# FERRANTI MERCURY COMPUTER

# PROGRAMMERS' HANDBOOK

Ferranti

F E R R A N T I  L I M I T E D

# FERRANTI MERCURY COMPUTER

## PROGRAMMERS' HANDBOOK

Computer Department

Office and Works:
West Gorton,
Manchester 12.

London Computer Centre:
21 Portland Place,
London, W.1.

List CS 225A
April 1960

# FERRANTI MERCURY COMPUTER

# PROGRAMMERS' HANDBOOK

## PREFACE

This document is intended as a reference handbook for those who have learnt to use the Ferranti Mercury Computer in the standard way. This is to be distinguished from the use of the computer in the simplified method of the Autocode.

This handbook describes the machine from the user's point of view, and gives the techniques of what has come to be known as "conventional" programming whereby actual machine instructions are written. This permits the fullest utilisation of all the facilities of the computer.

It should be emphasised that this document is a handbook rather than an introduction to programming. Those seeking to learn about using the computer should first study the Ferranti publication List CS 158 "Ferranti Mercury Computer - Programming Manual".

Even the use of basic machine instructions, however, involves an input routine for converting them from the external form used in writing programmes to the internal form used in the machine. The standard Input Routine has been developed jointly by Ferranti Ltd. and Manchester University. It contains all the facilities of the Manchester University input routine "PIG 2" plus additional ones for diagnosing programme faults etc. This handbook describes how programmes are written to make full use of all the facilities of the standard Input Routine.


## ACKNOWLEDGEMENTS

The major part of this handbook has been prepared by Dr.R.B.Payne of the Computing Machine Laboratory of Manchester University. Some small additions and amendments have been made to bring it into line with the standard Input Routine.

Contributions to the input programme were made by Dr.M.de V.Roberts and Mr.C.E.Phelps (both late of Ferranti Limited).

The query print facility was inspired by methods of checking programmes used by Dr.A.Curtis, Dr.A.Hassitt and Dr.H.E.Wrigley of the Atomic Energy Research Establishment (Harwell). A similar technique is used at the Norwegian Defence Research Establishment.

The Quickies are based on v-routines supplied by Ferranti Ltd., Manchester University and the Norwegian Defence Research Establishment.

# C O N T E N T S

# CHAPTER 1

## THE FERRANTI MERCURY COMPUTER

### 1.1 General remarks on electronic computers

A high-speed digital computer is capable of carrying out a predetermined sequence of elementary arithmetical or logical operations. Mercury comprises five main parts :

an input unit whose object is to take in to the computer instructions and data which have been punched on paper tape;

a store to hold numbers read in and generated during the calculation;

an arithmetic unit in which elementary arithmetical and logical operations are carried out;

a control unit which selects the next operation to be performed;

an output device which gives the result of the calculation.

### 1.2 Scales of notation

With a human operator it is usual for arithmetic to be done in the scale of ten. The numbers he writes down consist of groups of the ten digits 0 - 9, e.g. 273 means $2 \times 10^2 + 7 \times 10 + 3$, and there are also other symbols like decimal points, spaces, etc. In the vast majority of electronic computers the scale of two or binary scale is used, numbers consisting of groups of the two digits 0 and 1 only. The reason for this is that it easily matches the on-off property of electronic circuits. With the Mercury Computer a pure binary representation of numbers is used, so that a binary number 11001

means $1\times2^4 + 1\times2^3 + 0\times2^2 + 0\times2 + 1$

or 25 in decimal notation.

### 1.3 Forms of storage

The information store in Mercury consists of two parts; to one, access is immediate, to the other there may be some delay before the required information is available. These two parts of the store are known respectively as the computing store and the backing store.

The computing store is made up of cores, which are small magnetisable rings of ferrite. Each core represents a single binary digit: it has two possible states of magnetisation corresponding to the binary values 0 and 1 of the digit. The capacity of the computing store is 2048 words each of 20 binary digits; the location of a word in the computing store is called a register. The computing store is divided into 32 equal groups of registers known as pages, so that each page holds 64 words of 20 binary digits. Each register is identified by a number, known as its address, by which references can be made to the register. It is convenient to think of the registers of the computing store as arranged sequentially in the numerical sequence of their addresses.

Each instruction occupies one 20-digit word, but words of two other lengths are also widely used in Mercury: 40-digit words and 10-digit words. The 20-digit registers are coupled in pairs so that each pair can hold a 40-digit word: such a pair is called a long register, and a 20-digit register called a medium register to preserve the distinction. A 20-digit register is also divided to form two short registers each holding one 10-digit word.

In the writing of a programme, the addresses used are all addresses of medium registers: a long register is referred to by the address of the medium register forming its first half, and a special device is used to distinguish between the two short registers forming one medium register. But when such a programme is read into the computer's store by the input routine, the input routine has to do some conversion of these addresses, because the computer itself works in three different address systems, one for each of the three possible lengths of register, and it requires the reference to a register to use the address in the system appropriate to the length of register. The address system for a particular length of register is obtained by considering the computing store as a set of registers of this length and numbering them successively from zero. Thus the first 40 digits of the store form the first 40-digit register, which has address 0 in the 40-digit address system; if the computing store is regarded as a set of 20-digit registers these same 40 digits form the first two 20-digit registers, with addresses 0 and 1 in the 20-digit address system; and in the same way they are also short registers 0, 1, 2 and 3 in the 10-digit address system. Similarly the short registers 24, 25, 26, 27, for example (in the computer's system of addresses) form medium registers 12, 13 (computer's addresses) and long register 6 (computer's address).

The larger backing store consists of rotating drums, the surfaces of which are magnetisable and store a total of 655,360 binary digits, or just over 16,000 long words. Unlike the computing store, direct access cannot be obtained to individual registers in the backing store and larger blocks of information must first be transferred to the computing store. The backing store is divided into 512 sectors numbered 0 to 511, each the same size as a page of the computing store, i.e. 64 medium registers. The transfer of information from the backing store is called reading from the drums and the reverse operation is writing to the drums. Physically, a sector is formed on circumferential tracks on the drum, the complete sector occupying just less than a semicircle. On each drum all even numbered sectors effectively begin at the same angle round the drum and all odd numbered sectors effectively begin 180 degrees further on. The sectors on each drum are divided into eight equal groups known as columns.

## 1.4 The control unit

This unit interprets the information contained in a 20-digit medium register of the computing store as an instruction in accordance with a set of rules known as the instruction code of the machine. Instructions are of the single-address type, usually refer to a register of the computing store and cause some operation to be done with the contents of that register. This operation may involve the arithmetic unit, input or output, a transfer to or from the backing store or the control unit itself. Normally the control unit selects instructions from consecutive registers, and when this happens the control address C or CA, i.e. the medium address in the computing store of the next instruction to be obeyed, is increased by one after each instruction has been performed. Certain instructions can, however, transfer control or cause a jump by replacing C by some address other than C + 1 from which the next instruction will be obeyed. Moreover, these transfers of control can be made conditional on numbers stored in the computer.

## 1.5 Instructions

The 20 binary digits of an instruction are divided into

7 function digits which specify the operation

3 B-digits (see below)

10 address digits.

The ten address digits permit machine addresses 0 to 1023. Thus instructions referring to long registers can refer to any part of the

computing store. Instructions referring to medium registers, e.g. transfers of control, can only refer to one of the 1024 medium registers in the first half of the computing store. Similarly, instructions themselves can only be obeyed from the first half of the computing store as C consists of ten digits. Instructions referring to short registers can refer to 1024 short registers in one quarter of the computing store only. With each instruction referring to a short register a second instruction referring to the 1024 short registers in the second quarter of the computing store is provided so that together the two instructions can refer to any short register in the first half of the computing store. Instructions referring to a short register in the first quarter have 0 for the first or most significant function digit and those referring to the second quarter have 1 for the most significant function digit but are otherwise identical. The only exception is the print instruction 63 (q.v.) where the second instruction of the pair is missing. Care should be taken if ever it is necessary to use this instruction.

## 1.6 The arithmetic unit

This part of the machine is concerned with the operations of addition, subtraction and multiplication and contains a special 40-digit register called the accumulator. Since with most arithmetical operations two numbers are involved, one of the numbers is first put into the accumulator and the result of the operation is also placed in the accumulator. For example, the addition instruction (42) adds the contents of a long register of the computing store to the contents of the accumulator leaving the sum in the accumulator; the contents of the computing store remain unchanged but the original contents of the accumulator are destroyed.

## 1.7 Representation of numbers

### 1.7.1 10 binary digit short integers (fixed-point).

(i) If the rth digit from the right is regarded as the coefficient of $2^{r-1}$ one can represent all the positive integers 0 to 1023. The number is then said to be unsigned or in the plus convention (+).

(ii) If the most significant digit is regarded as the coefficient of $-2^9$ and the rest of the digits as coefficients of $+2^{r-1}$ one can represent all the integers from -512 to +511. The number is then said to be signed or in the plus-minus convention (±).

| For example: | unsigned | signed |
|---|---|---|
| 0 000 000 111 | 7 | 7 |
| 0 111 111 111 | 511 | 511 |
| 1 000 000 000 | 512 | -512 |
| 1 111 111 111 | 1023 | -1 |

1.7.2 40-digit long numbers (floating-point). These represent $2^y \times x$ where the exponent, y, is ten digits and the fractional part x, is thirty digits. One digit of y is reserved to make possible the accumulator overflow positive test, so that $-256 < y < 255$. If an accumulator instruction other than an exact copy (40 and 41) gives an answer with $y > 256$ the machine stops automatically. Also, if it gives an answer with $y < -256$ the exponent is automatically increased to -256, x being unaltered. Otherwise the machine might wrongly interpret the product of two very small numbers as a large number.

The fractional part, x, represents a signed fraction, the first digit being the coefficient of $-2^0$ but otherwise the rth digit from the left is the coefficient of $+2^{1-r}$. Thus

$$-1 \leq x < 1 - 2^{-29}$$

A number is standardised if the two most significant digits of the fractional part are different

i.e. if $-1 \leq x < -\frac{1}{2}$

or $\frac{1}{2} \leq x < 1$

In this way the maximum number of significant digits are used and the number is represented uniquely. In the store the four short registers holding a floating-point number contain y, the least significant 10 digits of x, the middle 10 digits of x and the most significant ten digits of x in that order.

## 1.8 B-Registers

The machine is designed to take advantage of the repetitive nature of calculations. Thus when adding 100 numbers it is necessary to be able to tell the machine first how to do one addition and then how to repeat the operation on the remaining 99 numbers. B-registers are devices to facilitate operations of this kind, and in particular to select successive locations within the stores.

Seven B-registers are provided, B1 - B7, each of 10 digits, and the control unit is arranged so that the contents of one of these registers - that named by the B-digits of the instruction - is automatically added to the stored or presumptive machine address before the instruction is obeyed. For instructions referring to long or medium registers, the effective machine address is interpreted modulo 1024 so that the sign convention of the contents of the B-register is of no consequence. With instructions referring to short registers the signed contents of the B-register are added and the resulting machine address is interpreted modulo 2048, the machine instructions being considered to have presumptive machine addresses 1024 to 2047. With some instructions B-modification never occurs and if no B-register is specified in an instruction no B-modification occurs. The contents of B0 are always zero, or more correctly, there is no B0. It is possible to set the contents of, and count in B-registers.

## 1.9 Parity digit checking

With every 10-digit word stored in either the computing or the backing store, there is also stored an extra parity digit which is set so that the total number of ones is made even. This digit is not available to programmers but is automatically set when the word is copied into either store. When the word is later copied from the store the parity is checked and if it is wrong the computer stops. The parity stop can occur :-

(i) when a short or long word is copied from the computing store, e.g. for an addition;

(ii) when an instruction in the computing store is obeyed (the instruction is copied from the computing store to the control unit);

(iii) when a page of information is written into the backing store;

(iv) when a sector of information is read from the backing store; the check in this case is against the parity digit stored with each 10-digit word in the backing store.

# CHAPTER 2

## WRITTEN FORM OF INSTRUCTIONS

### 2.1 Introduction

Very few instructions are ever actually written in the binary machine form by programmers. Instead, a convenient shorthand has been devised for writing instructions, and the machine itself is furnished with instructions called the Input Routine for translating this into binary machine instructions. In this way some of the tedious effort of programming is avoided, and by allowing the machine itself to assist, programming mistakes are reduced. In the main, one written instruction corresponds to one machine instruction; there are a few exceptions such as Quickies (See Chapter 4) where one written instruction causes the Input Routine to insert perhaps a hundred machine instructions.

### 2.2 Layout of instructions

Just as in the binary machine form, written instructions consist of :-

function B-register address

The function is written as two decimal digits and the various operations that can be specified are described in the instruction code in chapter 3. The corresponding seven binary digits bear little systematic resemblance to the two decimal digits.

e.g. 99 0 000 000 stop

59 0 000 001 unconditional jump

The B-register is specified by a single decimal digit in the range 0 - 7. The address is always written in the medium register address system. A long register consists of a pair of 20-digit medium registers and for an instruction referring to a long register the written address is always the medium register address of the first of the pair. Thus long register addresses are written as 0, 2, 4, ..., 2046 and must always be even. For instructions referring to short registers, the written address is the medium register address containing the short register; a medium register consists of two short registers and the medium register address alone denotes the first short register or left half register, while followed by a terminal plus it denotes the second short register or right half register. Thus short addresses are written as

0, 0+, 1, 1+, ....., 1023+

During input the written medium register address is converted into a machine address by halving or doubling according to the type of instruction. With instructions referring to medium registers the machine address has the same value as the written medium register address. With short addresses, which are doubled during input, the terminal plus denotes $\frac{1}{2}$ e.g. 1+ becomes 3. Also, if the address of a written instruction gives a short register in the second quarter of the computing store, the second of the two machine instructions is selected by inserting a 1 in the most significant binary function digit position. Since instructions can only refer to short registers or medium registers in the first half of the computing store, short addresses must not exceed 1023+, and medium addresses must not exceed 1023. Also it is sometimes convenient to write negative addresses such as -2, e.g. with a B-modified instruction. As a long register address this is synonymous with 2046, and as a short or medium address -2 is the same as 1022,

the first half of the computing store then being cyclic. The limits on the written medium register address are :-

| | | | | |
|---|---|---|---|---|
| Long | -2046 | ≤ | L | ≤ 2046 |
| Medium | -1023 | ≤ | M | ≤ 1023 |
| Short | -1023+ | ≤ | H | ≤ 1023+ |

When B-modification occurs the effective written address is the presumptive written address plus

| | |
|---|---|
| 2B | for long addresses |
| B | for medium addresses |
| $\frac{1}{2}$B± | for short addresses |

(c.f. chapter 1) Addresses may also be written in page and line form e.g. 1.2+ is the medium register address of half register 2+ in page 1, i.e. half register 66+. During input the page number is multiplied by 64 and the line number added. No restrictions on the line number are imposed e.g. 1.234 is permitted, provided the resulting register address satisfies the above inequalities. A minus sign may come before the page number and refers to the whole fixed numerical address, but not to a symbolic address (see below) e.g. -1.2 is the same as -66.

## 2.3 Long numbers

It may be required to include some 40 binary digit long numbers amongst the instructions. These are distinguished by prefixing them with either + or - e.g.

+123.456

-98765

The long number goes into the next available long register, a dummy instruction, 570, being inserted if an (odd numbered) medium register is wasted. Similarly, if a short integer (see below) has just been inserted in a left half register a following instruction goes intc the next whole register leaving the right half register unaltered (zero). Long numbers may also be written in the floating decimal form of

| | fractional part | | comma | | (signed) exponent |
|---|---|---|---|---|---|
| e.g. | -123.456,25 | means | -123.456 | multiplied by | $10^{25}$ |

The accuracy with which written decimal numbers are converted into floating-pcint binary numbers is as follows:-
The fractional or fixed point part of the decimal number is read into the accumulator as an integer. The position of the decimal point (if any) and the exponent (if any) determine a power of ten by which the integer is multiplied. Thus

+3.000

will be converted less accurately than

+3

All integral numbers less than, and some integral numbers greater than ±500,000,000 are converted exactly e.g.

-123456789<br>+1,12 } Both are converted exactly.

For fractional numbers +0.5 is converted exactly but other numbers may contain errors, sometimes 3 or 4 times the least significant binary digit. For very large or very small numbers more accurate routines are available.

## 2.4 Short integers

It is sometimes desired to insert 10 binary digit short integers into half registers. These are distinguished from instructions by prefixing them with one of the symbols > = or ≠, e.g.

> 1
= -25
≠ 16.0

The commonest way of inserting short integers is with the symbol =, when the value of the 10 binary digits in the machine is the same as the written decimal integer. It is, however, occasionally useful to have the 10 binary digits in the machine equal to double or half the written decimal integer, and this can be achieved if the integer is begun with > or ≠ respectively. = 2, > 1 and ≠ 4 are all equivalent as are also

= -1, > -0+, ≠ -2, = 1023, > 511+ and ≠ 2046

Short integers are treated in the same way as the address part of instructions of type 2, 1 or 4 according as >, = or ≠ is used. (see chapter 3) This also applies to short integers written in terms of symbolic addresses (see below).

## 2.5 Introduction to symbolic addresses

It is not always convenient to decide which registers of the computing store instructions will occupy. If a transfer of control to another part of the programme is required, it is necessary to specify that address. This can be done by using a label and transferring control to the register containing the labelled instruction. Labels 1 to 99 are available and are written on the right of the instruction.

e.g. 400 32 (1

An unconditional jump to the register occupied by this instruction is written

590 v1

The symbolic address v1 is just a name for the address of the register occupied by the instruction labelled 1. During input the Input Routine converts the symbolic addresses into machine addresses.

## 2.6 Bracket ignore

If a left bracket ( occurs at the beginning of a line, all further characters are ignored until the next right bracket ). The tapes of Library Routines begin with the name of the Routine in brackets before the x-Routine directive (see Chapter 4). When a library tape is copied into a programme, the name in brackets may also be copied but this name will of course be ignored by the Input Routine.

The bracket ignore facility may be used in testing a programme. If on the first run it is desired to omit a block of instructions these can be included on the tape (see chapter 7) but between brackets. Then for the second run each of the two brackets may be converted into an erase character using a hand punch.

## 2.7 Even register

If an item is required to go in the next available even numbered register the two characters +) are inserted before it. What effectively happens is that on reading the + the Input Routine prepares for a long number by inserting a dummy instruction if necessary as long numbers always begin in an even register. The right bracket is then similar to the bracket ignore and need not necessarily be followed by CR LF.

