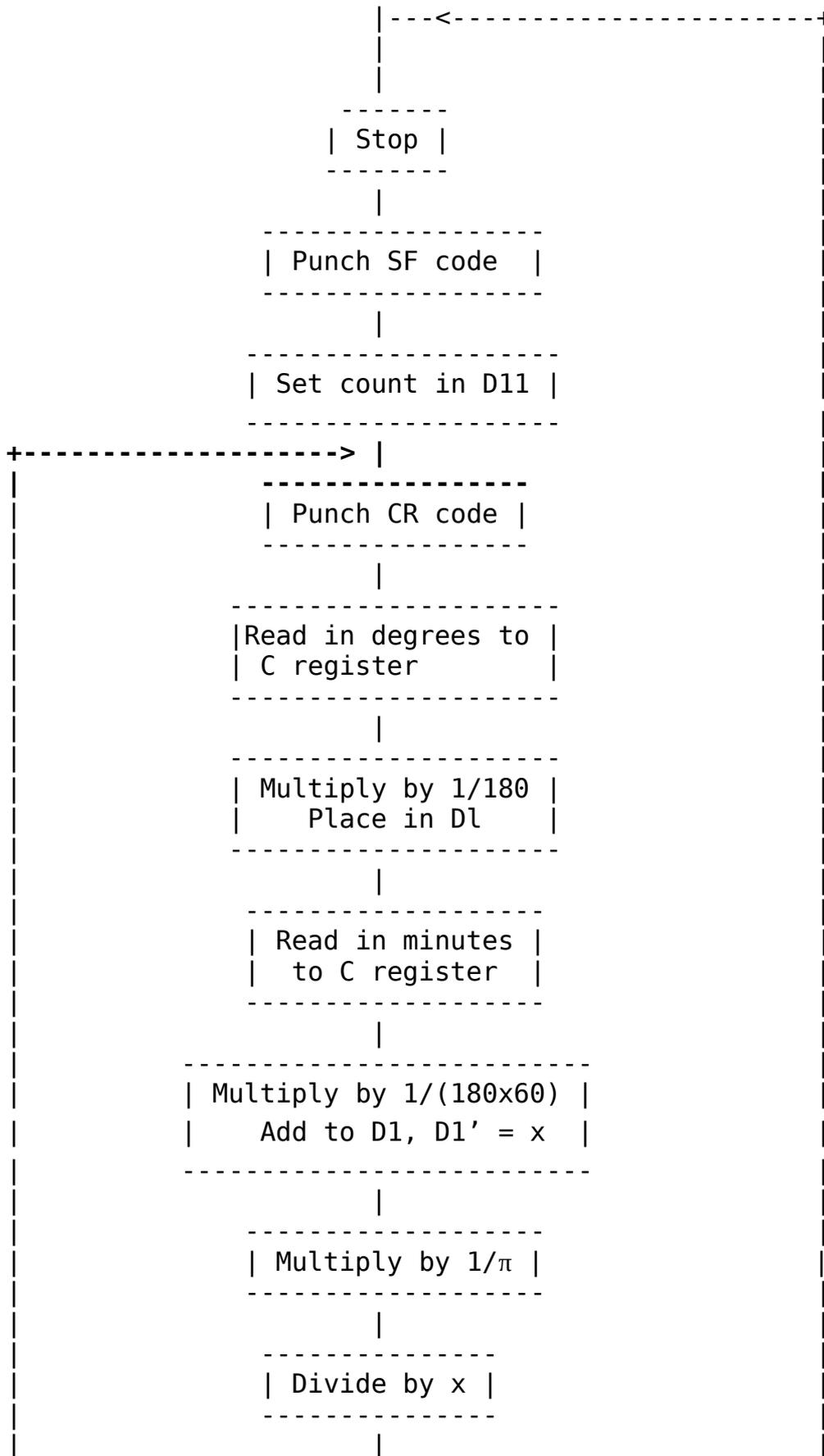
The next stage is to draw up a flow diagram as shown. The amount of detail included in such a diagram is an individual matter. In general, however, the more detail that is thought out at this stage, the fewer errors made in coding. When complete, the diagram should be studied to see if any shorter method suggests itself.

