Ferranti

SIRIUS COMPUTER

PROGRAMMING MANUAL

LIST CS 244A

The FERRANTI SIRIUS COMPUTER

10/-

CONTENTS

SIRIUS PROGRAMMING MANUAL

CHAPTER 1

SIRIUS

1.1 INTRODUCTION

1.1.1 Sirius is a small decimal computer. In its basic form it has a store of 20 nickel delay lines, each consisting of 50 locations, giving a total capacity of 1000 locations. There are also 9 short delay lines each consisting of one word. Information is put into the computer and extracted from it by means of 5-channel punched paper tape.

1.1.2 Each location in the computer, whether in the main store or the accumulators, holds ten decimal digits. These digits will represent either a number or a computer instruction, and either of these can be called a computer word.

1.2 NUMBERS WITHIN THE COMPUTER

1.2.1 Since each location can hold ten digits the largest number which can be put in any location is 9999999999, equal to $10^{10} - 1$. If 0 is included then all numbers between 0 and $10^{10} - 1$, that is 10^{10} numbers, can be expressed in Sirius.

1.2.2 In order to deal with the positive and negative numbers this range is divided equally. Positive numbers raise little difficulty and are held within the machine as written i.e. for x positive

$$0 \leqslant x \leqslant 4999999999$$

The convention adopted for negative numbers is to hold them in the machine as the complement with respect to 10^{10}, so that for example, -3 is held in the machine as $10^{10} - 3$, appearing in the machine as 9999999997. In other words the convention in Sirius is for the first digit of a number to include the sign. If the first digit of x^* (the number as held in the machine) is anything from 0 to 4 then x is a positive number of magnitude x^*. If the first digit of x^* is anything from 5 to 9 then x is a negative number of magnitude $10^{10} - x^*$. The following are examples of numbers and the form in which they are held in the machine:-

x : number	x^* : as held in Sirius
+123	0000000123
+900906	0000900906
-671	9999999329
-3960000000	6040000000
-2183200000	7816800000
-5000000000	5000000000

1.2.3 During the course of calculations numbers may arise which are outside the range permissible in the computer; this is said to be overflow and a special overflow indicator will be set. Steps can be taken to deal with this situation and will be considered later.

1.2.4 Note that, although the number -5000000000 can be represented, the number +5000000000 cannot.

1.3 ADDRESSES

1.3.1 As already mentioned, the working store of the basic machine has 1000 locations. Associated with each location are two quantities, the address of the location and the contents of the location. The addresses run 0, 1, 2... up to the end of the available store. The contents of the location is either a decimal number or instruction as

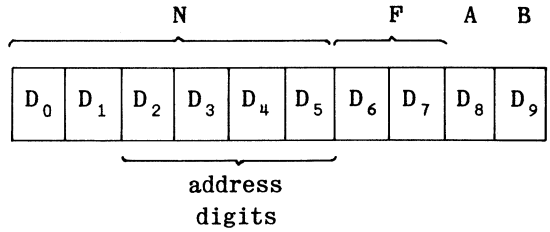
mentioned earlier, e.g. the location whose address is 603 may contain the digits 1246806103. The 9 accumulators are numbered 1 to 9.

1.4 THE CONTROL UNIT

There are only a limited number of types of operations or steps that a computer can perform. These can be carried out any number of times and in various ways but each calculation proceeds one step at a time according to the written programme. Th programme consists of a list of instructions or orders together with some data. The control unit of the computer selects instructions one by one, usually from consecuti locations, and as this is going on the address of the location containing the curren instruction is held in accumulator 1. The instruction itself is put into the order register by the control unit. As this instruction is carried out the address in accumulator 1, i.e. the control address, is increased by one before the current instruction is completed, so that for programming purposes the control address is th of the next instruction, i.e. the current instruction address plus 1. Care should b taken therefore not to use the accumulator 1 for ordinary arithmetic as this would change its contents and cause a transfer to some unwanted instruction. Note that, although only 4 digits of accumulator 1 at most, are needed for the control address, is in fact a full 10-digit register.