# CHAPTER 3

## THE INSTRUCTION CODE

### 3.1 Abbreviations

| | |
|---|---|
| A | The accumulator or the contents of the accumulator |
| B | B-register |
| Bt | B-test register |
| C | The Control address, CA |
| D | The contents of a Sector of the drum backing store |
| E | The exponent of the accumulator |
| G | A spot on the Manchester University Graphical output |
| H | Short integer or half register of 10 binary digits |
| $I_i, I_o$ | Imperial Chemical Industries input/output character |
| L | Long number or long register of 40 binary digits |
| M | Medium register of 20 binary digits |
| ms. | Milliseconds. 1 ms. = 0.001 seconds |
| $\mu$s. | Microseconds. 1 $\mu$s. = 0.000,001 seconds |
| $M_i, M_o$ | Manchester University magnetic tape input/output character |
| n | The address part of the instruction regarded as short integer |
| P | Page of the computing store |
| S | Sac or B7 special |
| St | Sac-test register |
| T | Sector number |
| $t_i, t_o$ | Paper tape character; $t_i$ input, $t_o$ output |

In the notation for describing instructions, values of quantities after the instruction has been obeyed are distinguished by primes, whereas values before the instruction is started are unprimed. Quantities which do not occur primed are not changed by the instruction. It is usual to write the function digits and the B-digit together and to separate them from the address.

### 3.2 Accumulator instructions for addition and subtraction. Codes 40 -43

Since floating-point arithmetic is done in the accumulator, it is often necessary to copy a long number from the computing store into the accumulator. The instruction

400 32

copies L32 into A. The reverse operation is the 41 instruction which copies the contents of the accumulator into the computing store.

e.g. 410 34

copies the contents of A into L34. The instruction 42 adds a long number from the computing store to the accumulator e.g.

420 40

adds L40 to the accumulator, and the answer also appears in the accumulator after the operation. Similarly the 43 instruction subtracts a long number in the computing store from the accumulator.

Using several of these instructions other more complicated operations can be done e.g. Put x+y+z into L40 where x is in L32, y is in L34 and z is in L36.

| | | | |
|---|---|---|---|
| 400 | 32 | A′ = L32 | = x |
| 420 | 34 | A′ = A + L34 | = x+y |
| 420 | 36 | A′ = A + L36 | = x+y+z |
| 410 | 40 | L40′= A | = x+y+z |

The computer obeys the instructions sequentially and after the final instruction the sum is in L40. With the 42 and 43 instructions the result is rounded-off and standardised (see appendix 1). Occasionally the result is required without round off; the 44 instruction gives unrounded addition and the 45 instruction gives unrounded subtraction. When doing accumulator arithmetic with exact integers, unrounded instructions are required but the 30 binary digit fractional part of a floating-point number is usually a truncated approximation and unrounded instructions will give biassed results.

3.3 Accumulator instructions for multiplication. Codes 50 - 51

The multiplication instructions find the product of the accumulator and a long number from the computing store, putting the result in the accumulator rounded-off and standardised. The 50 instruction gives the product and the 51 instruction gives minus the product. e.g.

500 32

multiplies the contents of the accumulator by L32.

Examples:-

1. Put xyz into L31.0 (where xy and z are located as above)

| | | | |
|---|---|---|---|
| 400 | 32 | A′ = L32 | = x |
| 500 | 34 | A′ = A × L34 | = xy |
| 500 | 36 | A′ = A × L36 | = xyz |
| 410 | 31.0 | L31.0′ = A | = xyz |

2. Find $x^2 - y^2$

| | | | |
|---|---|---|---|
| 400 | 32 | A′ = L32 | = x |
| 500 | 32 | A′ = A × L32 | = $x^2$ |
| 410 | 40 | L40 | = $x^2$ |
| 400 | 34 | A′ = L34 | = y |
| 510 | 34 | A′ = A × -L34 | = $-y^2$ |
| 420 | 40 | A′ = A + L40 | = $x^2 - y^2$ |

3.4 B-instructions for addition and subtraction. Codes 00 - 03, 10, 12 and 13

In order to B-modify the accumulator instructions it is necessary to be able to put short integers into the B-registers and to add and subtract in B-registers. The instruction 00 copies a short integer

from the half register in the computing store into the B-register specified by the B-digits e.g.

007 32 B7′ = H32

The 01 instruction does the reverse operation of copying a B-register into the computing store. e.g.

017 32+ H32+′ = B7

H32+ is the right half register, H32 being the left half register. Instruction 02 adds (fixed-point) a short integer from the computing store into the specified B-register e.g.

024 1.0 B4′ = B4 + H1.0

and the 03 instruction subtracts a short integer in the computing store from the B-register e.g.

034 1023 B4′ = B4 - H1023

The instructions 10 12 and 13 where the decimal codes are ten more than those above do similar operations but in these instructions the address part of the instruction is itself regarded as the short integer with which the operation is performed e.g.

106 25 B6′ = 25

126 1 B6′ = B6 + 1 = 26

136 5 B6′ = B6 - 5 = 21

There is no instruction corresponding to the 01 instruction. Whenever an operation is done on a B-register the B-test register is set (See below).

## 3.5 Sac instructions. Codes 20 - 23, 30, 32 - 33

The above B-instructions are not B-modified, the B-digit specifying on which of the seven B-registers the operation is required. The Sac instructions, where the decimal codes are twenty more than the B-instructions, do the same operations except that the operation is always on B7. When used with these Sac instructions B7 is called Sac, an abbreviation for short accumulator. The B-digit of the instruction specifies the B-register by which B-modification is required. Whenever an operation is done on Sac (Codes 20 - 38) the Sac-test register is set (See below).

## 3.6 Jump instructions. Codes 59, 49, 08 - 09, 18, 28 - 29, 38

The instruction 59 is the unconditional jump or transfer of control, C′ = n e.g.

590 64 C′ = 64

Other jump instructions are conditional on the number contained in some location in the computer. The 49 instruction transfers control if the accumulator is positive or zero. If the accumulator is negative control is not transferred and C′ = C + 1 in the usual way, i.e. the instruction in the next register is obeyed. The 09 instruction causes control to jump if the B-test register is positive or zero. The 08 instruction causes control to jump if the B-test register is not zero and the 18 instruction combines testing and counting by causing control to jump if Bt $\neq$ 0 and also adding one to the B-register specified by the B-digit whether or not control jumps. The Sac testing instructions are twenty more than the corresponding B testing instruction. A Sac instruction sets St but not Bt .

It is now possible to programme operations which involve counting.

Examples:-

1. Add the number in L32 into the accumulator a hundred times

```
106   100              B6' = 100
420   32     (1        A' = A + L32
136   1                B6' = Bt' = B6 - 1
080   v1               Jump if Bt ≠ 0
```

In this example the same long number is added to the accumulator each time.

2. Add the hundred (different) numbers in the consecutive long registers from L16.2 onwards into the accumulator.

```
106   100              B6' = 100
426   16.0   (1        A' = A + L16.0 modified by B6
136   1                B6' = Bt' = B6 - 1
080   v1               Jump if Bt ≠ 0
```

Each time the addition instruction is obeyed the contents of B6 are one less and the consecutive long numbers are added, effectively twice the contents of B6 being added to 16.0. One could of course write out each of the addition instructions one after the other but the programme would then be unnecessarily long. Other ways of coding cycles of instructions are described in Chapter 8.

## 3.7 Backing store instructions. Codes 67 - 69

When reading or writing a sector of the backing store to or from a page of the computing store it is necessary to specify what are essentially two addresses, the sector number and the page number. Since each instruction is of the one-address type, two instructions are necessary, the first a 67 instruction to select the sector followed by a 68 or 69 instruction which initiates the reading or writing and in which the page number is specified. Once a sector is selected several reading or writing transfers can be obeyed with this sector which remains selected until the next 67 instruction is obeyed

## 3.8 Input/output instructions to the computing store, 60 - 63

The usual method of getting information into and out of the Computer is by means of punched paper tape. A stripe across the tape has five positions, in each of which a hole may be punched: this provides 32 tape characters, and a character is read into the Computer as five binary digits, a 1 for a position in which there is a hole and a 0 for a position in which there is no hole.

The 60 instruction reads a character from the input paper tape into the least significant five digits of a short register in the computing store, clearing the most significant five digits. The 62 instruction punches on the paper tape output the least significant five digits of the address part of that instruction. The 63 instruction punches the least significant five digits of a short register of the computing store.

The 61 instruction is not a paper tape instruction; it copies into a short register in the computing store the binary number read from a row of ten keys, or handswitches, on the control desk of the computer.

## 3.9 Multiple channel input/output to Sac. 90 - 97

The instructions of this group are input and output instructions which can select which of a number of input and output devices attached to the Computer is to be operated. The Computer has seven input channels and seven output channels, and to these are connected devices which handle information in units of up to 10 binary digits. For example, the 5-channel paper tape readers and punches handle information 5 bits at a time. It is usual to have tape readers attached to two of the input channels, though individual Mercury installations differ in the equipment they have.

Functions 90 and 92 - 97 are input functions and function 91 is the output function. They are all Sac functions: that is to say the input character is placed in Sac or the output character taken from Sac, unlike the 60 and 63 functions which refer to half-registers in the computing store.

The b-digit used with any of the functions 90 - 97 is interpreted in an unusual way, to specify the channel to be used. The seven input channels and the seven output channels are both numbered 1 - 7, and a channel is operated by a 9 group instruction with the appropriate b-digit.

If the b-digit in the instruction is zero, "channel 0" is interpreted as the exponent of the accumulator, so that the instructions in this case are not input or output instructions at all; for example, the instruction

901 0

takes the next character from channel 1 and places it in Sac, and

900 0

takes the (10-bit) exponent of the accumulator and places it in Sac.

The interpretation of the address digits of an instruction using one of channels 0 - 3 differs from that of an instruction using one of channels 4 - 7. In the former case, channels 0 - 3, the number given by the address digits is added to the input or output information, so that for example if Sac contains 6, say, the instruction

911 8

will send the Space character, with value 14, to output channel 1.

The output channels 4 - 7 are designed to take not just one device each, but in fact any number of devices up to 1024, and the n digits in the instruction are used in this case to determine which of the devices attached to the channel is to be used.

The 60 function always uses input channel 1, and the 62 and 63 functions always use output channel 1.

Note:- Some Mercury computers do not have functions 90 - 97.

## 3.10 Instruction types

Instructions are classified according to the following types:-

Type 1 e.g. 59 $C' = n$

The machine address has the same value as the written medium register address. For the written address $-1023 \leq M \leq 1023$. This type also includes instructions whose address is irrelevant e.g. 99 stop.

Type 2 e.g. 63 $T' = H$

A short address instruction where the written medium register address

is doubled but no most significant function digit is inserted. For fixed written addresses $-511 + \leq H \leq 511 +$ and floating addresses are modulo 512

Type 3 e.g. 10 $B' = n$

Fixed written addresses are treated as for type 1. Floating addresses are treated as types 1, 2 or 4 according to whether an instruction, short integer or long number is labelled (See chapter 5).

Type 4 e.g. 40 $A' = L$

Long address instructions where the written medium register address is halved during input. For fixed numerical addresses - $2046 \leq L \leq 2046$.

Type 5 e.g. 00 $B' = H$

Short line instructions where the written medium register address is doubled during input and the most significant function digit inserted appropriately. There are actually two machine functions for each decimal function but this need cause no confusion. B-modification for type 5 and also for the 63 instruction adds $\frac{1}{2}$ B+ to the written medium register address (See chapter 2).

## 3.11 The complete instruction code

The following instructions are standard on all full-sized Mercury computers. Some Mercury Computers have additional auxiliary equipment; instructions for these special devices are not included here.

### B-register instructions

00 $B' = Bt' = H$ Type 5 60 $\mu$s.

The contents of the short register of the computing store are copied to the B-register specified by the B-digit and the B-test register is also set. The store register is unaltered.

01 $H' = B$ Type 5 60 $\mu$s.

The contents of the B-register specified by the B-digit are copied to the short register of the computing store. The B-register itself is unaltered and the B-test register is not set.

02 $B' = Bt' = B + H$ Type 5 60 $\mu$s.

The contents of the short register are added into the B-register, the store register being unaltered. Since B-registers only hold ten binary digits, the result is the sum modulo 1024.

03 $B' = Bt' = B - H$ Type 5 60 $\mu$s.

The contents of the short register are subtracted from the B-register modulo 1024.

04 $B' = Bt' = B/2 - H$ Type 5 60 $\mu$s.

The unsigned contents of the B-register are first halved, the least significant digit being discarded and the most significant digit becoming zero, and then the contents of the short register are subtracted modulo 1024.

05 $B' = Bt' = B \,\&\, H$ Type 5 60 $\mu$s.

The logical operation "and" on each of the corresponding binary digits of the B-register and the short register of the computing store gives 1 if both digits are 1 and 0 otherwise.

$$0 \,\&\, 0 = 0$$

$$0 \,\&\, 1 = 0 = 1 \,\&\, 0$$

$$1 \,\&\, 1 = 1$$

which is digit by digit multiplication

e.g. if B = 10101 01010

and H = 00000 11111

then B&H = 00000 01010

06 $B' = Bt' = B \not\equiv H$ Type 5 60 $\mu$s.

The logical operation "non-equivalence" on each of the corresponding binary digits of B and H gives 1 if they are not equal and 0 if they are equal.

$$0 \not\equiv 0 = 0$$

$$0 \not\equiv 1 = 1 = 1 \not\equiv 0$$

$$1 \not\equiv 1 = 0$$

which is digit by digit addition (with no carry)

e.g. if B = 10101 01010

and H = 00000 11111

then $B \not\equiv H$ = 10101 10101

07 $Bt' = B - H$ Type 5 60 $\mu$s.

The B-test register is set as with the 03 instruction but the B-register is unaltered.

08 $Bt \not\equiv 0$, $C' = n$ Type 1 60 $\mu$s.

A conditional transfer of control where n is the ten-digit address part of the instruction.

09 $Bt \geq 0$ $C' = n$ Type 1 60 $\mu$s.

A conditional transfer of control.

10 - 17 These B-register instructions are the same as 00 - 07 except that the address part of the instruction is itself the integer n with which the operation is performed.

10 $B' = Bt' = n$ Type 3 60 $\mu$s.

12 $B' = Bt' = B + n$ Type 3 60 $\mu$s.

13 $B' = Bt' = B - n$ Type 3 60 $\mu$s.

14 $B' = Bt' = B/2 - n$ Type 3 60 $\mu$s.

15 $B' = Bt' = B \,\&\, n$ Type 3 60 $\mu$s.

16 $B' = Bt' = B \not\equiv n$ Type 3 60 $\mu$s.

17 $Bt' = B - n$ Type 3 60 $\mu$s.

18 $Bt \not\equiv 0$, $C' = n$; $B' = Bt' = B + 1$ Type 1 60 $\mu$s.

The content of the B-test register before the instruction is obeyed is tested and this is not necessarily the content of the B-register specified by the B-digit. 1 is added to the B-register specified by the B-digit. If Bt = 0 then C' = C + 1 in the usual way and B' = Bt' = B + 1 e.g.

186 v1 jumps to address v1 if Bt ≠ 0 and

adds 1 to B6

## Sac instructions

20 - 38 These instructions are the same as 00 - 18 except that the operations are referred to Sac and the Sac-test register rather than to a B-register and Bt. The B-digit specifies the B-register for B-modification of the address.

| | | | |
|---|---|---|---|
| 20 | S' = St' = H | Type 5 | 60 μs. |
| 21 | H' = S | Type 5 | 60 μs. |
| 22 | S' = St' = S + H | Type 5 | 60 μs. |
| 23 | S' = St' = S - H | Type 5 | 60 μs. |
| 24 | S' = St' = S/2 - H | Type 5 | 60 μs. |
| 25 | S' = St' = S & H | Type 5 | 60 μs. |
| 26 | S' = St' = S ≠ H | Type 5 | 60 μs. |
| 27 | St' = S - H | Type 5 | 60 μs. |
| 28 | St = 0, C' = n | Type 1 | 60 μs. |
| 29 | St ≥ 0, C' = n | Type 1 | 60 μs. |
| 30 | S' = St' = n | Type 3 | 60 μs. |
| 32 | S' = St' = S + n | Type 3 | 60 μs. |
| 33 | S' = St' = S - n | Type 3 | 60 μs. |
| 34 | S' = St' = S/2 - n | Type 3 | 60 μs. |
| 35 | S' = St' = S & n | Type 3 | 60 μs. |
| 36 | S' = St' = S/2 ≠ n | Type 3 | 60 μs. |
| 37 | St' = S - n | Type 3 | 60 μs. |
| 38 | St ≠ 0, C' = n; S' = St' = S + 1 | Type 1 | 60 μs. |

## Accumulator instructions

| | | | |
|---|---|---|---|
| 40 | A' = L | Type 4 | 120 μs. |

The contents of the 40 - digit long register of the computing store are copied to the accumulator. No standardisation, round-off or accumulator overflow positive test is performed.

| | | | |
|---|---|---|---|
| 41 | L' = A | Type 4 | 120 μs. |

The contents of the accumulator are copied to the long register of the computing store without standardisation or round-off. No overflow test is performed.

42 $A' = A + L$ Type 4 180 $\mu$s.

The contents of the long register of the computing store, as a 40-digit floating-point number, are added into the accumulator with standardisation and round-off. For details of the floating-point arithmetic see Appendix I.

43 $A' = A - L$ Type 4 180 $\mu$s.

The contents of the long register are subtracted from the accumulator with standardisation and round-off.

44 $A' = A + L$ unrounded Type 4 180 $\mu$s.

The contents of the long register are added to the accumulator with standardisation but without round-off.

45 $A' = A - L$ unrounded Type 4 180 $\mu$s.

The contents of the long register are subtracted from the accumulator with standardisation but without round-off.

46 $A' = A + L$ unrounded and unstandardised Type 4 180 $\mu$s.

The contents of the long register are added to the accumulator with neither standardisation nor round-off. The result is shifted down one place and the exponent adjusted.

47 $A' = A - L$ unrounded and unstandardised Type 4 180 $\mu$s.

The contents of the long register are subtracted from the accumulator with neither standardisation nor round-off. The result is shifted down one place and the exponent adjusted.

48 Shift $\leq 31$, $C' = n$ Type 1 60 $\mu$s.

A conditional transfer of control depending on the difference of the exponents of the accumulator and the long register of the computing store for the last addition or subtraction. If the difference of the exponents is greater than 31, $C' = C + 1$ in the usual way.

49 $A \geq 0$, $C' = n$ Type 1 60 $\mu$s.

A conditional transfer of control. If the accumulator is negative $C' = C + 1$ in the usual way.

50 $A' = AL$ Type 4 300 $\mu$s.

The contents of the accumulator are multiplied by the contents of the long register with standardisation and round-off.

51 $A' = -AL$ Type 4 300 $\mu$s.

Minus the contents of the accumulator are multiplied by the contents of the long register with standardisation and round-off.

52 $A' = AL$ unrounded Type 4 300 $\mu$s.

The contents of the accumulator are multiplied by the contents of the long line with standardisation but without round-off.

53 $A' = -AL$ unrounded Type 4 300 $\mu$s.