Coding.

As 12 repetitions are required a count of 11 is set in D11. The master assumes that the input routine is designated by 2S, the sine/cosine by 3S, the division by 4S and the punch routine by 5S.

* Generally an integer assembled from tape will appear in the A-register. Note, however, that for the example in section 4.7 the integer is formed in C.

FLOW DIAGRAM FOR THE EVALUATION OF SIN $\pi x/\pi x$



0	+----->		PS	T	Stop to insert data tape
1		27	K	OP	Set punch on figure case
2		11	K	D	Set count
3	+---->	29	K	OP	CR as end of number code
4		15	S	D	
5	2A		K	S	Read degrees into C
6	1A	27	M	XB	Mul by 1/180 giving a fraction in B
7		1	RB	D	D1' = degrees as a fraction of π
8		15	S	D	
9	2A		K	S	Read in minutes
10	1A	28	M	XB	Mul by 1/180x60, giving a fraction in B
11		1	RB	PD	D1' = x as a fraction of π
12		1	D	A	A' = x
13		15	S	D	
14	3A		K	S	A' = sin px
15	1A	29	M	C	C' = 1/p
16			CA	XB	A' = (sinpx)/p
17		1	D	C	C' = x
18		15	S	D	
19	4A		K	S	A' (sin px)/px
20	5A		K	S	Punch fraction on tape
21		11	PE	SD	Adjust count
22		11	SD	CS	test for 12 repetitions
23	A-1A	3	K	S	Return to read in next value
24		29	K	OP	CR punched on tape
25		17	K	OP	'STOP' code for Flexowriter
26	A- 1A		K	S	Stop the computer
27		2	27	1	1/180
28			1	17	1/180 x 60
29		5	2	31	6 1/ π

Note that the link number planted by command 18 will return the machine to command 20 after the division routine so it immediately enters the punch routine. Since the link number is again increased by one in this routine, the machine returns to 21 after completion of the punching. Thus only one 'plant' order is required preceding a sequence of cues, which call in routines using the same link register, to return the machine to the correct command in the master after completion of the routines.

The programme can be adapted to read in any number of values by using a zero test to exit from the loop instead of a count. This requires two commands

```

ZA CS
1A 24' K S (Address must be adjusted)

```

following command 12 (with appropriate adjustments to relative address constants 1/180 etc.). The angle 0° 0' following the data would then return

the machine to the head of the master. The zero test ensures that division by zero is never attempted and makes the program two commands shorter.

Routines may be used without knowing how they work. It must, however, be noted

- (a) which registers are used, particularly for data, results and links,
- (b) at which command the routine should be entered,
- (c) the designations associated with the routine, if any,
- (d) the store space required.

In this example, x cannot be stored in D as this is used by the routines and its contents would consequently be altered.

Routines are general purpose tools and can sometimes be adapted to save time and space in an individual programme.

For instance in the sine/cosine routine $(\sin px)/4x$ is obtained on completion of command 16, so that this routine could be shortened and the division routine omitted. For programmes which will occupy considerable computer time economy of this type warrants study of the library specifications and details of the routines involved.

4.8 Assembly of a Complete Programme, A 'tape layout' schedule is given as a guide in preparing a programme tape. For the programme of section 4.7 the 'tape layout' instructions are:-

		<u>Copy</u>	Primary and Control
Punch	2S	"	Integer Input
"	3S	"	Sine/cosine routine (section 3,13)
"	4S	"	Division routine (section 3.14)
"	5S	"	Fraction punch (section 4.3)
		"	Master routine including its IS designation
"	1A	DO	K S.

It is useful to make a list of tape and box numbers of the library routines used, Each tape is copied leaving some blank tape between each routine. The designation is punched from the keyboard before the routine is copied. *

The master routine should first be punched on a separate tape together with any specially designed routines. It should then be checked by obtaining a 'Symbol Print'. The latter and the flow diagram should be carefully searched for errors before being used in testing the performance of the programme.