1.5 FORM OF INSTRUCTION

1.5.1 When the ten decimal digits are interpreted as an instruction they are treated as four parts as below:



The first six digits in the order are the N digits. They can specify an address in the store and so there could be up to 1,000,000 words of store. As in fact no Siriu will have as much as this some of the digits in the address will not be used; in th basic machine with a 1000 word store, for instance, only 3 of 6 N digits are used, so the largest address is 999. The first digit, D_0, is the most significant digit and the tenth, D_9, the least significant. These will often be abbreviated to m.s. and l.s. In some orders all six of the N digits are used but this will be explained later.

1.5.2 The digits D_6 and D_7 are called the function digits of the order and determin the type of operation which the machine will perform, e.g. addition or subtraction o transfer from one part of the store to another. The function digits are denoted by and can have values from 00 to 99, e.g. the function 60 has the effect of transferri the contents of an accumulator to a storage location, and the function 10 adds the contents of one of the ordinary registers to one of the accumulators.

1.5.3 Most operations also involve a particular accumulator and this is specified b the A digit, D_8. Generally, A can take a value from 2 to 9 inclusive, denoting one the eight general purpose accumulators.

1.5.4 We can write, for example, A = 3, if we wish to use accumulator 3, and the digit 3 is put in the A position of the order. The accumulator 1 as already mention holds the control address. The purpose of the B digit, D_9, will be considered later for the time being B will be put equal to zero.

1.5.5 Although the instruction is held in the machine as above, it is convenient write it in the following form:-

F A B N *

1.5.6 As an example the function 14 copies the contents of location N to the accumulator specified under A so that the instruction.

F	A	B	N
14	2	0	231

copies the contents of location 231 into accumulator 2.

1.5.7 It will be noticed that the three left-hand zeros in N are omitted. This always be done.

1.6 REGISTERS AND THEIR CONTENTS - NOTATION

1.6.1 The 1000 (or more) locations are usually referred to as the main store; th are sometimes called ordinary registers, as opposed to the accumulators which may called X-registers. The word register alone could mean either an accumulator or a location in the main store.

1.6.2 It is conventional to use capital letters for addresses and small letters t represent the contents. Thus N denotes a main store address, and n its contents. contents of an accumulator specified in the A-digit of an order would be a, but i one were to refer to a specific accumulator it would be more usual to write X8, a for instance.

A small letter primed denotes the contents of a register after an instructio been obeyed, e.g. a' would denote the contents of the accumulator specified by A a the operation.

Example: If the number in location 431 is 103 and the number in accumulator 4 is then if N = 431 we have n = 103 and if A = 4 then a = -23. If now the function 10 is used to add these numbers and leave the result in accumu 4 i.e. if the instruction

10	4	0	431

is performed then a' = 103 - 23 = 80 and n' = 103 (unaltered).

* The reason for writing instructions this way is that the N part of the instruct varies in length and this ensures a neat layout when the programme is typed.

CHAPTER 2

ARITHMETICAL ORDERS & JUMP INSTRUCTIONS

2.1 THE ORDER CODE OF SIRIUS

The part of the order which determines the action performed by the computer is the function part. The basic functions available on Sirius will now be considered in some detail.

2.2 THE ORDERS 10 to 14

2.2.1 These are simple addition, subtraction or copying instructions between one of the ordinary registers in the main store and one of the accumulators. In each case the result is left in the accumulator and replaces the original contents of the accumulator, and in all these orders the contents of the main store locations are unchanged. The function 14 has the effect $a' = n$, i.e. the contents of location N are copied into accumulator A. The original contents of A, namely a, are lost and the contents of N, namely n, remain unchanged. For example the order

F	A	B	N
14	3	0	249

causes the contents of the location 249 to be copied into accumulator 3.