Minus the contents of the accumulator are multiplied by the contents of the long register with standardisation but without round-off.

54 A′ = AL least significant half Type 4 300 μs.

The complete product of the contents of the accumulator and the contents of the long register has a sixty digit fractional part; the most significant half is given by the 52 instruction. The result of this instruction is not standardised and not rounded-off and is always positive or zero (see Appendix I). The exponent is the sum of the two exponents minus 29.

55 A′ = -AL least significant half Type 4 300 μs.

The complete product of minus the contents of the accumulator and the contents of the long register has a sixty digit fractional part; the most significant half is given by the 53 instruction. The result of this instruction is not standardised and not rounded-off and is always positive or zero. The exponent is the sum of the two exponents minus 29.

57 Dummy Type 1 60 μs.

This instruction does nothing other than C′ = C + 1 in the usual way.

58 Hoot Type 1 60 μs.

An impulse is applied to the diaphragm of a loud-speaker. By doing this repeatedly a musical note can be produced to attract the attention of the operator, e.g. to insert a data tape.

59 C′ = n Type 1 60 μs.

The unconditional transfer of control.

Input/Output instructions

60 H′ = t Type 5 120 μs. (see 3.15)

An input instruction. The five binary digit character which is under the reading head is copied to the least significant five digits of the short register of the computing store, clearing the most significant five digits, and advancing the tape one character.

61 H′ = hs Type 5 60 μs.

The ten binary digits set on the handswitches are copied to the short register of the computing store.

62 t′ = n Type 1 120 μs. (see 3.15)

The least significant five binary digits of the address part of the instruction are punched as one character by the output.

63 t′ = H Type 2 120 μs. (see 3.15)

The least signficant five binary digits of the short register of the computing store are punched as one character by the output.

64 Display ′ = L Type 4 120 μs.

The contents of the long register of the computing store are displayed on the control desk monitors, the exponent as ten binary digits and the fractional part as thirty binary digits. The instruction must be repeated continuously to give a visible display.

Backing store instructions

67 T′ = n Type 1 60 μs.

Select sector n for a transfer of information to or from the backing

store.

68 $P' = D$ Type 1 (For the time, see 3.14)

Read the contents of the selected sector to page n where n is the integer forming the address part of the instruction.

69 $D' = P$ Type 1 (For the time, see 3.14)

Write the contents of page n to the selected sector of the drum.

## Exponent instructions

These exponent instructions are B-instructions which operate on the exponent of the accumulator. The address part of the instructions is added to the exponent (except for the 71 instruction).

70 $B' = Bt' = E + n$ Type 1 60 $\mu$s.

The exponent of the accumulator is copied to the B-register specified by the B-digit.

71 $E' = B$ Type 1 60 $\mu$s.

The contents of the B-register specified by the B-digit are copied to the exponent of the accumulator.

## The Destandardise Instruction

78 $E' = 0.2^n$ unrounded unstandardised Type 1 180 $\mu$s.

The effect of this instruction is to force the accumulator into an unstandardised form with a specified exponent: because of the one place right shift which accompanies all unstandardised operations this exponent is n + 1. The value of the contents of the accumulator is unaltered except for bits which may be lost off the bottom of the argument due to the destandardisation.

The effect of the 78 instruction is identical with that of adding to the accumulator with a 46 instruction a number with argument zero and exponent n. To work properly, then, the original exponent of the accumulator must be less than or equal to n.

## Multiple Channel Input/Output Instructions

90 - 97 Type 1

| | b digit = 0 | b digit = 1 to 3 | b digit = 4 to 7 |
|---|---|---|---|
| 90 | $S' = St' = E + n$ | $S' = St' = I + n$ | $S' = St' = I$ |
| 91 | $E' = S + n$ | $0 = S + n$ | $0 = S$ |
| | $(S' = St)$ | $(St' = St)$ | $(St' = St)$ |
| 92 | $S' = St' = S + (E+n)$ | $S' = St' = S + (I+n)$ | $S' = St' = S + I$ |
| 93 | $S' = St' = S - (E+n)$ | $S' = St' = S - (I+n)$ | $S' = St' = S - I$ |
| 94 | $S' = St' = S/2 - (E+n)$ | $S' = St' = S/2 - (I+n)$ | $S' = St' = S/2 - I$ |
| 95 | $S' = St' = S \,\&\, (E+n)$ | $S' = St' = S \,\&\, (I + n)$ | $S' = St' = S \,\&\, I$ |
| 96 | $S' = St' = S \neq (E+n)$ | $S' = St' = S \neq (I+n)$ | $S' = St' = S \neq I$ |
| 97 | $St' = S - (E+n)$ | $St' = S - (I+n)$ | $St' = S - I$ |

E represents the exponent of the accumulator, I information (of up to 10 binary digits) from an input channel, and O information (of up to 10 binary digits) sent to an output channel.

Instructions of this group with zero b-digit take 60 $\mu$s, and with non-zero b-digit 120 $\mu$s plus waiting time (see 3.15).

The 9 group with b-digit zero are similar to the 7 group except that they affect Sac and the Sac test register instead of a B-register and the B test register.

The Stop Instruction

99 Stop Type 1

This instruction stops the computer and lights the 'stop flip flop' light on the control console. If the prepulse button is pressed the computer will carry on from the next instruction.

## 3.12 Instructions for special equipment

The above instructions are standard on all full-sized Mercury computers. Some Mercury computers have additional auxiliary equipment and the instructions used are included here. Other Mercury computers have twice the standard drum backing store, viz have 1024 sectors, but no additional instructions are required for this.

### 3.12.1 Card input/output and line printer

Two buffer stores are provided, one to hold an exact copy of the card just read in and the other to hold an exact copy of the card about to be punched or line about to be printed. Standard cards containing 80 columns and 12 rows are used. (Additional buffer stores provide bit-by-bit checking during both input and output). The line printer prints one line of 100 characters. Each card input is read twice, each card punched is read at subsequent reading station for automatic checking and a self checking code is also employed.

80 Conditioning Type 1 60 $\mu$s.

The address part of the instruction specifies whether or not an exact binary copy of the contents of the computing store/buffer is required or whether disciplined code punching is required, and whether card or line printer as follows:-

For an exact binary copy, the 80 digits in the top row of the card correspond to the first 8 short registers of a page of the computing store, the next row to the next 8 short registers, etc. For disciplined code punching, the binary equivalents of the characters in each of the 80 columns are contained in the first 80 short registers of the page.

In the address of a conditioning instruction, the most significant four digits (i.e. the page position) concern the style of input, and the remaining digits (the line position) concern the style of output:-

| Page | Input |
|---|---|
| 0 | Disciplined code input |
| 1 | Binary input |

| Line | Output |
|---|---|
| 0 | Print line (disciplined) |
| 1 | Punch card in disciplined code |
| 5 | Punch card in binary code |

Modification works as for any type 1 function.

81 Read card Type 1 approximately $5\frac{3}{4}$ m sec (see 3.15)

This copies the contents of the input buffer to the page specified by the address part of the instruction and reads the next card to the buffer.

82 Punch card/print line Type 1 (for time see below)

The page, specified by the address part of the instruction, is copied to the buffer. The exact contents of the output buffer store are punched/printed (if a line is printed, the paper is advanced).

83 Paper throw Type 1

In addition to the normal paper step, paper is fed on (at a rate of about 10 inches per second), stopping at a preset position.

The speeds of the card input/output and line printer are:-

| | |
|---|---|
| Read card | 200 per minute |
| Punch card | 100 per minute |
| Print line | 100 per minute |

### 3.12.2 Magnetic Tape Backing Store

Magnetic tape decks, or mechanisms, are connected to Mercury through a special control unit; two of these control units can be attached to the computer and each control can operate up to four tape decks, giving a maximum of eight tape mechanisms in all.

The magnetic tape used is pre-addressed, and the possible operations are rewind, search (for a particular address) and transfers to and from the computing store. Each control can perform one search or rewind operation and one transfer simultaneously, and these can take place while the computer is obeying other programme so long as the computing store requirements do not conflict. If an attempt is made to use a control which is already occupied, the computer will be held up until that control is free; the new magnetic tape instruction will then be initiated and computation will proceed simultaneously. If an attempt is made to use a part of the computing store which is involved at the time in a magnetic tape transfer, the computer will stop: this is known as the interlock stop. Instructions are provided to test if the transfer part of each control is busy so that the interlock stop can be avoided by programme.

The magnetic tape operates at a speed of 60 inches/sec. and a reel of tape can be up to 3000 feet long. The tape has eight tracks, six for information, one address track and one clock track. Information is stored on the tape in blocks, addressed sequentially. At the end of each block a 6 digit check sum is automatically written, and when a block is read from the tape this check sum is checked; if the check fails the block is re-read. If the check fails repeatedly the computer indicates a tape failure and stops.

The principal method of storing information on magnetic tape is in blocks of 128 forty-bit words, so that one block of information from tape occupies 4 pages of the computing store. Such a block occupies 6.4 inches on the tape, including its address, and there is a gap of 1 inch between blocks.

A magnetic tape operation is performed by means of two instructions: the first selects which deck is concerned (and therefore which control) and the type of operation, and the second gives the required tape or computing store address (if any) and initiates the operation. The functions are:-

87 Select deck and operation Type 1 60 $\mu$sec.

In the address of this instruction, the most significant digit is ignored, the next three digits give the type of operation required, the next three digits are ignored and the least significant three specify the deck number.

The types of operation are:-

| | |
|---|---|
| 101 | Rewind |
| 001 | Search |
| 000 | Read from following block |
| 010 | Write to following block |
| 100 | Read from preceding block |
| 110 | Write to preceding block |

The arrangement of digits in the address is chosen so that the address can be written in page and line form, with the"page" corresponding to the operation and the "line" to the deck.

This instruction determines the effect of:

| | | | |
|---|---|---|---|
| 86 | Operate | Type 4 | 120 μsec (plus waiting time, see 3.15) |

The 86 instruction initiates the operation specified by the last 87 instruction obeyed.

In the case of a transfer to or from the computing store, the address specifies the register of the computing store from which the transfer is to start: this address must be the beginning of a page.

In the case of a search operation, the address in the 86 instruction is the address of a 40-bit word of the computing store; this word must contain the required block address in the least significant 20 bits of its argument. The remaining ten bits of the argument, and the ten exponent bits, are ignored.

In the case of a rewind operation, the address in the 86 instruction is ignored.

A block of information on tape is read or written in the forwards direction, so that after are block has been read the tape is positioned ready for reading the block immediately following. It is not necessary to give a "select" instruction for each "operate" if the same deck and operation are required each time, so that, for example, successive blocks can be read from one deck by means of an 86 instruction to select "read from following block" followed by several "operate"s. Blocks cannot be read in the reverse order simply by repeated "read from preceding block"s because when a block has been read, the "preceding block" is the block that has just been read.

The two instructions to avoid the computing store interlock are:-

| | | | | |
|---|---|---|---|---|
| 88 | TC 1 busy, | C′ = n | Type 1 | 60 μs. |
| 89 | TC 2 busy, | C′ = n | Type 1 | 60 μs. |

Each of these causes control to jump if the transfer part of the tape control concerned is busy. It is often necessary to test whether a particular magnetic tape operation has finished or not; the time elapsed since the "operate" instruction is not usually a sufficient guide because there may be an automatic repeat of the operation due to a checksum failure first time. These two functions are designed to avoid any reference to a part of the computing store involved in a tape transfer; any such reference will stop the computer. It is only in the computing store that any clashing must be avoided - if a tape control is referred to when it is already busy, the computer will be held up until the control is free to perform the second operation, and then the programme will proceed.

The method of storing information on magnetic tape in 128 word blocks is known as the 4-page mode; there is an alternative method in which the blocks on tape are much shorter, known as the Pegasus mode because this type of tape is compatible with the Ferranti Pegasus Computer and the Ferranti Magnetic Tape Converter. In this mode a block on magnetic tape is 112 6-bit characters in length and occupies about $3\frac{1}{4}$ inches on the tape, including the address, with a gap of 1 inch between blocks. In the computing store of Mercury this block occupies the first 112 half-registers of a page. On transfer to tape in this mode, the 6-bit characters are taken from the least significant 6-bits of successive half registers; on transfer from tape the characters are put into the least significant six bits of successive half registers, the top four bits being cleared. In both cases the remaining 16 half registers in the page are unaltered.

The instructions for using tape in this mode are the same as for the 4-page mode. The mode in which recording on tape is to be done is determined by a switch on the tape deck. Since the blocks in the two modes are of different lengths, the addressing in the two modes is different, so that to be usable a tape must be addressed in the right mode.

The times given above for the magnetic tape functions are the times for which they occupy the central computer, after which the tape operation

proceeds independently.

The times of the tape operations are:-

| | |
|---|---|
| Read/write next block: | 127 msec (four page mode)<br>53 msec (Pegasus mode) |
| Read/write preceding block: | 234 msec (four page mode)<br>100 msec (Pegasus mode) |

A search takes 127 msec for 4-page block scanned, or 53 msec for Pegasus-mode block scanned.

A rewind takes up to 5 minutes for a full 3000 ft. reel.

### 3.12.3 Manchester University Graphical Output

56 G' = L Type 4 120 μs.

The contents of the third and fourth short registers composing the long register, each modulo 256 are the co-ordinates of a point displayed on a special cathode-ray tube. This may be viewed by the computer operator or photographed by a camera permanently set up in front of a duplicate cathode-ray tube.

65 Open shutter Type 1 (for time see below)

For the camera on the Graphical Output, open the shutter.

66 Close shutter Type 1 (for time see below)

For the camera on the Graphical Output, close the shutter and advance the film one frame.

Although the machine time for these two instructions is 60 μs., the shutter actually takes approximately 100 ms. to open or close and consequently a delay must be programmed.

### 3.12.4 Manchester University magnetic tape input/output (provisional)

As an alternative to paper tape, there is available a static-reading magnetic tape system in which input and output operate at 1000 characters per second.

A single tape mechanism is used for both input and output. Information is recorded on magnetic tape in 5-bit characters at a density of 100 characters per inch; the tape has 8 tracks in all, 5 information tracks, 1 track for parity digits and 2 tracks for additional checking. On output, each character is automatically re-read for checking; if it has been wrongly written on the tape, the character is repeated in the next position with an indication in the check tracks that the previous character is wrong, and this second copy of the character is also re-read and checked.

The maximum length of tape on a spool is 1,800 feet; when running at full speed the equipment cannot stop on a single character, there being an overshoot equivalent to about five characters. The tape is primarily intended for fast recording of results of computation, the tape subsequently being transcribed to paper tape for printing; it can also be used as an additional backing store.

The equipment is designed to be connected to the multiple input/output channels and to be operated by the 9 group functions. It is conventional to attach it to input channel 3 and output channel 3, leaving channels 1 and 2 for paper tape equipment.

This magnetic tape equipment is being developed by New Electronic Products Limited to whom all enquiries should be made.

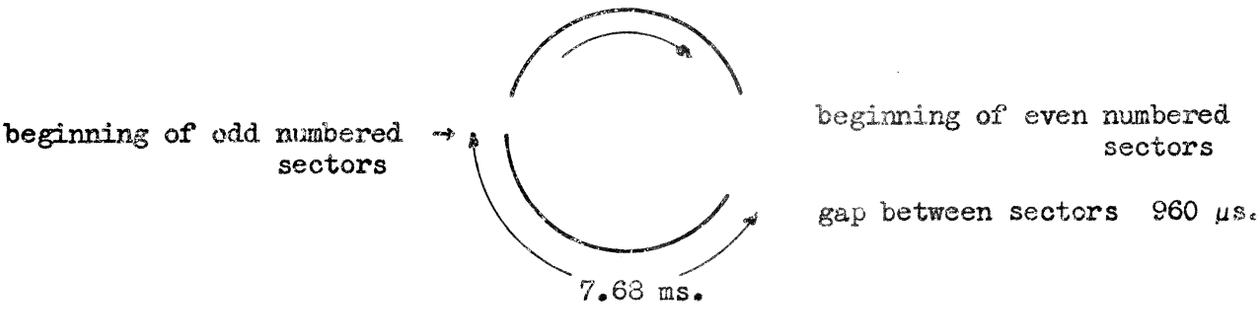
3.13 Unspecified instructions

The instructions described in the two lists above do not include all of the 128 possible different machine instructions. Of the remaining machine instructions some are duplicates of instructions described above e.g. 1 000 000 is also a prepulsable stop. Others perform useless operations or operations which can be done by other instructions, whilst others are dummy instructions. However the decimal codes which are not described above are all converted during input into the second prepulsable stop 1 000 000, the 99 instruction alone being 0 000 000.

3.14 Time of the drum transfer instructions. 68 P′ = D and 69 D′ = P.

The sectors are arranged on the drum two to a track; the drum is rotating continuously and the transfer to or from a particular sector takes place while the sector passes under a magnetic head near the surface of the drum, so that the computer has to wait until the beginning of the sector is in the right place before the transfer can start. The average waiting time for random access to the drum is half a revolution, i.e. about $8\frac{3}{4}$ msec, and the transfer itself takes about $7\frac{3}{4}$ msec. The physical arrangement of the sectors on the drum surface is effectively as shown in Figure 1.

Figure 1. Section of a drum, perpendicular to the axis of rotation.




For transferring several sectors it is consequently quickest to transfer odd and even numbered sectors alternately (which is done if consecutively numbered sectors are transferred). In the gap between sectors there is time for sixteen B-instructions.

3.15 Times of input/output instructions, etc.

In addition to the times given in sections 3.11, 3.12, certain functions may have a variable waiting time because they refer to equipment which is still completing a previous operation. This applies particularly to input and output instructions, where the equipment concerned is limited by its speed. For example, a 200 character/second tape reader can read a character every 5 msec, so that after one input instruction (itself taking 120 μs. only) the reader will not be ready to obey another input instruction for 5 msec. In most cases (and unless otherwise stated), if a second instruction is encountered too soon the computer is held up until the equipment is ready and the instruction is then obeyed.

This affects the timing of functions 60, 62, 63, the input and output functions of group 9, the card functions, the magnetic tape operate function, and the graphical output functions: these last form a special case in which the hold-up is not automatic but the delay must be programmed - see 3.12.3.

# C H A P T E R 4

## DIRECTIVES AND OTHER ALPHABETICAL INFORMATION

### 4.1 Introduction

As well as actual instructions, it is also necessary to direct the Input Routine in various ways. This is done by directives such as CHAPTER, ROUTINE and ENTER which are usually ordinary English words. To the input routine only the first letter is relevant, further letters of the word being ignored, so that spelling mistakes like ACCROSS are allowed. (For Autocode, however, correct spelling is essential.)