4.9 Switchboard Instructions for the Insertion of a Programme Tape into the Computer.

- (1) Insert programme tape in the 12-hole tape reader, sprocket holes side towards hinge.
- (2) All console switches off, except those coloured red-yellow.
- (3) Restart computer until first tape row appears on input neons,
- (4) Clear all registers, the mercury store (if necessary) and the sequence register.
- (5) Single shot switch off. (Red-yellow switch nearest restart button).
- (6) Restart and stop computer when reader is resting on blank tape after primary.
- (7) NA & S to K switch off. (Red-yellow switch to right of single shot switch). Clear sequence register.
- (8) Restart to read in programme tape.

* When the Master Programme has been copied, some blank tape is left before the 'DO' command is punched, The complete tape must then be checked against the originals.

4.10 12-hole Tape Input and Output. The 12-hole reader and punch are selected by setting switches on console. When the reader is used the source I transmits digits to registers A, B, C, H or D (but not to other D-registers) and the next row of 12-hole tape is read into I. Channels X and Y correspond to positions P19 and P20, the remaining 10 channels are read into positions P10 to P1. When the 12-hole punch is used the digits transmitted to OP are punched as one row on tape. Again digits P20, P19 correspond to Y and X respectively, but the 10 channels correspond to digit positions P10, P9 ... P1 and P15, P14 ... P11, transmitted to the lower 5 channels. The 10 channels thus correspond to P10, P9, P8, P7, P6, disjuncting P15 and P5, P14 and P4, P13 and P3, P12 and P2, P11 and P1.

The 20 binary digits of a word may be punched as two rows on 12-hole tape for subsequent use by the computer. If an X-hole marks the row corresponding to the lower half of a word, the tape may be read back into the computer using the 'primary' routine. The tape may be punched with X-designations for lower halfwords by the commands:-

0		A	HU	
1		HU	SA	A' = lower half only
2		HL	CS	
3		PE	PS	Punch 'address' row if it is non zero
4		HL	OP	
5	8	K	PA	Add X = P19
6		CA	OP	Punch X-tagged lower half

Some library tapes call for data punched on 12-hole tape in 'compact' punched form. Decimal numbers are punched, using the editing keyboard, with two decimal digits per row. The end of a positive number is denoted by a Y hole and of a negative number by XY holes. Punching directions for the decimal fractions +.123456, -.056759 and -.032 respectively, are:-

1	2	*	0	5	*	and	0	3	*
3	4	*	6	7	*	2	0	*	

5 6 Y* 8 9 X Y* 0 0 X Y*

Other conventions are noted in the specifications of library routines.

4.11 FIVE-HOLE INPUT ROUTINE

=====

FUNCTION THIS ROUTINE CONVERTS AN INTEGER OR A SIX-DECIMAL FRACTION, PUNCHED ON FIVE-HOLE TAPE TO BINARY FORM IN A.

LENGTH 33

WORKING SPACE ABCHD

DIRECTORY FOR TAPE-CODE INTERPRETATION									
ORDER 9									
0 0	1		K D		BL	Z Z			
1	20	12	A SA						
2	1		I HL	1			H' = TAPE CODE		
3	2		HU PK	2			DECODED USING		
4	1A	1	M B	*		AVOID	DIRECTORY		
5	4		R CS	4			R = 0 FOR DIGIT		
6		17	K PS	(AVOID	TO 24		
7	22	23	B HU)		HU T	R = 1 FOR OTHER CODES		
8		7	HL PK	7			CHANGING ORDER 9		
9	8		M DA	8			AS LISTED		
10	31	23	K PS	#		PS T	RETURN TO 2		
11	31	21	K PS	=		PS MC	TO 1, IF NO DIGIT RECEIVED		
12	25	16	SD CS	-		PL PS			
13		6	K PS	&		AVOID	TO 20 IF INTEGER		
14	31	10	PS PA	.		PS PD	IF FRACTION,		
15	17	17	PS C	TAB		D CS	MULTIPLY BY		
16	1A	1	M PC	,		AVOID	C' = .524238		
17			CA XB	0			ALLOWING FOR		
18	17	17	C PA	STP		D CS	P20 OF A AS +1		
19	25	23	R PA	£		PL T			
20	3		S PK	3			IF MINUS SIGN,		
21	26	23	CA PA	'		K T	THIS BECOMES CA SA		
22	5	15	PE PD	5					
23	6	15	D S	6			LINK TO MASTER		
24	23	23	TA C	/		S T			
25	24	23	TA PC	x		PE T			
26	9		CA PC	9			C' = 5xPREVIOUS ENTRY		
27	16	8	K L	+		RC SC	NEW DIGIT IN A		
28	20		PL PD	SF		Z DA	RECORD 'DIGIT RECEIVED'		
29	17	17	C PA	SP		D CS			
30	17	17	C PA	CR		D CS	ADD 10xPREVIOUS ENTRY TO A		
31	31	2	K PS	SL		PS P	RETURN TO 2		
1 0	24	12	13 30	ER		PE Z	1+.524288		

NOTES

1. THE ROUTINE ACCEPTS NUMBERS PUNCHED AS SHOWN
-0.003456 +0.250000 .250000 .0 -.739634
2793 -45 0 +524287 -524288 -2793 -2793 -002793
2. FRACTIONS MUST HAVE 6 DIGITS AFTER DECIMAL POINT (EXCEPT .0)
3. END OF NUMBER CODES ARE TAB, SP, STP OR CR, WHICH ARE IGNORED UNLESS USED TO END A NUMBER.
4. ERASE-CODES ARE IGNORED ('LINE FEED' KEY USED TO OVERPUNCH ERRORS)
5. TO PUNCH A FRACTION NEAR +1, USE .999998 BUT NOT .999999, WHICH IS ASSEMBLED AS 'PS' = -1 .000000

Appendix 1

CSIRAC CODING

=====

CODE	BINARY	GATES		TELEPRINTER		FLEXOWRITER	
		(S)	(D)	LETTER	FIGURE	LETTER	FIGURE
0	00000	M	M	A	0		
1	00001	I	I	B	1	Q	1
2	00010	NA	OT	C	2	W	2
3	00011	NB	OP	D	3	C	*
4	00100	A	A	E	4	R	4
5	00101	SA	PA	F	5	K	(
6	00110	HA	SA	G	6	L)
7	00111	TA	CA	H	7	U	7
8	01000	LA	DA	I	8	I	8
9	01001	CA	NA	J	9	D	#
10	01010	ZA	P	K	+	V	=
11	01011	B	B	L	-	A	-
12	01100	R	XB	M	.	F	&
13	01101	RB	L	N)	M	.
14	01110	C	C	O	(G	TAB
15	01111	SC	PC	P	i	N	,
16	10000	RC	SC	Q	j	P	0
17	10001	D	D	R	k	J	STOP*
18	10010	SD	PD	S	▼	H	£
19	10011	RD	SD	T	φ	E	3
20	10100	Z	Z	U	Ψ	B	'
21	10101	HL	HL	V	Θ	T	5
22	10110	HU	HU	W	Ω	Y	6
23	10111	S	S	X	Γ	S	/
24	11000	PE	PS	Y	π	Z	x
25	11001	PS	CS	Z	Σ	O	9
26	11010	K	PK	Λ	E	Z	+
27	11011	MA	MA	SF*	SF*	SF	SF
28	11100	MB	MB	SL*	SL*	SP	SP
29	11101	MC	MC	LF*	LF*	CR	CR
30	11110	MD	MD	CR	CR	SL	SL
31	11111	PS	T	SP	SP	LF**	LF**

* Note that a space is also given with these calls.