### 4.2 Chapter

Most programmes cannot be accommodated entirely in the computing store so are broken up into units called chapters, each of which can be accommodated in the computing store. The complete programme is stored in the backing store and the chapters are transferred to the computing store when required. The beginning of a new chapter is indicated to the Input Routine by say

CHAPTER 6

or by

C 6

since only the first letter is relevant. Chapter numbers can be 1 to 100 and each must have a different number. Unless there is a PAGE directive (see below) all chapters begin in page 1 and they must not extend beyond the end of page 15 unless special measures are taken to allow them to do so: see paragraph 4.13. Chapters are stored in the backing store in the order that they presented to the machine rather than by chapter numbers; each begins on a new sector, the next available after the end of the previous chapter, and occupies only as many consecutive sectors and pages as required or specified.

### 4.3 Routine

The chapters of a programme are themselves further divided into units called routines. The beginning of a routine is indicated by the word ROUTINE, or the letter R, followed by the routine number. All routines begin on the next available even register, a dummy instruction, 570, being inserted if necessary. Two kinds of routines exist, distinguished by whether the routine number is less than 1000 or not.

#### 4.3.1 v-Routines

These are numbered 1 - 999 and each v-routine in the complete programme must have a different routine number. For each v-routine a new set of labels 1 to 100 (see chapters 2 and 5) can be used and floating addresses referring to labels in the same v-routine are v1 v2 v3 etc.whilst floating addresses referring to labels in another v-routine are written as v2/3 say, meaning the address indicated by label 2 of routine 3. For each v-routine the label 0 is automatically set to the address of the beginning of the routine and this can be referred to by v0 or by v alone. e.g. to transfer control to the beginning of v-routine 901 the instruction can be

590 v/901

### 4.3.2 x-Routines

These are numbered from 1000 upwards and the number may contain a decimal point and fractional part (several decimal points are permitted) but not more than 77 digits in all. x-Routines are merely a device for inserting library routines but otherwise do not concern the programmer. The library tape is copied on to the programme tape by means of the tape copying or reperforating equipment (see chapter 7).

An x-routine does not terminate a v-routine, but rather it is part of a v-routine. Since it is not possible to label a routine directive and it is inconvenient to insert v labels (of the current v-routine) after instructions in an x-routine, it is usual to put the x-routine at the beginning of a v-routine, e.g.

```
R 234                    v-routine
R 1234                   x-routine
500  16x
 .
 .
 .
```

and then 590 v/234 transfers control to the beginning of the x-routine.

On reading the x-routine directive the Input Routine automatically sets the floating address x or x0 to the address in which the x-routine begins c.f. v0 for v-routines. Thereafter references to addresses in the x-routines are written in terms of the floating address x (see chapter 5). The actual x-routine number (a routine number greater than or equal to 1000) has no significance to the Input Routine , no permanent record is kept of x-routines, the address x0 being reset at the beginning of each x-routine. Thus two different x-routines in the one programme may have the same number (though this should rarely be necessary) and the same x-routine may occur several times in the one programme, e.g. in different chapters.

Hereafter the word routine will imply a v-routine.

## 4.4 Quicky

Many of the commonly used Library routines are stored in the computer within the Input Routine. If for instance the set of instructions which replace the contents of the accumulator by its reciprocal ($A' = 1/A$) is required, all the programme need contain is

```
Q 1
```

Quickies always begin in an even register, a dummy instruction being inserted if necessary. In appendix 4 a summary of details of the quickies includes the number of registers occupied and the working space used for storing intermediate numbers. A fuller description of the quickies is contained in the Ferranti publication CS 202. It will be seen that all quickies use at most

(i) The accumulator

(ii) Working space M 32-39

(iii) Sac St Bt.

None of the B-registers 1 to 6 are altered; some quickies use B5 and B6 during the quicky but the contents on entry are restored before the exit. The last instruction of the quicky always transfers control to the register after the end of the quicky.

Like an x-routine, a quicky is part of a v-routine and as such a quicky can be labelled, the address indicated being the first (even)

register occupied by the quicky e.g. if in a programme the instruction

Q 1 (2

occurs, then the instruction 590 v2 anywhere in the same v-routine transfers control to the beginning of the quicky.

4.4.1 Automatic Quicky Selection

A reference to a label number in the range 101 to 119 will cause a quicky to be inserted at the end of the chapter in which the reference occurs, and the reference will be interpreted as the address of the first instruction of the quicky. v101 corresponds to quicky 1, v102 to quicky 2 and so on. If several such references to one quicky are made in a chapter, only one copy of the quicky is inserted and all the references refer to this copy.

The quickies are inserted at the end of the chapter in the order in which the first references to them occur, each quicky beginning in an even numbered register. In addition to the instructions of the quicky, a 591 0 instruction is put in at the end of each quicky, so that it is stored in a "B1-closed" form. There is one exception to this: quicky 9 has a 591 3 instruction put in at the end - this is to conform with the non-standard entry required by this quicky (see Appendix III).

All library routines use the quickies in this "B1-closed" form and call for them using the automatic selection, so that if two library routines in one chapter require the same quicky, only one copy of the quicky will be included in the chapter and both library routines will use it.

4.5 Enter

At the end of a programme a directive is required to indicate to the Input Routine that it must stop compiling and begin obeying the programme. This is done by the directive ENTER followed by an address e.g.

E v/1

The chapter containing routine 1 (not necessarily chapter 1) is then transferred from the backing store to the computing store and control is transferred to the register indicated by the label 0 (the beginning) of routine 1.

4.6 Across

With a programme of more than one chapter, it is necessary to be able to transfer a second chapter from the backing store to the computing store when the instructions of the first chapter have been obeyed. This can be done by writing, say,

A v1/7

which when obeyed causes the chapter containing routine 7 to be transferred to the computing store and control transferred to the register indicated by label 1 of routine 7.

4.7 Down

On some occasions it is desired to call in a second chapter in such a way that, when this second chapter has been obeyed the original chapter is again transferred to the computing store, and control transferred back to the place where the chapter was left. The second chapter is called in by writing

D v1/7

It is convenient to think of chapters as being on different levels, the second chapter being a sub-chapter of the first. Sub-chapters can themselves call in sub-sub-chapters, etc. 9 levels being permitted. A chapter on a higher level is called a master chapter.

## 4.8 Up

When a sub-chapter has been completed and it is desired to return to the master chapter, the word

UP

is written. Sub-chapters can call in chapters on the same level by an ACROSS, the UP always causing a return to the level above at the place where that level was last left. UP is of course never followed by an address, whereas ACROSS and DOWN are always followed by an address.

Like quickies, ACROSS and DOWN always begin on an even register, a dummy instruction being inserted if necessary. For ACROSS and DOWN four registers of programme are inserted. For UP two registers of programme are inserted. ACROSS, DOWN, and UP may all be labelled; the address of the first instruction is indicated. For ACROSS and DOWN the label must follow the address.

An ACROSS or DOWN directive can be written without a v-reference in its address: for example ACROSS 2.0/15 will bring down the chapter containing R15 and enter it at 2.0 - or rather jump to 2.0 whether the new chapter, in fact, uses page 2 or not.

Pre-set parameters can also be written in ACROSS or DOWN directives. The routine number after the solidus only affects a v-reference if the reference occurs immediately before the solidus: for example, in

DOWN x1v1/10

the v1 has the value of label 1 of routine 10, but in

DOWN v1x1/10

the v1 has the value of label 1 of the current routine.

The ACROSS, DOWN and UP directives are called "cues".

## 4.9 Title

All programmes should begin by causing the machine to print out the title of the programme. The title is printed as it is read in by the Input Routine and whenever the programme is restarted using key 8 (See chapter 7). The title must come before the first CHAPTER directive on the tape. After the first figure shift character following the letter T the title comprises all characters (erases being ignored) until two consecutive figure shift characters occur, i.e. until a length of blank tape. A maximum of 244 characters are permitted in one title but several titles may be included in one programme (before the first C), each of which is punched during input, but only the last is stored in the backing store.

```
T
BILL SMITH
REACTOR 777/1/2 WITH NO COOLING
(FS FS)
```

It is useful to include a "carriage return" and "line feed" character in the title before the first word or figure.

4.10 Firstsector

Unless there is a FIRSTSECTOR directive, which must come before the first CHAPTER directive (but either before or after the title), the firstsector is taken as 128. On this sector are stored the title and the entry cue only. The first chapter begins on the next sector (129 if there is no F directive). This directive is useful if a programme requires a large number of consecutive sectors of the backing store, including drum 0, for working space. e.g.

F 440

C 1

will cause the programme to be stored on sector 440 onwards, the first chapter beginning on sector 441. It may also be desired to keep two separate programmes in the backing store simultaneously when they must obviously begin on different sectors.

The title of a programme is stored in its "first sector" from line 0 onwards; in line 61 is stored a count of the number of character pairs in the title. Information about the last ENTER directive obeyed - called the "entry cue" - is stored in registers 62, 63 of the "first sector".

4.11 Page

It may be desired to begin a chapter in some page other than page 1. (Chapters begin in page 1 if there is no PAGE directive). The PAGE directive must be written on the same line as the CHAPTER directive, the C directive beginning the line. e.g.

C 2 P 8

As usual, the first page of the chapter, P 8, will be stored on the next available sector.

This is useful when Chapter 1, say, contains several routines which are also required in Chapter 2. If these are put at the beginning of Chapter 1 (pages 1 to 7) then Chapter 2 can call in the routines of Chapter 1 by direct transfers of control without doing a chapter change each time. Care must be taken that the relevant routines of Chapter 1 are really in the computing store. When a chapter is transferred to the computing store only the relevant pages are read down, other pages being unaffected.

It is also possible to predict the highest page of a chapter e.g.

C 2 P 8 - 11

If chapter 2 spreads beyond the end of page 11, then fault 9 is reported by the Input Routine. This is useful:-

(i) when pages 12 onwards are being used as working space.

(ii) when the precise sectors occupied by the chapters are required, since chapter 2 will now occupy four sectors of the backing store corresponding to pages 8 to 11 even if the last instruction of the chapter was only in page 9.

(iii) when a lengthy MCORRECTION is anticipated.

If a chapter occupying say pages 8-11 is expected to spread into Page 12 because of a long correction, allowance must be made for this in the original chapter directive, for if the chapter as read in only occupies 4 pages, only 4 sectors will be allowed to it on the drum, and a correction running on to the next sector will spoil the beginning of the next chapter. Therefore a directive P 8-12 should be written to ensure that 5 sectors are left on the drum for the chapter.

It is unnecessary to predict a highest page 15 for reason (i) alone, as no chapter may spread beyond the end of the page 15. A chapter may be directed to begin in page 0, but it cannot then be transferred from the backing store to the computing store by ACROSS, DOWN, UP or ENTER which use the chapter changing sequence.

4.12 Sector

If a chapter is required to begin on a particular sector of the drum e.g. for organising transfers from the drum store to the computing store without using the chapter changing sequence, then a SECTOR directive should be given on the same line as the CHAPTER directive, either before or after a PAGE directive, if any. e.g.

C 3 S 25

will cause Chapter 3 to be stored on sector f + 25 where f is the FIRSTSECTOR which is usually 128. The sector number is always relative to the FIRSTSECTOR.

4.13 Line

Usually items within a chapter are stored consecutively in the computing store, short integers when encountered going into the next available half register, instructions going into the next whole register and long numbers going into the next even numbered register. If the next and subsequent items are required to go into some other register, a LINE directive, followed by an address may be given. The address may be a fixed numerical address or may consist of a fixed numerical part and one v floating address only. e.g.

L2.0 or L128 } These cause subsequent items to go into registers 128 and 50v6 respectively.
L 50v6 }

The second kind is useful when it is required to leave some working space within a routine. A LINE directive may occur anywhere in the middle of a routine the subsequent items being a continuation of the current routine. When a new routine is required to begin in a particular register the LINE directive should be given before the ROUTINE directive; if the ROUTINE directive precedes the LINE directive, the address v0 is of course not the address of the first item written in the routine.

The address after a line directive may contain at most one symbolic address, which may be a v-reference, a pre-set parameter, or an * (see 5.4). An * in an L directive refers to what would have been the next address, so that for example the last instruction of a quicky can be overwritten by writing LINE-1* immediately after the quicky.

The absolute and symbolic parts of the address in a line directive may each be greater than 16.0; they, and their sum, are both interpreted modulo 32.0.

It is important to keep in mind that v-references, preset parameters and * all take account of half registers. For example, suppose

= 2, = 7 (3

was written in line 2.20 say, then the value of v3 is 2.20+. A directive LINE 1v3 would then set the current address to 2.21 +, so that an instruction immediately after the directive would go into 2.22, not 2.21 as might be expected. LINE 0+v3 would set the current address to 2.21.

With the L directive the highest page occupied by a chapter is correctly set, (e.g. for later reading the chapter to the computing store). However the first page is not necessarily correctly set. If for instance a directive C1 P2 is later followed by a directive L1.0 then page 2 wrongly remains the first page and no check for this kind of blunder is included.

A chapter can be extended into the second half of the computing store only by the use of a line directive - over-running the end of Page 15 will produce fault 8. Once a chapter has been extended into the second

half of the store the only check on its length is the "overflow predicted page" check; there is no check on overflow of page 31.

Instructions, integers and numbers can be written in the part of a chapter occupying the second half of the store, and labels can be set by bracketing items. However, a label set in the second half of the store can only usefully be referred to by an accumulator instruction, since other instructions can refer to the first half only of the computing store.

## 4.14 Mcorrection

This directive is similar to the LINE directive. The address must contain one cross reference e.g.

M 4v1/2

Unlike L, the effect of an M directive is to cause the Input Routine to terminate the current routine and chapter. It then transfers the relevant sector of the chapter containing routine 2 to the computing store and causes the subsequent information to be stored beginning in the specified address. After an MCORRECTION all v floating addresses must be cross references and no new labels or new CHAPTER or ROUTINE directives may be given. MCORRECTIONS should be placed at the end of the tape, immediately before the ENTER directive.

The MCORRECTION may be used

(i) to rectify a known programming error e.g. an addition instruction has been incorrectly written as a subtraction

```
M    4v3/6
420  v1/6
E    v/1
```

(ii) to detect programming errors by doing a running check. This can be done by writing a special printing routine or chapter and putting in blisters at the points where a print-out would be useful e.g.

```
C 99                          The blisters
R 999
.                          ⎫
.                          ⎪  The 4 or 5 instructions
.                          ⎬  obliterated by the first
.                          ⎪  MCORRECTION
590  v10                   ⎭
400  32      (1            ⎫
.                          ⎪  The 4 instructions
.                          ⎬  obliterated by the
.                          ⎪  second MCORRECTION
590  v10                   ⎭
:
106  0       (10           ⎫
206  0                     ⎪
Q 8                        ⎬  Print page 0 as short
176  127                   ⎪  integers
186  1v10                  ⎭
```

```
400   16.6  ]
300   9     }     Prin  L16.6 to 9
Q 10        ]     deci al figures

UP

M  3v2/6    ]     Call in first blister (DOWN
D  v/999    }     takes 4 or 5 registers)
M  v/11     ]     Call in second blister (DOWN
D  v1/999   }     takes 4 registers here)

:

E  v/1              Enter
```

The only symbolic address permitted in an M directive is a v-reference, and there must be only one. The address takes account of half-registers as in L. As in cues, addresses such as 2.0/15 can be written after M: in this case the correction is made to the chapter containing routine 15 and the correction starts at 2.0.

M corrections cannot be made to programme occupying the second half of the store. The absolute and symbolic parts of the address in an M directive are interpreted seperately modulo 16.0, and so is their total.

If the address after an M is written in the form a.b, without any v-reference or /, the address is taken to specify sector a, line b. For example, M129.10 causes the succeeding programme to go on to sector 129 starting at line 10. In the programme following such an M heading cross references may be used, but no *.

When using correction tapes, it is important to remember that "Start without Clear" unsets all preset parameters and resets the "first sector" to 128, so that an entry cue at the end of M tape, read in after "Start without Clear", will overwrite the last two lines of sector 128 unless a new F-directive has been put at the head of the correction tape.

An M correction is not fully compiled until the start of the next directive is read, so that an M tape should finish either with an ENTER or with a dummy directive, such as M→.

## 4.15 Interlude and Jump

These directives permit a short burst of computing while the programme tape is being read in (c.f. chapter 0 of Autocode). The interlude is commenced by the directive

INTERLUDE

Subsequent items go into registers 1.0 onwards, but must not extend beyond the end of page 13, and are stored on sectors 114 to 126. The interlude is terminated and entered by a directive

JUMP

which reads the interlude from the backing store to the computing store and transfers control to register 1.0. To return from obeying the interlude to reading more of the main programme at the point where it was interrupted, control is transferred to register 15.0. e.g. to set x1 equal to the value of v1 of the current routine (i.e. x1' = v1):-

```
INTERLUDE
400     27.56
410     24.46
590     15.0
JUMP
```

The interlude may use pages 0 to 14 as working space. The usual entries are in registers 0 to 12 of page 0 but the remainder of page 0 and page 14 are not cleared and contain junk. The Page 0 used during input is stored on sector 127.

Labels during the interlude are labels of the current routine of the main programme, and a ROUTINE directive in the interlude terminates this current routine. Quickies and LINE directives are permitted, but it is not possible to write ACROSS or DOWN in an interlude (without a special device since cues are filled in at the ends of chapters). A CHAPTER directive must not be given in an interlude. A second interlude is written on the same sectors of the backing store, overwriting the first interlude. An interlude may be entered any number of times by directive JUMP alone.

Details, contained in appendix 2, enable interludes to be written for:-

(FIRSTSECTOR x1)
(SECTOR x1)
(PAGE x1 - x2)
(x1 = current sector)
(clear labels list only)
(Print preset parameters)
(Print selected labels)
(x2 = -x1)
(x3 = x2 times x1)
etc. etc.

## 4.16 Wait

The directive WAIT, or the figure shift symbol →, interrupts the programme input and produces a repeated hoot. When handswitch 9 is depressed and released, input carries on from where it left off.

This WAIT hoot can be put in anywhere, even half-way through an instruction, but it is mostly used at natural breaks such as the ends of chapters. It is often a good idea to put a WAIT hoot symbol on the programme tape immediately before the Enter directive, so that if necessary a correction tape can be read in before the Enter is obeyed. A correction tape should of course finish with a copy of the Enter directive or, when it becomes one of several, another WAIT.

## 4.17 Name

The directive NAME or N causes the succeeding characters on the programme tape to be copied on to the output tape, up to the first CR symbol, but only if * printing is set; otherwise the characters up to CR are ignored. Erases are not copied to the punch. The Name directive provides a form of title which does not get stored. Each library tape has a name sequence at the beginning.

## 4.18 Other Directives

Another use of the letters L, P and S is described in section 5.5.1 (page 35). See also section 6.4 (page 41) for the use of the letter shift character "?".

# CHAPTER 5

## SYMBOLIC ADDRESSES

### 5.1 General description

One kind of symbolic address was introduced in chapter 2 (q.v.). These v symbolic addresses are usually specified by a label after an instruction. It is also possible to specify a v symbolic address by labelling a short integer or a long number e.g.

```
= 25        (2
+3.1416     (3
```

and also by labelling the groups of instructions, QUICKY ACROSS DOWN and UP in which case t he label must follow the quicky number or the address. Finally, a v symbolic address may be set by an equation e.g.

```
v1 = 66
```

This is not an instruction occupying a register but rather is a directive to the Input Routine. The ways a symbolic address may be specified are classified by the item labelled:-

Item 1 Long number labelled

Item 2 Instruction labelled

Item 3 Integer labelled

Item 4 Equation set.

If QUICKY, ACROSS, DOWN or UP are labelled this is classified as item 4 and v0 the address of the beginning of a v-routine is also treated as if it were equation set. The item labelled is significant with a symbolic address of type 3 instructions only.

The address part of an instruction can be composed of any number of symbolic addresses which are added together e.g.

```
400  v1x2        (see below)
```

An address may also contain one fixed numerical part which must be written first e.g.

```
590  2v3
```

transfers control to the second register after the label 3 and

```
400  -4v1
```

copies to the accumulator the long number which is 4 medium registers back from the label 1, the minus sign referring to the fixed numerical part only. One label may also be included, the label and symbolic addresses appearing in any order. However the advantage of occasionally saving a few milliseconds of input time is dubious and labels are best put well to the right of the address.

When a long number is labelled 1, the four short integers comprising it are referred to as v1, 0+v1, 1v1 and 1+v1 respectively. When an instruction is labelled 2, the function and B-digits are referred to as v2 and the address of the instruction as 0+v2 e.g.

```
016  0+v2
```

Short integers may themselves be written in terms of symbolic addresses and are always treated as the address part of instructions,

> as type 2 instructions

= as type 1 instructions

and ≠ as type 4 instructions

e.g. = v1, ≠ 6v2

With directives ACROSS, DOWN, ENTER and MCORRECTION the symbolic address must specify a routine which in turn specifies the chapter. Consequently the permitted address is restricted to

(i) a fixed numerical address (optional) together with one v symbolic address which is a cross reference e.g.

A v/1

A 4v1/2

(ii) a fixed numerical address together with a routine number e.g.

E 1.0/1

transfers down the chapter containing routine 1 and transfers control to register 64.

With the LINE directive the address may consist of a fixed numerical part followed by one v symbolic address only e.g.

L 4v1

or a preset parameter or an asterisk. With L and M the symbolic address must of course be a backward reference, i.e. the symbolic address must have been previously set by an item further back on the tape.

## 5.2 v symbolic addresses

These symbolic addresses are associated with labels. One hundred different v symbolic addresses are permitted with each v-routine (though usually only about ten are used). In a complete programme the total number of different v labels set must not exceed 1023. v101 to v119 correspond to quickies 1-19 (see section 4.4.1 above).

## 5.3 n symbolic addresses

This symbolic address is minus the corresponding v address e.g. if

v1 = 6

then 400 n1

has address equivalent to the written register address -6 or 2042. This kind of symbolic address is particularly useful when say, the number of long numbers in a list is required e.g.

```
+1.1      (1
-1.2
 :
 :
+10       (2
```

then 300 v2n1

puts the number of long numbers minus 1 into Sac (Since long numbers are labelled the medium register addresses are halved during input - see later). n on its own or n0 is minus the address in which the routine begins.

5.4 * symbolic addresses

The relative address asterisk is set to the address of the item (an instruction or short integer) in which it occurs. It is equivalent to v1(1 but has the advantage that a label is not used. e.g.

```
590  2*
```

transfers control to the address occupied by the next but one instruction. Similarly a continuous hoot at an audible frequency might be

```
580  0      Hoot
300  -25    S' = -25
380  *      Jumps to itself 25 times taking 1½ ms.
590  -3*    Jump back 3
```

With all symbolic addresses it is recommended that the fixed numerical part should be kept small or the advantage of symbolic addresses are diminished; the appropriate item should be labelled.

5.5 x symbolic addresses

(i) x-Routines (q.v.) are written in terms of a special symbolic address x or x0. This address is automatically set to the address of the beginning of each x-routine. The address x0 can only be set in this way; it cannot be set by an equation. Instructions of an x-routine might be

```
400  48x
590  7x
```

(ii) Preset parameters x1 to x100. It is sometimes convenient to have symbolic addresses which are not associated with any particular routine and which are easily varied. These preset parameters are always set by an equation e.g.

```
x99 = 6
x2  = 31.0
```

which is usually conveniently placed at or near the beginning of the tape. The preset parameters may however be set or reset at any time.
e.g. A general routine might be written to solve simultaneous equations in which the coefficients are stored in consecutive long registers in the computing store. The medium register address in which the first coefficient is stored might be specified by the value of x1 and an instruction of the routine might be

```
406  x1
```

Also, the number of equations which are to be solved might be specified by the value of x2 and an instruction of the routine might be

106 -1x2

(The distinction between an integer and an address is rather arbitrary c.f. instructions of which the address part is regarded as an integer.)
The equation setting a preset parameter may contain on the right hand side a preset parameter which has been previously set. e.g.

x2 = 3x1

sets x2 to 3 plus the value of x1. Any number of preset parameters may occur on the right hand side, but only one fixed numerical part which must come first e.g.

x2 = 3x1x6x7x7

the sum being implied. It is not possible to subtract preset parameters in this way but in the rare cases when it is desired it can be done by using an intermediate n e.g.

v99 = x2

300 x1n99

puts x1 - x2 into Sac. Preset parameters may also be included in the right hand side of an equation setting a v symbolic address but in neither x nor v setting equation may a v, n or * occur on the right hand side.

### 5.5.1 L, P and S on the right hand side of equations.

The letters L, P and S can be written on the right-hand side of equations, and have value the current line, page and sector respectively. At most one can appear in any one equation, and must follow any absolute or other symbolic part of the right-hand side.

L has the value the address of the next register, taking account of half-registers, P and S the current page and sector numbers respectively.

After the last item of a page has been read, the register indicator in the input routine is advanced, but the page and sector indicators are not changed until the start of the next item, so that P and S give the page and sector of the last item read.

L, P and S can be used in equations setting x and v symbolic addresses.

## 5.6 Filling in symbolic addresses

Like all written information which is fed into the computer via the Input Routine a symbolic address is a kind of shorthand for the binary digits of the machine address into which it is translated. It is however often convenient to think of symbolic addresses as the equivalent written medium register address. Just as a written register address is sometimes halved or doubled during input according to the type of the instruction, so also symbolic addresses are similarly halved or doubled. For type 3 instructions e.g. B' = n the symbolic address is halved if a long number is labelled, doubled if a short integer is labelled, and has the same value as the equivalent written medium register address if an instruction is labelled or if the symbolic address is equation set. The following are always regarded as equation set:-

v0, *, x and x1 to x100 as well as v1 to v100 when a QUICKY, ACROSS, DOWN or UP is labelled.

For a general written address the fixed numerical part is read first. This must be consistent with the size of the machine so that for

| | | | | | |
|---|---|---|---|---|---|
| type 1 instructions | -1023 | ≤ | M | ≤ | 1023 |
| type 2 instructions | -511+ | ≤ | H | ≤ | 511+ |
| type 3 instructions | -1023 | ≤ | M | ≤ | 1023 |
| type 4 instructions | -2046 | ≤ | L | ≤ | 2046 |
| type 5 instructions | -1023+ | ≤ | H | ≤ | 1023+ |

(If any of these inequalities is not satisfied this is what is called address overflow). The fixed numerical part is then appropriately halved or doubled if necessary and stored as a short integer. If the result is not an exact integer (+ counts as $\frac{1}{2}$) this is what is called address underflow. A general address may also consist of several symbolic addresses. These are treated in the order that they occur. If the value of the address has not been previously set details of this forward reference are stored for filling in later. If the value of the address has been set for each symbolic address part it is appropriately halved or doubled and then added to the machine address so far and stored as a short integer. The integral part of the result is then taken, any halves or quarters being discarded. The result is taken as modulo 1024 except in the case of type 5 instructions so that with symbolic addresses, after the fixed numerical part of the address has been read both overflow and underflow are ignored. It is convenient to regard symbolic addresses or the equivalent medium register address as being

modulo 1024 for type 1 instructions
modulo 512 for type 2 instructions
modulo 1024 for type 3 instructions
modulo 2048 for type 4 instructions
modulo 1024 for type 5 instructions

This information is summarised in table 1 for the symbolic address of an instruction. M is the equivalent medium register address, and the entry for the type of instruction and the item labelled is the value of the machine address. Type 5 instructions are here regarded as constituting one machine instruction with an address of 11 binary digits.

Table 1.

| | t = 1<br>C' n or "=" | t = 2<br>T'=H or ">" | t = 3<br>B'= n | t = 4<br>A'= L or "≠" | t = 5<br>B = H |
|---|---|---|---|---|---|
| modulo | 1024 | 512 | 1024 | 2048 | 1024 |
| i = 1<br>number ( | M | 2M | M/2 | M/2 | 2M |
| i = 2<br>instruction ( | M | 2M | M | [M/2] | 2M |
| i = 3<br>integer ( | [M] | 2M | 2M | [M/2] | 2M |
| i = 4<br>equation set | [M] | 2M | [M] | [M/2] | 2M |

The square brackets denote "the integral part of" and could enclose every entry but have been omitted in those cases when a half or quarters cannot arise in finding the machine address. i denotes the item number (see sec. 5.1.).

If something different from the above is required to be done to the written medium register address, one of the characters $>=$ or $\neq$ may be inserted before the complete address. The type of the instruction is then changed to type 2, type 1, or type 4 respectively. Thus the address is treated as a short integer though of course the item remains an instruction. The characters $>=$ or $\neq$ are rarely required before an address and the temptation to insert them unnecessarily should be resisted as errors are easily made by doing so. The cases when they are essential are usually with instructions of type 3 e.g. when a long number is input as 4 short integers one of which is labelled 1, then

106 $\neq$ v1

406 0

puts the long number of which the labelled short integer is part, into the accumulator.

A preset parameter must not be reset in terms of itself, that is to say the same x parameter must not appear on both sides of an equation. If such an equation is written, it will reset the preset parameter to a wrong value.

# CHAPTER 6

## AUTOMATIC PRINTING FACILITIES

### 6.1 Fault print

During input various obvious programming blunders are detected by the Input Routine which immediately prints:-

FAULT n

where n indicates the particular fault as listed below and comes to a 99 stop so that a mark can be made on the tape. The fault number n is, at this stage, also contained in SAC and the last figure shift character read is in B6. Except for fault 9, on giving a prepulse the Input Programme ignores all further characters until the next carriage return, printing out the characters ignored, and then continues reading the tape in the normal way. A fault 9 for overflow of the predicted page is reported at the end of the chapter, which is indicated by the next CHAPTER or ENTER directive, and comes to a 99 stop. On giving a prepulse it continues to read the C or E directive and the programme will be composed correctly if the overflow of the predicted page does not matter.

#### 6.1.1 Fault 1 Address underflow

This occurs when a fixed numerical address or the fixed numerical part of a general floating address is not an even whole register address for type 4 instructions, or is a right half register address for types 1, or 3 instructions e.g.

```
400  41
300  0+v1
```

} For both of these fault 1 is reported.

When the written medium register address is appropriately halved or doubled the machine address must be a whole integer.

#### 6.1.2 Fault 2 Address overflow

This occurs with a fixed numerical address or the fixed numerical part of a general floating address which does not satisfy

| | |
|---|---|
| $-1023 \leqslant M \leqslant 1023$ | for types 1 and 3 instructions |
| $-511+ \leqslant H \leqslant 511+$ | for type 2 instructions |
| $-2046 \leqslant L \leqslant 2046$ | for type 4 instructions |
| $-1023+ \leqslant H \leqslant 1023+$ | for type 5 instructions |

#### 6.1.3 Fault 3 Spurious character

This is reported when a character occurs in a position where it cannot be correct e.g. the v in

```
40v  2
```

will cause fault 3 to be reported. As with all faults, when a character is reported as spurious the character itself is necessarily at fault if, and only if, all the preceding characters of that item are correct e.g. an instruction

```
300  0+v1
```

is reported as fault 1 for address underflow when it may be that the actual fault is that the function digits should have been 20. When the machine itself is not functioning correctly it is found that fault 3 is reported more often than other faults.

6.1.4 Fault 4 Label set twice

A v-floating address is specified twice e.g. if two labels with the same number occur in the same v-routine, fault 4 is reported when the second label is read. Fault 4 also occurs when two routines have the same number, as then v0 is set twice.

6.1.5 Fault 5 Label not set

With an L or M directive the v-floating address must have been previously specified; if not, fault 5 is reported. Other references to v-floating addresses which are not specified do not cause a fault stop but when the ENTER directive is read the Input Routine prints e.g.

v6/1 NOT SET

followed by the address of the offending reference in the form Sector . line. After each there is a high pitched hoot lasting a second but the programme is entered. This may be useful when a programme is being tested in sections e.g. suppose routine 25 has not been written but is called in by 590 v/25. Since v/25 is not set, when this instruction is obeyed control is transferred to register 0 which contains a dummy instruction and the machine comes to a 99 stop in register 1 when routine 25 would have been reached.

6.1.6 Fault 6 Preset parameter not set

Unlike v-floating addresses, in a reference to a preset parameter x1 to x100 the preset parameter must always have been previously specified (by an equation).

6.1.7 Fault 7 Too many references

Backward references are filled in immediately but details of each forward reference are kept in a list and the reference is filled in at the end of the routine containing the label. If the number of unfilled references exceeds 278, fault 7 is reported. This fault is rare and has so far only occurred in programmes specially written to produce this fault.

6.1.8 Fault 8 Overflow page 15, or Interlude too long

Fault 8 is reported if a chapter extends beyond the end of page 15 without a line directive to the second half of the computing store. The fault is reported after the first digit of the instruction after the end of page 15 has been read, or after the =, >, ≠, +, or - introducing an integer or floating-point number. If the tape is pulled back to the CR terminating the previous item before the prepulse is given, the rest of the chapter will be compiled in the second half of the store.

Interludes may not go beyond the end of page 13. (See paragraph 4.15).

6.1.9 Fault 9 Overflow predicted page

If a PAGE directive specifies the highest page of a chapter and if the chapter extends beyond the end of the specified page then fault 9 is reported at the end of the chapter.

6.1.10 Fault 10 Title too long

Since the title is stored on the first sector of the programme the number of characters in the title is limited to 244 characters. Fault 10 is often reported when the FS FS terminating the title is omitted so that the first chapter is wrongly read as a TITLE.

6.1.11 Fault 11 Wrong quicky number

Not all the Quicky numbers 1 to 20 are used and if e.g. the unspecified

Q3

is written fault 11 is reported.

If a non-existent quicky is called for by the automatic selection, e.g. v103, fault 11 is reported at the end of the chapter, and the reference remains as to an unset label. No more filling in of quickies will be done at the end of the chapter.

6.1.12 Fault 12 Too many labels

Details of every v-floating address which is specified are kept in a list with space for 1023 labels. Fault 12 will be reported at the end of the routine in which the $1024^{th}$ is set.

6.1.13 Fault 13 Chapter numbers, parameter numbers or label numbers over 100

The chapter list and preset parameter list have spaces for numbers 1 - 100 and the label list of any given routine has only spaces for numbers 1 - 100.

6.1.14 Fault 14 ? number greater than 10

If the number written after a ? is greater than 10, fault 14 is reported (see section 6.4.).

6.2 Error Print

Whereas a fault print indicates certain programming blunders which are detected during the input of the programme, some programming blunders are detected during the actual running of the programme. Many of the quickies evaluate elementary functions of the long number in the accumulator over a restricted range. If the Quicky is entered with the accumulator contents out of range, control is directed to a part of the input programme for an error print e.g. Q12 finds the square root of a positive number or zero and calls in the error print if it is entered with a negative number in the accumulator. The word ERROR, the quicky number, the contents of B1 to B6 and the contents of the accumulator as four short integers (in the order exponent, least significant ten binary digits of the fractional part, middle ten binary digits and most significant ten binary digits i.e. H0, H0+, H1, H1+ if the number were in L0) are printed and the machine comes to a hoot stop.

It may be that an error print has been anticipated by the programmer (or has occurred when the programme was last run) and further printed information is required. This can be arranged by writing a printing routine which begins in register 64. If at the hoot stop, key 9 of the handswitches is tapped or if the handswitches are negative, control is directed to register 64. At this stage, the contents of the accumulator, B1 to B6 and page 0 are restored to their values on entering the error print, but S, St, Bt and T, the selected sector are changed. The error print routines all work in page 0 which is temporarily stored on sector 479.

It is also possible for a programmer to arrange to call in the error print if something goes wrong with one of his own routines. This is done by putting an identification number in Sac or B7 and transferring control to register 9. The number in Sac will be printed as the quicky number. As a prelude to doing a postmortem it may be useful to print out the B-registers and the accumulator by transferring control to register 9 by a manual instruction and so doing an error print.

## 6.3 Asterisk print

It is frequently desirable to know the equivalent fixed numerical addresses for the v-floating addresses and the sectors on which each chapter is stored e.g. to identify an error print when B1 gives the address at which the quicky was called in as a closed subroutine. If, and only if, there is an * on the tape at the beginning of a line the following information is subsequently printed during input.

(i) At each chapter directive:-

The letter C, the chapter number, the sector on which it begins and the first page.

(ii) At each routine directive:-

The letter R, the routine number and the register address in which it begins i.e. v0. The register address of the other labels can be found by counting the registers from the beginning of the routine.

A second asterisk at the beginning of a line causes Fault 3 to be printed.

A binary tape is also provided which when read in by binary input prints the addresses of all the labels and interludes can also print labels.

## 6.4 Query print

In testing a programme it is often desired to print out information at various stages and with the query print this may be done with little programming effort. The information that is printed is determined by the query number which is set by a directive e.g.

? 3

for printing the contents of B3. The places in the programme where the actual printing is desired are specified by a single figure - shift character n at the beginning of an item e.g.

420 v1

n

410 40

For all query numbers 1- 10, except ? 8, the n at the beginning of an item causes 12 or 13 instructions to be inserted in the programme, a dummy being needed if the register is odd; ? 8 is a special case (see table 2) for which only 3 instructions are inserted for the n. If there is no query directive, or if there is a directive ?0, then no printing is done and the n at the beginning of an item is ignored. During a query print, subroutines of the Input Routine are obeyed in page 0, the contents of which are temporarily stored on sector 479. At the end of the query print the B-registers including Sac, the B-test register, the accumulator and page 0 are restored. However the Sac-test register may not be restored and T, the sector selected, is changed to 479.

The n need not be followed by a carriage return and line feed so that if the line feed of the CR LF terminating an item is converted into an n by inserting the most significant digit with a hand punch, that item is then terminated by the CR alone and n at the beginning of the next item is obtained. Thus a query print can be inserted without reperforating the programme tape. However it may be desired to print different information at each point when it is convenient to set the query number just before each print e.g.