** This call has no effect on Flexowriter printing, so may be used to erase when punching.

Appendix 2

CSIRAC COMMAND CODE: SOURCE FUNCTION GATES

CODE	SYMBOL	FUNCTION
0	n M	Transmit the contents of cell number n of the main store (20 digits)
1	I	Transmit the content of the input register (20 digits) and shift the input tape
2	NA	Transmit the contents of hand set register No. 1 (20 digits)
3	NB	Transmit the contents of hand set register No. 2 (20 digits)
4	A	Transmit the contents of the A-register (20 digits)
5	SA	Transmit the sign digit of A, i.e. the most significant digit of A.
6	HA	Transmit the contents of A divided by 2 (Half A).
7	TA	Transmit the contents of A multiplied by 2 (Twice A).
8	LA	Transmit the least significant digit of A.
9	CA	Transmit the contents of A, and clear A to zero.
10	ZA	If A = 0, transmit zero, otherwise transmit a PL digit.
11	B	Transmit the contents of the B-register (20 digits)
12	R	If the most significant digit of B is 1, transmit PL, otherwise transmit zero.
13	RB	Transmit the contents of the B-register shifted one place to the right, with zero as the most significant bit.
14	C	Transmit the contents of the C-register.
15	SC	Transmit the sign bit of C, i.e. the most significant digit of C.
16	RC	Transmit the contents of C shifted one place to the right, with zero in the sign digit position.
17	n D	Transmit the contents of the nth D-register (20 digits).
18	n SD	Transmit the sign bit of the nth D-register.
19	n RD	Transmit the contents of the nth D-register shifted one place to the right, with zero in the sign digit position.
20	Z	Transmit zero (20 digits).
21	HL	Transmit the contents in the position group P1-P10 of the H-register
22	HU	Transmit the contents in the position group P11-P20 of the H-register
23	S	Transmit the contents of the S-register (20 digits).
24	PE	Transmit 1 in the P11 position.
25	PL	Transmit 1 in the P1 position.
26	n K	Transmit from the interpreter-register (K) the number n as the most significant digits of a 20 digit number, with the least significant 10 digits zero.
27	n MA	Transmit the contents of cell No. n of the magnetic drum store No. 1.
28	n MB	Transmit the contents of cell No. n of the magnetic drum store No. 2.
29	n MC	Transmit the contents of cell No. n of the magnetic drum store No. 3.
30	n MD	Transmit the contents of cell No. n of the magnetic drum store No. 4.
31	PS	Transmit 1 in the P20 digit position.

Appendix 3

CSIRAC COMMAND CODE: DESTINATION FUNCTION GATES

CODE	SYMBOL	FUNCTION
0	n M	Replace the content of cell n of the main store by the digit entering.
1	Q	Has no effect
2	OT	Print on the teleprinter the character corresponding to digits 1 to 5 of the output register. (See Chapter 4 for details)
3	OP	Output to the five hole punch the digits in positions 1-5 of the output register. (See Chapter 4 for details) Output to the 12 hole punch specific digits of the output register.
4	A	Replace the contents of the A-register by the 20 entering digits.
5	PA	Add to the contents of A and hold the sum.
6	SA	Subtract from the contents of A and hold the difference.
7	CA	Replace the contents of A by the digit by digit logical product of its contents and the entering digits (i.e. conjunction).
8	DA	Replace the contents of A by the digit by digit logical sum of its contents and the entering digits (i.e. disjunction).
9	NA	Compare digit by digit the contents of A with the entering digits, placing 0 or 1 in the digit position as the digits compared are the same or different.
10	P	Transmit the entering bit stream to the loudspeaker.
11	B	Replace the content of the B-register by the entering 20 digits.
12	XB	Substitute the entering number into the B-register. Then form the product of the contents of B and C in A and B, the top 20 digits of the product being <u>added</u> to A and placing the lower 19 bits in B with a zero in the PL position.
13	L	Left shift the contents of A and B n places, $1 \leq n \leq 7$. (See Ch 2)
14	C	Replace the contents of the C-register by the 20 entering digits.
15	PC	Add to the contents of C and hold the sum.
16	SC	Subtract from the contents of C and hold the difference.
17	n D	Replace the contents of the nth D-register by the 20 entering digits
18	n PD	Add to the contents of the nth D-register and hold the sum.
19	n SD	Subtract from the contents of nth D-register and hold the difference.
20	Z	Has no effect.
21	HL	Replace the 10 digits of the H-register by the P1-P10 bits of the entering number.
22	HU	Replace the 10 digits of the H-register by the P11-P20 digits of the entering number
23	S	Replace the contents of the S-register by the 20 entering digit.
24	PS	Add to the contents of the S-register.
25	CS	Add one P11 digit to the S-register if either the P1-P11 or the P15-P20 group of the entering digits is not completely zero, and 2P11 if neither group is entirely zero.
26	PK	Replace the content of the K-register by the 20 digits entering. Add the digits forming the next command and obey the command represented by this sum.
27	n MA	Replace the 20 bits of cell No. n of the magnetic drum store No.1 by the entering digits.
28	n MB	As for 27 but using auxiliary store No. 2
29	n MB	As for 27 but using auxiliary store No. 3
30	n MB	As for 27 but using auxiliary store No. 4
31	T	If one or more digits received, computer; do not proceed to the next command

Appendix 4

PRIMARY AND CONTROL

T001 B100

=====

FUNCTION.

PRIMARY (STORE 0-13) STORES ORDERS IN SERIAL STORE CELLS. ORDERS ARE ASSEMBLED FROM 12-HOLE TAPE, WITH SOURCE-DESTINATION ROW X-PUNCHED (P19), AND ADDRESS, IF ANY, ON THE PREVIOUS TAPE ROW.
CONTROL (14-24) EXECUTES CONTROL-ORDERS PUNCHED 'Y' (P20)

OPERATING PROCEDURE.

1. 12-HOLE READER AND RED AND YELLOW SWITCHES ON, OTHERS OFF.
2. CLEAR ALL REGISTERS AND MEMORY.
3. SINGLE SHOT UNTIL ORDER D PD ON LIGHTS.
4. CLEAR SEQUENCE REGISTER.
5. 'SINGLE SHOT' OFF, STOP WHEN BLANK TAPE ON READER.
6. CLEAR SEQUENCE REGISTER AND SWITCH OFF 'NA&S TO K'
7. START. CONTROL AND PROGRAM STORED.
TAPE MUST HAVE N, T BEFORE PROGRAM.

PRIMARY

=====

0 0	D PD	
1	SD CS	
2	HU PA	ADDRESS TO A IF NO X-PUNCH
3	SD C	
4	S PS	TO 10
5	HL PA	IF X-PUNCH, ADD TO LOWER PART OF A
6	B PS	B=11 AFTER 'DO' ORDER, OTHERWISE B=0
7	C PK	
8	CA M	STORE ORDER
9	PE PC	ADD 1 TO C FOR NEXT STORAGE
10	I D	READ LINE OF TAPE
11	D HL	
12	SD CS	IF Y-.PUNCH, GO TO 14
13	Z S	
		CONTROL OVERWRITES THE FOLLOWING-
14	HU C	AFTER '14,Y' STORE NEXT ORDER IN 14
15	D SD	
16	Z HU	
17	Z S	

CONTROL

=====

0 14 Y ENTERED WITH M, N IN A

14	HU PS	A AT 18
15	24 M PA	T = 0Y M N K C
16	23 M PA	S = 1Y 0 N+24 C M
17	15 M PA	A = 2Y 0 N+24 M PA
18	19 CA M	ENTER FROM 6 AFTER 'DO' ORDER
19	PS T	REPLACED FROM 18
20	31 21 K PK	IF PERFORMED, CANCELS NEXT ORDER
21	11 K B	PUTS 11 IN B AFTER 6Y
22	10 K S	RETURN FOR NEXT ROW OF TAPE
23	13 27	CONSTANTS FOR FORMING CONTROL
24	31 8 12 14	ORDER FOR DESIGNATIONS T & S

CONTROL DESIGNATIONS

PUNCH CODE	SYMBOL	FUNCTION
m, n 0, 0Y	m, nT	<u>Transfer</u> control so that m,n is taken as the next transfer number*
0, n 0 1Y	nS	<u>Store</u> the current transfer number* as the nth parameter.
0 n 0 2Y	nA	<u>Add</u> the nth parameter to the address of the next command assembled from tape.
0 4Y	R	<u>Repeat</u> the control designation last executed
0 6Y	D	<u>Do</u> the command next assembled from tape instead of storing it.
0 3Y	-	Not used except with the Tape Symbol Punch. This designation causes the following orders to be punched as 32-scale numbers until a 0 7Y punch is sensed when alphabetic notation is registered.
0 7Y	-	0 7Y has no effect except with the Tape Symbol Punch.
0 5Y	-	Has no effect.

* The transfer number is the address of the storage cell in which the next command will be placed.