?6, n

?9, n

(The comma terminates the query directive so the n is at the beginning of an item).

With each query print a letter of the alphabet is printed, letters being allocated in the order that the n's occur on the programme tape, excluding n s which are ignored. For ?8 this is all the printing that is done. For other ?s the letter is immediately followed by the state of the B-test register, +, -, or 0, and on the next line the "return address" preceded by M. The "return address" is the address of the last of the inserted instructions. The remaining information printed is shown in Table 2.

TABLE 2

| Query Number | Information printed |
|---|---|
| 1 | the contents of B1 |
| 2 | the contents of B2 |
| 3 | the contents of B3 |
| 4 | the contents of B4 |
| 5 | the contents of B5 |
| 6 | the contents of B6 |
| 7 | the contents of B7 |
| 8 | (letter only) |
| 9 | the accumulator as 4 short integers |
| 10 | the B registers, and the accumulator as 4 short integers |

A ? number bigger than 10 gives fault 14.

When using the query print care must be taken to allow for the inserted instructions when referring to registers by use of nearby symbolic address. For example, in

```
490    2*
n520   2
410    0.32
```

the 490 jump is to two registers further on, which will be the 410 instruction only if no extra instructions have been inserted for the n; if instructions have been inserted, the jump is into them.

An n calling for ? printing cannot be written in Page 0. This is detected as fault 3.

Several ? directives can be given in one programme tape, and an n is interpreted according to the last ? directive.

If handswitch 4 is set when an n is read at the beginning of an item, any ? directive is over-ridden and the n is ignored. This means that a programme with several ns calling for query printing can be read in without any of the printing routines. However, if handswitch 4 is set when the Initial Transfer Button is pressed, it has a different meaning (see 7.4.6), so it must be set after the Initial Transfer Button is pressed.

# CHAPTER 7

## RUNNING A PROGRAMME

### 7.1. Tape preparation

The written programme must first be transferred to paper tape. To do this sets of tape editing equipment are available separate from the computer itself. The complete set consists of a keyboard similar to that on a typewriter, a punch, a teleprinter and a tape reader. When a key is pressed the corresponding character is punched on a tape and is also printed on a roll of paper. For letters a special character called letter shift must first be punched and followed by a figure shift character when it is required to return to figures. The keyboard has a locking device to ensure that the shift characters are not omitted. If a mistake is noticed it may be possible to back space and convert the wrong characters into "erases". The tape reader is used to copy tapes e.g. the library tapes or for correcting part of a tape (the corrected tape can be joined with opaque sellotape); a facsimile of the tape going though the reader is produced by the punch and a printed version is produced by the printer. The punch may be switched off so that only the printed version is obtained e.g. to print results from the output of the computer.

Each item or line of the written programme, e.g. an instruction or a directive is followed by the two characters "carriage return" and "line feed" in that order. Except for long numbers, items may also be terminated by a comma. After the item has been terminated all further CR's or LF's are ignored until the next item is commenced. Items are commenced by a decimal digit for instructions, a plus or minus for long numbers, a $>$ $=$ or $\neq$ for short integers or a letter shift character for directives. Between items FS or blank tape, and LF is ignored but once an item has been started an unnecessary figure shift or line feed is a spurious character producing fault 3. The characters space and erase are ignored everywhere, and it is usual to separate the three digits of the function and B-register from the address by a single space. It is also convenient to insert about three spaces before each label. Reasons for this are:-

(i) The labels are easily located on the print out.

(ii) In finding an item on the tape in order to correct it with a hand punch, the nearest label can first be easily located by the adjacent block of spaces.

### 7.2 Layout of the backing store

The Input Routine occupies sectors 0 to 63 and may be isolated by the switches on drum 0 for columns 0, 1, 2 and 3: (or columns 0 and 1 on the new large drums) so that it cannot be overwritten. If the Input Routine is overwritten it may itself be read in, a tape being provided on which instructions are coded in binary suitable for a short input programme which is on sector 1, sectors 0 and 1 being permanently isolated. While the Input Routine is being read in lists of labels etc. are compiled on sectors 64 to 113, interludes are stored on sectors 114 to 126, and sector 127 is used to store page 0 during input, (during a fault print or an interlude). When the programme is entered the Chapter Changing Sequence is transferred to sector 478 and sector 479 is frequently used to store the contents of page 0 during the running of the programme e.g. during chapter changing and automatic printing. On sectors 480 to 511 are usually kept the engineers test routines, these being isolated by the switches on columns 6 and 7 of drum 3 (column 7 of drum 1 on the new large drums).

## 7.3 Layout of the computing store

When a programme is entered the following is put in Page 0:-

| Register | | | |
|---|---|---|---|
| 0 | +0 | | |
| 1 | | | |
| 2 | -1 | | |
| 3 | | | |
| 4 | 670 | 479 | Entry to the chapter changing sequence |
| 5 | 690 | 0 | |
| 6 | 670 | 478 | |
| 7 | 680 | 0 | |
| 8 | 597 | 0 | Return |
| 9 | 670 | 479 | Entry to the Error print |
| 10 | 690 | 0 | |
| 11 | 670 | 5 | |
| 12 | 680 | 0 | |

The long numbers +0 and -1 are frequently useful and are used by many of the quickies. The computing store from line 0.14 onwards is cleared to programme zero (990 0) and B-registers 1 - 6 are clear. The vacant space in page 0 from register 13 is available for ad hoc working space but it must be remembered that the contents of Page 0 are destroyed whenever the initial transfer button is pressed. Registers 32 to 39 are used by quickies and some library x-routines may also use registers 24 to 31. It is generally advisable for programmers to avoid these registers and use only registers 13 to 23 and 40 to 63 in page 0 except for very short term work.

## 7.4 Starting procedures

The programme tape is placed in the reader and the initial transfer button pressed. This transfers the contents of sector 0 to page 0, transfers control to register 0, and obeys the first instruction which is

610 2+

which reads the handswitches. This is the only time (except after an error print) that the Input Routine reads the handswitches so they can immediately be set if required during the running of the programme. Depending on the initial setting if the handswitches various modes of starting are available. Except for key 9 only the most significant key which is non-zero is relevant i.e. if two keys are set to the "one" position the least significant is ignored. The most significant key in the "one" position has the following effect.

7.4.1 Keys all in the zero position Start clear

This is the normal way of starting. The backing store from sectors 64 to 511 is first cleared to programme zero 990 0 everywhere. Just before the programme is entered the computing store is cleared to programme zero, but the labels list, etc., are left on sectors 64 to 127.

7.4.2 Key 0 Start no clear

This is the same as start clear except that the backing store, including the labels list etc., is not first cleared. This may be used to re-enter a programme by reading an ENTER directive from tape (assuming the labels list etc., to be intact) perhaps with an MCORRECTION or at the beginning of a second data tape. Unless the required FIRSTSECTOR directive is given the new entry cue is stored on the usual sector 128. It is possible to re-enter the programme at various different points.

7.4.3 Key 1 Engineers' tests

These are a programme of routines designed to check that the various parts of the machine are functioning correctly. Key 1 should be returned to zero and the prepulse button pressed. The engineers' tests can be run whenever the machine would be otherwise idle.

7.4.4 Key 2 Tele-input

A short input routine stored on sector 1, which is normally isolated, for reading binary tapes. This is a fast compact input routine for reading in data or fully developed programme. Tapes for input by tele-input are usually produced by the computer by means of tele-output, described in the next section.

7.4.5 Key 3 Tele-output

This and the following settings require key tapping to indicate required sectors on the drum. The bottom row of 10 keys on the control console are numbered 0-9, and have two "set" positions, up and down, as well as the central "unset" position. The down position is spring loaded so that if a key is pressed down it returns to the "unset" position as soon as it is released. "Handswitch tapping" refers to a procedure whereby a three decimal digit sector number is indicated by pressing and releasing three keys in succession; three digits must always be given so that sector 99 say is specified by pressing ("tapping") first key 0, then key 9, and then key 9 again. The handswitch tapping routine is included in the programme on sector 2; it is obeyed in page 0.

Tele-output is a routine for punching out specified sectors of the drum in binary, in a form suitable for re-input by tele-input. The information on the output tape corresponding to each sector has a few characters at each end which tele-input will interpret as information about where the sector is to go; between each sector of information tele-output leaves one inch of blank tape. With each sector of information is also punched a check sum, the value of which is checked by tele-input when it reads in the tape.

To punch out information from the drum, using tele-output, key 3 is set to the up position and the Initial Transfer Button pressed. Key 3 is then return to zero and two three-digit sector numbers are tapped, the first and last sectors respectively that are required to be punched out on the binary tape. These sectors are then punched out with a leader of 10" of blank tape; after the punching tele-output re-enters the handswitch tapping programme, waiting for a third number to be tapped, the "first sector" number.

(i) if this third number is tapped as zero, (0, 0, 0) tele-output punches out a single character, tele-input's hoot loop character. When tele-input comes to this character on re-input of the tape, it comes to a hoot stop.

(ii) if the third number tapped is not zero, it is assumed that the information punched out included a programme, and that the third number tapped is the number of this programme's "first sector". Instead of a hoot loop character there is punched on the tape a sequence of characters which when read by tele-input, cause the programme to be entered according to the entry cue stored in lines 62, 63 of its "first sector". (see section 4.10).

In both cases, 1" of erases is punched at the very end of the tape.

If several groups of sectors are to be punched out, the entry warning sequence is only wanted at the end of the last group. This can be achieved by re-entering tele-output after each group (except the last) has been punched: when tele-output re-enters the handswitch tapping programme after punching out a group of sectors, instead of tapping a sector number handswitch 3 should be re-set and the Initial Transfer Button pressed, to re-enter tele-output for more punching.

See also section 7.4.12 for another mode of operation for tele-output.

7.4.6 Key 4 Rescue

This is used to preserve the contents of the computing store by writing it to some convenient part of the drum.

Key 4 is returned to zero and one three-digit sector number is tapped; the contents of the computing store are written to this sector and the 31 following sectors and the programme comes to a hoot stop. Since page 0 is destroyed when the Initial Transfer Button is pressed, what is copied on to the sector tapped on the handswitches is sector 479, that is the contents of page 0 at the last chapter change. Pages 1-31 go to the next 31 sectors unscathed.

7.4.7 Key 5 Post mortem instructions

This is for printing out the instructions in the computing store or on the drum in the usual decimal form of two function digits, a B-digit, and a fixed numerical register address.

Key 5 is returned to zero and a three digit number is tapped. If this number is in the range 0-31, it is taken as a page number, if in the range 32-999 it is taken as a sector number, and the contents of that page or sector are printed out as instructions. Sectors 0-31 cannot be "post-mortemed"; since they will usually contain part of the Input Routine this is little loss.

After punching out the contents of one page or sector, the programme comes to a 99 stop; on a prepulse the next page or sector is printed out, and so on. After page 31, sector 32 is punched; after sector 999, sector 1000.

If the first seven binary digits in a medium register do not correspond to a function in the decimal function code, the contents of that register are printed out as two (ten-bit) integers.

7.4.8 Key 6 Post mortem integers

As key 5 except that the contents of the page or sector are interpreted as 10-bit integers. Two integers are printed to a line, each preceded by "=".

7.4.9 Key 7 Post mortem floating point numbers

As key 5 except that the contents of the page or sector are interpreted as floating point numbers. Numbers are printed in floating decimal form, an argument followed by a comma and a decimal exponent. An * indicates that the following number is in the store in an unstandardised form. Numbers

are normally printed four to a line, but if a 40-bit word is encountered whose "exponent" is outside the permissible range -256 to +255, it is printed as four short integers on the left hand side of the page as for key 6, and so spoils the lay-out.

7.4.10 Key 8 Restart

This can also be used to re-enter a programme stored in the backing store. Key 8 is returned to zero and the FIRSTSECTOR is indicated by tapping a three-decimal-digit number. The title is first printed and the programme entered using the entry cue stored on the FIRSTSECTOR so that the labels list is not required.

7.4.11 Key 9 Sector enter

The sector indicated by the number (in binary) on the remaining nine keys is read to page 0 and entered at line 63. An example is that with key 9 and key 0 up Binary input (stored on sector 1) is entered.

7.4.12 Key 9 Entry to Tele-output

Tele-output as described in section 7.4.5 uses sectors 0, 2 and 3. However, the part of its programme which is on sector 0 can be used by itself to give a simipler form of tele-output which is available when sectors 2 and 3 have been overwritten by other programme.

For this mode of entry, sector 0 is entered by Key 9, so that key 9 only is set and the Initial Transfer Button pressed. The programme comes immediately to a 99 stop: the number of the first sector to be punched should be set up in binary on the handswitches and a prepulse given. The programme now comes to another 99 stop, and the last sector to be punched should be set up in the same way. On a prepulse, those sectors are punched out in binary and the programme comes to a third 99 stop. If more punching is required, tele-output should be re-entered by setting key 9 and pressing the Initial Transfer Button; if no more punching is required a prepulse should be given, when the hoot loop character is punched following by 1" of erases.

7.5 The Control Desk

For normal running the switches on the control desk (see diagram in appendix VII) are set as follows:-

write current on, to permit writing to the backing store

hoot on

inhibit parities off, to stop the computer if a parity check fails

manual/automatic to auto

handswitch stop off

single/continuous to continuous

handswitches all zero, for start clear

The function keys and address keys are irrelevant.

The programme tape is inserted in the reader, the intial transfer button is pressed and the single/continuous switch is turned to continuous, (if it was on single).

The computer may stop for the following reasons:-

(i) A 99 stop instruction is obeyed (or the other prepulsable stop instruction is obeyed).

(ii) A parity failure

(iii) Accumulator overflow

(iv) The optional stop. With the handswitch stop switch on, the computer stops when the instruction, in the register indicated by the setting (in binary) of the ten address keys, has been obeyed (see below for exceptions).

(v) A backing store read or write transfer is attempted with a non-existent sector, e.g. sector 512.

(vi) A loop stop, where an instruction transfers control to itself, e.g. 590 *

(vii) The single/continuous switch is turned to "single". This interrupts the control unit and individual instructions can be obeyed one at a time by pressing the prepulse button. In this way a programme stored in the computer can be obeyed but the time taken is at least 1,000 times the time taken on "continuous" so this is not recommended.

For stops (i) to (iv) the stop flip flop lights up, The computer will resume operation when the prepulse button is pressed. For stops (v) and (vi) the prepulse button is of no avail, but the computer will resume operation if the initial transfer button is pressed. On "single" most instructions require one prepulse (here defined as one pressing of the prepulse button); the 60, 62 and 63 instructions require two prepulses. Similarly with the optional stop on these instructions the computer stops twice. The optional stop should not be used on a 68 or 69 instruction. If it is used no drum transfer takes place (a modification of the computers is pending).

Instructions can be obeyed manually by turning the manual/autoswitch to manual. The 20 binary digits of the instruction are set up on the function and address keys and a prepulse given. Manual instructions are mainly used by the maintenance engineers.

# CHAPTER 8

## PROGRAMMING TECHNIQUES

### 8.1 Introduction

This chapter contains details of some coding tricks for Mercury and also some general strategy for writing programmes. These have evolved from several years of work by many experienced programmers and are intended to assist the beginner. It is not suggested that these are the only methods, for better methods can be found for individual programmes, and, of course, each of us prefers certain ways of doing certain things. Some of the procedures described are more or less standard on all similar computers.

### 8.2 Numerical methods for computers

Before actually coding up a problem it is advisable to consider the numerical methods available. Methods suitable for hand computing are not always satisfactory, e.g. relaxation. The time taken by the computer should first be estimated; an abnormally long calculation may indicate an unsuitable method rather than a large problem, and a certain amount of common sense is required. Many of the frequently used numerical methods are available as library routines, the Ferranti library for Mercury containing x-routines for Runge-Kutta, Gauss integration, interpolation, etc.

In this section Mercury is regarded as being principally a scientific computer. For commercial work and for data processing additional equipment for input, output and backing store vastly increases the range of problems in these fields which can be easily tackled.

#### 8.2.1 Runge-Kutta for differential equations

A powerful method of calculating a function is to solve the ordinary differential equation which it satisfies. This is particularly useful when, as it usually happens, the explicit formula for the function is most unpleasant. Physical problems usually give rise to a differential equation; the explicit solution is frequently no use whatever for evaluating the solution numerically. The commonly used method of Runge-Kutta modified by Gill solves a set of simultaneous first order differential equations; a second order equation may be easily converted into two simultaneous first order equations, and similarly an nth order equation is solved as n simultaneous first order equations. The error for each step is $0(h^5)$ where h is the step length, so care must be taken with the choice of step length and the solution can be checked by repeating it with twice the step length. Variable-step methods also exist. The direction of integration is sometimes important e.g. when solving an equation to find the solution which $\sim e^{-x}$ for large x it is useless to proceed far in the direction of x increasing if another (unwanted) solution $\sim e^{+x}$. Small errors introduced in the first few steps would eventually exceed the desired solution. If the boundary conditions are not all given at the same point, several solutions may be found by starting with arbitrary conditions at one point and the required solution obtained by interpolation, or by adjusting starting conditions at both ends so that the solutions meet in the middle.

If for the equations

$$\frac{dy_1}{dx} = f_1(x, y_1, y_2)$$

$$\frac{dy_2}{dx} = f_2(x, y_1, y_2)$$

the value of $x$ is required when $y_2 = b$ say, then the last step can be performed with the equations

$$\frac{dy_1}{dy_2} = \frac{f_1}{f_2} \qquad \frac{dx}{dy_2} = \frac{1}{f_2}$$

where the independent variable is changed to $y_2$, and with the step length $b - y_2$. For programming this, it is simplest to solve

$$\frac{dy_1}{dx} = f_1, \quad \frac{dy_2}{dx} = f_2 \qquad \text{and} \qquad \frac{dx}{dx} = 1$$

and then for the last step solve

$$\frac{dy_1}{dy_2} = \frac{f_1}{f_2} \qquad \frac{dy_2}{dy_2} = 1 \qquad \frac{dx}{dy_2} = \frac{1}{f_2}$$

Other methods of solving differential equations are the Adams Bashforth method and of course by Taylor series, the latter being particularly useful for simple equations with polynomial coefficients.

### 8.2.2 Matrices

It should be remembered that the inversion of a matrix is more difficult than the solution of a set of linear equations. Both are usually done by pivotal condensation always using the largest remaining element for the pivot. Iterative methods are often prohibitively slow.

### 8.2.3 Integration or quadrature

The Gauss-type methods are recommended for evaluating a definite integral. The results may be checked by using methods of different orders. Simpson's rule may only occasionally be usefully employed.

### 8.2.4 Integral equations

These are often best solved by first reducing them to simultaneous linear equations. Iterative methods, even if known to be convergent, usually take considerably longer.

### 8.2.5 Partial differential equations

For elliptic equations the straightforward method of relaxation is not suitable for a computer and the Liebmann process, in which all the points are relaxed in turn, is superior. Improvements on the Liebmann process have recently been discovered e.g. sweeping alternately by rows and columns. For parabolic and hyperbolic equations finite difference methods can be used (perhaps involving matrix inversions) or for the latter the method of characteristics.

### 8.2.6 Series expansions

A function $f(x)$ can be expressed as

$$f(x) = \sum_{0}^{N-1} \frac{f^{n}(o)x^{n}}{n!} + R_N$$

if sufficient derivatives exist and are continuous. However this truncated Taylor series is not the best approximation which can be used for calculating $f(x)$. If $f(x)$ is expanded in Chebychev polynomials (perhaps with minor variations) the power series obtained can be shown to have a smaller error over any given range $|x| < a$ than any other

power series of the same degree. Thus if the desired accuracy is obtained by N terms of a Taylor series, the same accuracy can usually be obtained with a Chebychev series of fewer terms. The Chebychev coefficients can be calculated or may be available in tables. Many of the quickies use such methods.

8.2.7 Slowly convergent series

To find $\sum_{1}^{\infty} \frac{1}{n^{3/2}}$ by evaluating and adding successive

terms is far too slow and would take several hours on Mercury, even though the series is convergent. The Euler-Maclaurin summation formula or other formulae may sometimes be used to speed up the convergence.

8.2.8 Interpolation

Linear interpolation is seldom adequate. Formulae for higher order interpolation can only be used if the function is suitably continuous. If there are discontinuities in the function the higher the order of the interpolation the less accurate the results and one may be forced to use linear interpolation. A library routine exists for inverse interpolation, viz. for finding $x$ such that $f(x) = 0$, and this can be used for finding the zeros of a polynomial and the roots of a transcendental equation.

8.2.9 References

Gill, S. (1951) Proc. Camb. Phil. Soc., 47, pp 96 - 108

Hartree, D.R. (1955) "Numerical Analysis" Oxford Univ. Press

Hildebrand, F.B. (1956) "Introduction to Numerical Analysis" McGraw Hill

Lanczos, C. (1957) "Applied Analysis" Pitman.

8.3 Use of B-registers

Since B7 can be used as Sac certain operations can be done on it which cannot be done on other B-registers. Sac should consequently be used for short jobs whenever possible but it is not suitable for long term work and its contents are destroyed by all the quickies and also during chapter changing. If a second B-register is required for a short job B6 should be used and so on working down through the B-registers. B1 is conventionally used for closing subroutines. It is rarely possible to have a new B-register for every count in the programme so it is necessary to use some B-registers again and again.

8.4 Subroutines

Just as with subchapters, when a particular group of instructions are required to be obeyed at two or more places in the same chapter, it is advisable to write them down once only terminated with the instruction

```
591  0
```

At each of the places where they are required the two instructions

```
101  2*        Set  B1  to the return address
590            Jump to the subroutine
```

will call in the subroutine. When the instructions of the subroutine have been obeyed the closing instruction returns control to the master routine. If the subroutine itself call in another subroutine the first can begin with

```
011  0+v1
```

which plants the contents of B1 on entry in the address part of the closing instruction

```
590  0    (1
```

which will then return control to the correct place in the master. Except for short subr utines it is in fact an excellent habit always to begin by planting B1 as above. B1 can then also be used for short jobs within the subroutine.

Since quickies are used frequently they are often used twice in a chapter and many programmers assign the routine number the quicky number plus 900 and close the quicky. e.g.

```
R 901
Q1
591  0
```

With Q9, B1 must first be set by 101 * and the contents of B1 are then used to pick up the programme parameters m and n in the next but one register. Q9 is therefore used closed as

```
R 909
Q9
591  3
```

In connection with Q1 it will be noticed that on Mercury the time taken to perform a division is roughly ten times that for a multiplication. The number of divisions should be kept as few as possible e.g. if both $\frac{1}{x}$ and $\frac{1}{x^2}$ are required then $\frac{1}{x}$ may be first found using Q1 and $\frac{1}{x^2}$ found by multiplying $\frac{1}{x}$ by itself.

## 8.5 Cycles

Often an operation represented by a group of instructions, some perhaps B-modified, is required to be done say p t me . A count is is first set in a B-register and one of the B-test instructions used. Some ways this counting can be done are represented diagramatically by

| | | | |
|---|---|---|---|
| 106 p | 106 p - 1 | 106 1 | 106 1 - p |
| Operation (1 | Operation (1 | Operation (1 | Operation (1 |
| 136 1 | 136 1 | 176 p | 186 v1 |
| 080 v1 | 090 v1 | 186 v1 | |

The last two using the 18 instruction, are recommended, and for the fourth it is of course necessary that the operation does not change Bt. If Bt is changed by the operation the instruction 176 0 can be included before the 18 instruction. e.g to add long numbers in consecutive long registers the first of which is labelled 1 and the last labelled 2 use

```
400  0        Clear  A
106  v1n2     B6' = 1-p  if there are  p  numbers
426  v2       Add a number
186  -1*      Repeat
```

The corresponding programme for adding short integers in Sac is

```
300  0
106  v1n2
226  v2
186  -1*
```

## 8.6 Useful coding tricks

The following are some coding tricks which have come to the author's notice and many of them are used in the quickies and in the Input Routine itself. It is usually difficult to name the original author as there may be several and also a minor variation may or may not be regarded as changing the "ownership". Some of these are useful coding but others are perhaps just tricks.

### 8.6.1 Change the sign of the accumulator if it is negative i.e. A' = |A|

```
490  2*       Jump if  A > 0
520  2        Multiply by  -1
```

Note that multiplying negatively by plus one or by using rounded multiplication, minus the contents of the accumulator will not in general be exact.

### 8.6.2 Change the sign of B6. i.e. B6' = -B6

```
166  -1       Non-equivalent with -1
126  1
```

(If B6 is -512 this gives B6' = -512)

### 8.6.3 Double Sac i.e. S' = 2S

```
327  0
```

### 8.6.4 Multiply Sac by ten i.e. S' = 10S

```
210  2+*      Store Sac in the address of the third
              instruction
327  0        Sac  x  2
327  0        Sac  x  4  +  Sac
327  0        Sac  x  10
```

It is usually unnecessary to store short integers anywhere other than in the address part of instructions.

### 8.6.5 Put -x1 to Sac

```
337  x1
```

### 8.6.6 Put -x1 to St without changing Sac

```
377  x1
```

8.6.7 Put ±x1 to Bt without changing any B-register

100 x1 or 170 x1

Although the attempted operation on B0 cannot really be done and always leaves zero in B0, the B-test register is set as with normal B-registers. Similarly the instructions 000 and 070 are sometimes useful.

8.6.8 Halve B6 signed i.e. $B6' = \frac{1}{2}B6_{\pm}$

126 512

146 256

8.6.9 Put n in B6 where $B5 = 2^n$

| | | |
|---|---|---|
| 106 | -1 | $B6' = -1$ |
| 145 | 0 | $B5' = \frac{1}{2}B5$ |
| 186 | -1* | Jump if $B5 \neq 0$; add one to B6 |

(0 is put in B6 if $B5 = 0$)

8.6.10 Eriksen converter

For a five digit character in B6, change the most significant digit if the character has an even number of ones in the least significant four digits.

| | | |
|---|---|---|
| 306 | 16 | $S' = B6 + 16$ |
| 166 | 16 | Change the most significant digit |
| 357 | -1 | Remove the least significant digit of Sac |
| 280 | -2* | Repeat if Sac $\neq 0$ |

By this the copy of a tape character representing a decimal digit is converted to the binary form of that digit, the other figure shift characters become integers in a sensible order.

8.6.11 Pick up into Sac the programme parameter n after the instructions 101 *, 590 v/2, = n

| | | |
|---|---|---|
| R 2 | | |
| 011 | 2+* | Store B1 |
| 006 | 1+* | $B6' = 2\ B1$ |
| 126 | 0 | |
| 206 | 8.2 | $S' = $ [illegible] if $4.0 < B1 < 12.0$ |
| 171 | 4.0 | Jump if B1 is as above |
| 090 | 2* | |
| 206 | 2 | $S' = n$ if $0 \leq B1 < 4.0$ or $12.0 \leq B1 \leq 15.63$ |

8.6.12 Convert the short integer in Sac into a floating-point long number

(i) Sac signed or on the plus-minus convention $-512 \leq S \leq 511$

```
+) = 9, = 0, = 0, = 0  (1
210  v1          Store Sac
400  v1          Transfer to A  (not yet standardised)
440  0           Standardise  A  by adding zero
```

(ii) Sac unsigned or on the plus convention $0 \leq S \leq 1023$

```
+) = 19, = 0, = 0(1, = 0
210  v1
400  v1
440  0
```

8.6.13 Convert the long number in the accumulator into a short integer

```
+) = 29, = 0, = 0, = 640  (1
440  v1          This gives a long number with exponent  29
                 so that the required short integer is in the
                 least significant ten digits of the
                 fractional part
410  32
200  32+         Integer to Sac
```

If the accumulator contains a positive number which is an integer greater than 1023 but less than $2^{30}$ the result is the integer modulo 1024.

8.6.14 Find the integral part of A

```
+) = 29,  = 0, = 0, = 0     (1
440  v1
```

8.6.15 Find the fractional part of A

```
+) = 29,= 0, = 0, = 0     (1
410  32          Store  A = x
440  v1          A' = [x]
520  2           A' = - [x]
440  32          A' = the fractional part of  x
```

8.6.16 Multiply two short integers together i.e. $B4' = B5_{\bullet} \times B6_{\bullet}$ (B5 and B6 unsigned)

```
+) = 0, = 0(1,  = 0, = 0
015  v1   }
400  v1   }      B5  to the least significant ten digits
             }   of the fractional part of  A
016  v1
540  v1          This least significant half multiplication
410  32          instruction is unstandardized.
004  32+
```

8.6.17 Store Bt in Sac i.e. S = 1 if Bt > 0 , S = 0 if Bt = 0, S = -1 if Bt < 0

```
300  -1        S' = -1
090  2*        Jump if  Bt > 0
590  2*        Jump
187  *         Add  1  to  B7 (obeyed twice if  Bt > 0)
```

The reverse operation of putting Sac into Bt is

```
210  14*
100  0
```

8.6.18 Forward reference to preset parameter

A forward reference to a preset parameter is effectively obtained by

```
300  v1/99
```

and then later

```
R 99
v1 = x1
```

8.6.19 Clutched count

Calculates decimal digit by repeated subtraction of a power of 10 and leaves decimal digit in teleprinter code in B-line. Junk is left in the most significant five bits of the B-line.

```
105  543
125  177
450  v1        subtract power of 10
490  -2*
125  464       adjust "16" hole of character
090  -1*
625  0         punch digit
440  v1        restore  A > 0
```

8.7 Layout of results

Finally, a little effort in laying out results is well rewarded. The following may be included:-

(i) A heading, and a page number

(ii) Several numbers per line

(iii) An extra line feed every fifth line

(iv) Several extra line feeds at the end of a page.

# APPENDIX I

## DETAILS OF THE ACCUMULATOR ARITHMETIC

### Add and Subtract

The 40 and 41 instructions copy to and from the accumulator exactly; there is no standardisation, round-off, or test for accumulator overflow.

The 42 instruction adds a long number from the computing store to the long number in the accumulator. The long number from the computing store is first taken into the arithmetic unit. The exponent of this number is compared with the exponent of the contents of the accumulator, and the argument of the number with the smaller exponent is shifted down by a number of places equal to the difference between the exponents, copies of the sign digit being fed in from the left. The two arguments are then added as fixed-point numbers*, and the result of this sum is rounded off by putting a one in the least significant digit position, whether or not there was a one there before. This rounded argument is taken into the accumulator with the larger exponent and the resulting floating-point number is standardised so that the argument is in the range

$$\tfrac{1}{2} \leq x < 1$$

or $$-\tfrac{1}{2} > x \geq -1,$$

by shifting the argument and adjusting the exponent appropriately. If this process involves a shift up, zeros are fed in below the round-off digit; if it involves a shift down the argument is again rounded since the original round-off digit is lost. To allow for this, the fractional part of the accumulator is provided with an extra digit position beyond that which is usually thought of as the most significant, but the programmer does not have access to this digit. In most cases the two numbers to be added are both in standardised form; if not the addition is still as above and some significant digits may be lost.

The 43 instruction is similar to the 42 instruction except that it subtracts instead of adds. The stages are:-

Negate the store operand

Shift down one argument a number of places equal to the difference of the exponents

Add fixed-point

Round-off

Standardise, rounding again if this involves a shift down

The 44 instruction is the same as the 42 instruction except that there is no round-off. The stages are:-

Shift down one argument a number of places equal to the difference of the exponents

Add fixed-point

Standardise

The 45 instruction is the same as the 43 instruction except that there is no round-off.

---

* Note: if the difference between the exponents is 48 or greater, no addition is done, and the larger number is taken as the answer.

The 46 instruction adds a long number from the computing store to the long number in the accumulator without either round-off or standardisation. The stages are :-

shift down one argument a number of places equal to the difference of the exponents

add fixed-point

shift down one place, without rounding, and increase the exponent by one. This extra shift is to prevent the argument of the result being too large.

The 47 instruction does a subtraction in a similar way, the answer being unrounded and unstandardised.

Example:

The 44 instruction, unrounded addition,

$A' = A + L$

$A = 1\frac{7}{8}$

$L = 3$

let xA, yA; xL, yL denote the arguments and exponent of the numbers in the accumulator and computing store register respectively.

Then xA = 0. 11110 ..... 0, yA = 0000000001

xL = 0. 11000 ..... 0, yL = 0000000010

(i) shift the argument of the number with the smaller exponent:

xA' = 0.011110 ..... 0

(ii) add the argument fixed point:

xA' = (0)1.001110 ....0, yA' = 0000000010

(iii) standardise:

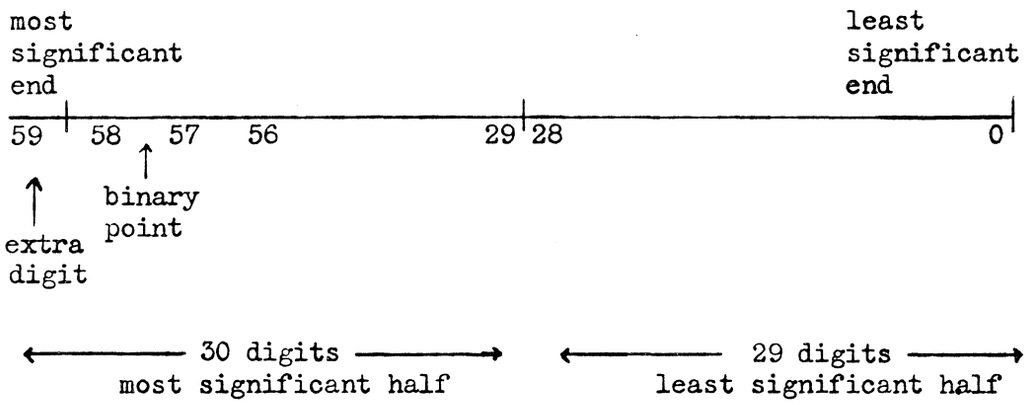
xA' = 0.1001110 ......0, yA' = 0000000011

which is the standardised floating-point representation $4\frac{7}{8}$.

Multiplication

The product of two thirty-digit fixed-point numbers is a sixty-digit fixed-point number. The instructions 50 51 52 and 53 give the thirty most significant digits (excluding the most significant digit itself) and the 54 and 55 instruction give the least significant 29 digits as in figure 2.

Figure 2




The 50 instruction for the product of the long number in the computing store gives the most significant half of the product with the 59 digit going into the accumulator's extra digit position and with exponent equal to the sum of the two exponents. This is rounded-off then standardised, and if standardising involves a shift down it is rounded again. The 51 instruction for negative multiplication begins with a sixty digit negative product but is otherwise the same as the 50 instruction. The stages are :-

take the most significant half

round-off

standardise, rounding again if this involves a shift down

The 52 instruction is the same as the 50 instruction except that there is no rounding-off. The stages are :-

take the most significant half

standardise

The 53 instruction begins with the sixty digit negative product but is otherwise the same as the 52 instruction. The 54 instruction gives the least significant 29 digits of the product and a 0 in the most significant digit position. The exponent is the sum of the exponents of the two numbers minus 29 and the result is neither rounded-off nor standardised. The 55 instruction gives the least significant 29 digits of the negative product with 0 in the most significant digit position and with exponent the sum of the two exponents minus 29. The result is neither rounded-off nor standardised. Thus the 54 and 55 instructions always give a number which is greater than or equal to zero.

Example:

51 instruction,

$A' = -AL$

when $A = 2\frac{1}{4}$

$L = \frac{1}{2}$

i.e. $xA = 0.10010 \ldots\ldots\ldots 0$ $\quad yA = 0000000010$

$x = 0.10 \quad \ldots\ldots\ldots 0$ $\quad y = 0000000000$

First stage : take the most significant half :-

$yA' = yA + y = 0000000010$

60 digit product = -00.010010 ..... 0

= +11.101110 ..... 0

xA' = 1.101110 ..... 0

Second stage: round-off :-

xA' = 1.101110 .....01 yA' = 0000000010

Third stage: standardise :-

xA' = 1.01110 ..... 010 yA' = 0000000001

i.e. $A' = 1\frac{1}{8} + \frac{1}{2^{-28}}$

# APPENDIX II

## STRUCTURE OF CUES, LABELS, ETC.

### Cues

The directives ACROSS and DOWN become on input

```
+)300  4*
590    4
```

followed by a long register storing the cue for the next chapter as follows:-

H0 = first sector minus first page

H0+ = first page. The 512 digit is 0 if ACROSS, 1 if DOWN

H1 = last page

H1+ = entry register

The directive UP becomes the two instructions

```
300  1
590  4
```

### Labels list

While a routine is being read in, its labels are stored in the computing store with label n in L(27.54 + 2n). At the end of the routine, indicated by another directive R, C, etc., the labels are packed in the order of increasing n into consecutive long registers on sectors 64 to 95 immediately after those of the previous routine. Each label is composed as follows :-

H0 = routine number

H0+ = label number

H1 = medium register address of item labelled

H1+ = item. Digits 0 - 4 address overflow
digits 5 - 8 32 × item
digit 9 0 for left half register, 1 for right half

### Routine list

The routine list on sectors 96 to 111 has a 20 digit entry in the nth medium register corresponding to routine n as follows:-

H0 = long address of v0 in labels list (0 to 1023)

H0+ = chapter number

### Chapter list

The chapter list on sectors 112 and 113 has a 20 digit entry in the nth medium register corresponding to chapter n as follows:-

H0 = first sector

H0+ = 32 × first page + last page

During the chapter the entry in H0+ is the first page.

Preset parameter list

The values of the preset parameters are kept in the computing store, the entry in L(24.44 + 2n) corresponding to preset parameter n, the entry for n = 0 being the x0 of x-routines. These are stored in the same way as v-labels, address in the second half of the computing store are permitted.

Reference list

v-floating addresses are filled in by the Input Programme as follows:-

(i) Backward references to a previous label in the present routine or a label in a previous routine are filled in immediately.

(ii) Forward references to labels in routine r are stored in the reference list and are filled in at the end of routine r.

(iii) References for cues are also stored in the reference list (type 6). These are not filled in as above, but are filled in at the end of the second of the two chapters involved. References not filled in are stored in long registers from L16.0 to L24.42 as follows:-

H0 = routine number

H0+ = 8 × label number + type

H1 = short address. The 512 digit is 0 if v, 1 if n

H1+ = sector

Entries in page 0 during input

| | |
|---|---|
| H56 | query number |
| H56+ | A,B,etc., for the query print |
| H57 | predicted last page |
| H57+ | FIRSTSECTOR |
| H58 | asterisk print. -1 if no asterisk, 0 if asterisk |
| H58+ | highest page so far |
| H59 | next position in labels list (long address) |
| H59+ | sector in page 31 |
| H60+ | next position in reference list (long address) |
| H61 | i present item |
| H61+ | h next half register in page 1 (short address) |
| H62 | p page being assembled |
| H62+ | s sector being assembled |
| H63 | r current routine |
| H63+ | c current chapter |

Sectors 0 to 5

With only sectors 0 - 5 of the Input Routine intact, the following starting procedures are available:

Key 2 for binary input

Key 3 for binary output

Key 8 for restart

Key 9 for sector enter.

These sectors include the Chapter Changing Sequence and the Error Print Sequence, so that a programme can be run using only these sectors.

# APPENDIX III

## Details of Quickies

| Quicky Number | Specification | Store Registers | Working Space | B-line Used | Notes |
|---|---|---|---|---|---|
| 1 | $A' = \frac{1}{A}$ | 24 | 32 - 35 | Sac<br>Bt St | A must be standardised<br>Error print if $\|A\| < 2^{-253}$ |
| 2 | $A' = \frac{1}{\sqrt{A}}$ | 33 | 32 - 35 | Sac<br>Bt St | A must be standardised<br>Error print if A < 0 |
| 4 | $A' = e^A$ | 47 | 32 - 35 | Sac St | Error print if $e^A > 2^{256}$ |
| 5 | $A' = \tan A$ | 62 | 32 - 37 | Sac St | |
| 6 | $A' = \sin A$ or $\cos A$ | 39 | 32 - 37 | Sac St | Enter at 2nd instruction for cos A |
| 7 | $A' = \cos A$ | 37 | 32 - 37 | Sac St | |
| * 8 | Punch Sac ± | 32 | | Sac St<br>Bt | Prints Sac as an integer in the range -512 to +511 |
| * 9 | Punch Acc fixed point | 80 | 32 - 35 | Sac St<br>Bt | Enter with 101 * 590 - = m, = n where m and n are the number of digits before and after the decimal point |
| * 10 | Punch Acc floating point | 102 | 32 - 35 | Sac St<br>Bt | Enter with Sac = number of decimal digits |
| * 11 | Punch Sac + | 34 | - | Sac St | Prints Sac as an unsigned integer in the range 0 - 1023 |
| 12 | $A' = \sqrt{A}$ | 35 | 32 - 37 | Sac St<br>Bt | Error print if A < 0 |
| 14 | $A' = \log_e A$ | 43 | 32 - 33 | Sac St | Error print if A ≤ 0 |
| 15 | $A' = \arctan y/x$<br>y = L 32,<br>x = L34 | 74 | 32 - 37 | Sac St<br>Bt | Error print if 0/0<br>Range 0 - $2\pi$ |
| 16 | $A' = \arcsin A$ | 47 | 32 - 35 | Sac St<br>Bt | Error print if $\|A\| > 1$<br>Range $-\frac{\pi}{2} + \frac{\pi}{2}$ |
| 18 | Read integer to Sac | 38 | - | Sac St<br>Bt | Reads short integers beginning with + - or decimal digit and terminating with CR or Sp. Ignores FS LF ER and also CR and Sp between numbers |

See List CS 202A for up-to-date details of Quickies.

| Quicky Number | Specification | Store Registers | Working Space | B-line Used | Notes |
|---|---|---|---|---|---|
| 19 | Read fixed or floating point decimal number to Acc | 150 | 32 - 36 | Sac St Bt | Reads numbers in form:- sign integral part decimal point fractional part comma sign exponent CR LF or Sp Sp<br>The signs, the integral part, the decimal point and fractional part, the comma and the exponent may be omitted when not required.<br>FS ER Sp(single) are ignored, also CR LF Sp when between numbers. |

* Note 1: In Quickies 8, 9, 10 and 11 each number is preceded by FS CR LF CR and terminated by Sp Sp. To print on the same line enter at the fifth instruction.

2: Alterations are at present being made to the Quicky programmes which will usually reduce slightly the number of store registers used.

# APPENDIX IV

## LIST OF FAULTS

1. Address underflow
2. Address overflow
3. Spurious character, (including Chapter directive in Interlude)
4. Label set twice
5. Label not set
6. Preset parameter not set
7. Too many references
8. Overflow page 15, interlude too long
9. Overflow predicted page
10. Title too long
11. Wrong quicky number
12. Too many labels
13. Chapter numbers, parameter numbers, or label numbers over 100.
14. Query number too large.

## 5.1 TABLE OF FUNCTION CODES

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | B′ = Bt′ = H | H′ = B | B′ = Bt′ = B + H | B′ = Bt′ = B − H | B′ = Bt′ = ½B − H | B′ = Bt′ = B & H | B′ = Bt′ = B ≠ H | Bt′ = B − H | Bt ≠ o, C′ = n | Bt ≥ o, C′ = n |
| 1 | B′ = Bt′ = n | | B′ = Bt′ = B + n | B′ = Bt′ = B − n | B′ = Bt′ = ½B − n | B′ = Bt′ = B & n | B′ = Bt′ = B ≠ n | Bt′ = B − n | Bt ≠ o, C′ = n; B′ = Bt′ = B+1 | |
| 2 | S′ = St′ = H | H′ = S | S′ = St′ = S + H | S′ = St′ = S − H | S′ = St′ = ½S − H | S′ = St′ = S & H | S′ = St′ = S ≠ H | St′ = S − H | St ≠ o, C′ = n | St ≥ o, C′ = n |
| 3 | S′ = St′ = n | | S′ = St′ = S + n | S′ = St′ = S − n | S′ = St′ = ½S − n | S′ = St′ = S & n | S′ = St′ = S ≠ n | St′ = S − n | St ≠ o, C′ = n; S′ = St′ = S+1 | |
| 4 | A′ = L | L′ = A | A′ = A + L | A′ = A − L | A′ = A + L UNROUNDED | A′ = A − L | A′ = A + L UNROUNDED AND UNSTANDARDISED | A′ = A − L | SHIFT ≤ 31, C′ = n | A ≥ o, C′ = n |
| 5 | A′ = A × L | A′ = − A×L | A′ = A × L UNROUNDED | A′ = −A×L | A′ = A × L LEAST-SIG HALF | A′ = A × L | G′ = L | D U M M Y | H O O T | C′ = n |
| 6 | H′ = $t_i$ | H′ = hs | $t_o$′ = n | $t_o$′ = H | Display′ = L | O P E N SHUTTER | C L O S E SHUTTER | T′ = n | P′ = D | D′ = P |
| 7 | B′ = Bt′ = E+n | E′ = B | B′ = Bt′ = B + E + n | B′ = Bt′ = B−(E + n) | B′ = Bt′ = ½B−(E + n) | B′ = Bt′ = B & (E + n) | B′ = Bt′ = B ≠ (E+n) | Bt′ = B−(E+n) | A′ = A+0.$2^n$ unrounded, unstandardised with one place right shift. | |
| 8 | CONDITIONING | READ CARD | PUNCH CARD or PRINT LINE | PAPER THROW | | | MAG− OPERATE | MAG−SELECT | TC1 BUSY, C′ = n | TC2 BUSY, C′ = n |
| 9 | S′ = St′ = I + n | 0′ = S St′ = [illegible] | S′ = St′ = S+(I + n) | S′ = St′ = S−(I +n) | S′ = St′ = ½ S−(I+n) | S′ = St′ = S&(I + n) | S′ = St′ = S ≠ (I+n) | St′ = S−(I+n) | | S T O P |

NOTES: 1) A function is defined by a ROW digit followed by a COLUMN digit. e.g. 59 denotes C′ = n

2) Some codes are unassigned. All these are at present stop functions.

3) Shaded squares represent function codes which are not B-modificable.

## 5.2 Table of function codes. Decimal/Octal

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 00 | 051 | 20 | 071 | 40 | 041 | 60 | 002 | 80 | 126 |
| 01 | 042 | 21 | 062 | 41 | 043 | 61 | 022 | 81 | 124 |
| 02 | 050 | 22 | 070 | 42 | 044 | 62 | 125 | 82 | 105 |
| 03 | 055 | 23 | 075 | 43 | 045 | 63 | 025 | 83 | 104 |
| 04 | 054 | 24 | 074 | 44 | 046 | 64 | 063 | 84 | 100 |
| 05 | 053 | 25 | 073 | 45 | 047 | 65 | 164 | 85 | 100 |
| 06 | 052 | 26 | 072 | 46 | 146 | 66 | 165 | 86 | 161 |
| 07 | 056 | 27 | 076 | 47 | 147 | 67 | 004 | 87 | 160 |
| 08 | 007 | 28 | 027 | 48 | 061 | 68 | 024 | 88 | 121 |
| 09 | 020 | 29 | 040 | 49 | 021 | 69 | 005 | 89 | 120 |
| 10 | 011 | 30 | 031 | 50 | 064 | 70 | 111 | 90 | 131 |
| 11 | 100 | 31 | 100 | 51 | 065 | 71 | 023 | 91 | 137 |
| 12 | 010 | 32 | 030 | 52 | 066 | 72 | 110 | 92 | 130 |
| 13 | 015 | 33 | 035 | 53 | 067 | 73 | 115 | 93 | 135 |
| 14 | 014 | 34 | 034 | 54 | 026 | 74 | 114 | 94 | 134 |
| 15 | 013 | 35 | 033 | 55 | 006 | 75 | 113 | 95 | 133 |
| 16 | 012 | 36 | 032 | 56 | 143 | 76 | 112 | 96 | 132 |
| 17 | 016 | 37 | 036 | 57 | 141 | 77 | 116 | 97 | 136 |
| 18 | 017 | 38 | 037 | 58 | 060 | 78 | 144 | 98 | 100 |
| 19 | 100 | 39 | 100 | 59 | 001 | 79 | 100 | 99 | 000 |

In the octal codes the seven binary function digits are grouped one three three, e.g.

40 (A = L) is 041 i.e. 0 100 001

## 5.3 Table of function codes. Octal/Decimal

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 000 | 99 | 040 | 29 | 100 | | 140 | |
| 001 | 59 | 041 | 40 | 101 | | 141 | 57 |
| 002 | 60 | 042 | 01 | 102 | 60* | 142 | 01* |
| 003 | | 043 | 41 | 103 | | 143 | 56 |
| 004 | 67 | 044 | 42 | 104 | 83 | 144 | 78 |
| 005 | 69 | 045 | 43 | 105 | 82 | 145 | |
| 006 | 55 | 046 | 44 | 106 | | 146 | 46 |
| 007 | 08 | 047 | 45 | 107 | | 147 | 47 |
| 010 | 12 | 050 | 02 | 110 | 72 | 150 | 02* |
| 011 | 10 | 051 | 00 | 111 | 70 | 151 | 00* |
| 012 | 16 | 052 | 06 | 112 | 76 | 152 | 06* |
| 013 | 15 | 053 | 05 | 113 | 75 | 153 | 05* |
| 014 | 14 | 054 | 04 | 114 | 74 | 154 | 04* |
| 015 | 13 | 055 | 03 | 115 | 73 | 155 | 03* |
| 016 | 17 | 056 | 07 | 116 | 77 | 156 | 07* |
| 017 | 18 | 057 | | 117 | | 157 | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 020 | 09 | 060 | 58 | 120 | 89 | 160 | 87 |
| 021 | 49 | 061 | 48 | 121 | 88 | 161 | 86 |
| 022 | 61 | 062 | 21 | 122 | 61* | 162 | 21* |
| 023 | 71 | 063 | 64 | 123 | | 163 | |
| 024 | 68 | 064 | 50 | 124 | 81 | 164 | 65 |
| 025 | 63 | 065 | 51 | 125 | 62 | 165 | 66 |
| 026 | 54 | 066 | 52 | 126 | 80 | 166 | |
| 027 | 28 | 067 | 53 | 127 | | 167 | |
| 030 | 32 | 070 | 22 | 130 | 92 | 170 | 22* |
| 031 | 30 | 071 | 20 | 131 | 90 | 171 | 20* |
| 032 | 36 | 072 | 26 | 132 | 96 | 172 | 26* |
| 033 | 35 | 073 | 25 | 133 | 95 | 173 | 25* |
| 034 | 34 | 074 | 24 | 134 | 94 | 174 | 24* |
| 035 | 33 | 075 | 23 | 135 | 93 | 175 | 23* |
| 036 | 37 | 076 | 27 | 136 | 97 | 176 | 27* |
| 037 | 38 | 077 | | 137 | 91 | 177 | |

* Refer to addresses in the second quarter of the computing store.

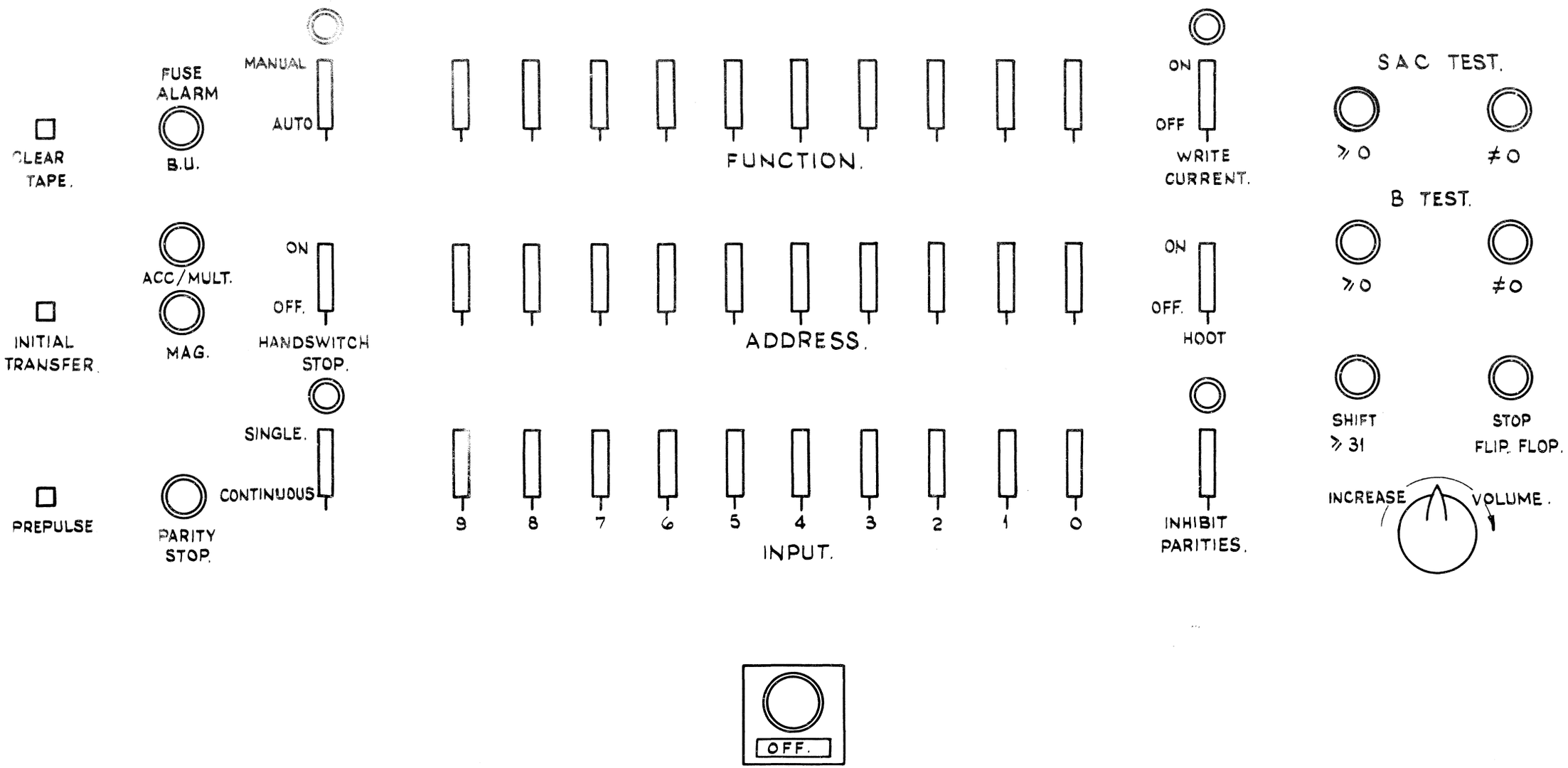
# APPENDIX VI

## TAPE CODES

| TAPE | VALUE | PRINTER | |
|---|---|---|---|
| | | FIGS. | LETS. |
| . | 0 | FIG. | SHIFT. |
| . ● | 1 | 1 | A |
| .● | 2 | 2 | B |
| .●● | 3 | * | C |
| ●. | 4 | 4 | D |
| ●. ● | 5 | ( | E |
| ●.● | 6 | ) | F |
| ●.●● | 7 | 7 | G |
| ● . | 8 | 8 | H |
| ● . ● | 9 | ≠ | I |
| ● .● | 10 | = | J |
| ● .●● | 11 | - | K |
| ●●. | 12 | v | L |
| ●●. ● | 13 | LF | M |
| ●●.● | 14 | SP | N |
| ●●.●● | 15 | , | O |
| ● . | 16 | 0 | P |
| ● . ● | 17 | > | Q |
| ● .● | 18 | > | R |
| ● .●● | 19 | 3 | S |
| ● ●. | 20 | → | T |
| ● ●. ● | 21 | 5 | U |
| ● ●.● | 22 | 6 | V |
| ● ●.●● | 23 | / | W |
| ●● . | 24 | x | X |
| ●● . ● | 25 | 9 | Y |
| ●● .● | 26 | + | Z |
| ●● .●● | 27 | LET. | SHIFT. |
| ●●●. | 28 | . | . |
| ●●●. ● | 29 | n | ? |
| ●●●.● | 30 | CR | £ |
| ●●●.●● | 31 | ER | ER |

# APPENDIX VII

## MERCURY CONTROL PANEL.



# INDEX