2.2.2 The function 10 has the effect $a' = a + n$, i.e. the contents of location N are added to the contents of accumulator A and left there, a being lost and n left unchanged. E.g. the instruction

10	4	0	341

causes the contents of location 341 to be added to accumulator 4 and left in accumulator 4.

2.2.3 The first 5 orders of group 1 may be summarised:-

F	Effect	Description
10	$a' = a + n$	Add the word in N into A.
11	$a' = a - n$	Subtract the word in N from that in A.
12	$a' = -a - n$	Negate the contents of A and subtract those of N.
13	$a' = -a + n$	Negate the contents of A and add those of N.
14	$a' = n$	Copy the contents of N into A.

Example: c, d, e are stored in locations 300, 301, 302; the quantity $(c - d + e)$ is required in accumulator 5.

14	5	0	300	$x_5' = c$
11	5	0	301	$x_5' = c - d$
10	5	0	302	$x_5' = c - d + e$

In order to be obeyed these instructions must be held in three consecutive registers in the store e.g. 500, 501, 502. These positions may be indicated on the left hand side as below

	F	A	B	N
500	14	5	0	300
501	11	5	0	301
502	10	5	0	302

2.3 THE ORDERS 60 and 64

These functions transfer or copy from the accumulators into the main store locations. The function 60 has the opposite effect to 14, i.e. $n' = a$, (copy the contents of accumulator A to storage location N). The original contents of N are lost and the contents of A remain unchanged.

Example: c, d are two numbers stored in locations 200, 201. Form $(c + d)$ and $(c - d)$ and store in locations 300, 301.

14	2	0	200	$x_2' = c$
10	2	0	201	$x_2' = c + d$
60	2	0	300	C(300) = $c + d$
14	2	0	200	$x_2' = c$
11	2	0	201	$x_2' = c - d$
60	2	0	301	C(301) = $c - d$

2.4 NOTATION

2.4.1 In this example the notation C(300) has been used to denote the contents of storage location 300. It can be used in general, as an alternative to that already in use, so that C(N) = n, C(A) = a.

2.4.2 The function 64 has the effect $n' = 0$. Clear the storage location N. A convention will be adopted by putting A = 0 in this order, since no accumulator needs to be referred to with the 64 order.

Example: Clear the location 400

64	0	0	400	C(400) = 0

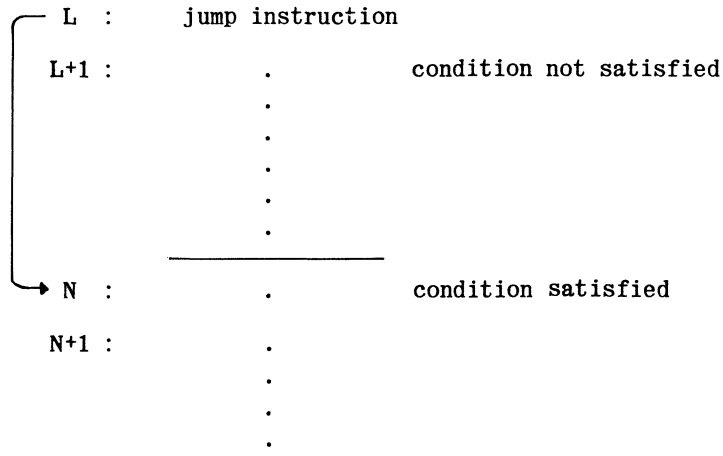
2.5 SOME OF THE ORDERS OF GROUP 5 - JUMP INSTRUCTIONS

2.5.1 In general the control address in accumulator 1, (which is used as the control register), is increased by one as each instruction is carried out. This causes the next instruction selected to be that in the next location in the store. This is not always convenient. Sometimes it is desirable to cause a jump or break in the sequence of instructions from those held in consecutive registers. This jump may be either backwards, in which case certain operations are repeated, or a jump forwards, in which case some operations are omitted. For example, the 55 order is an "unconditional" jump. In all cases this order causes a jump to be made to the address specified by the N digits.

Example: If programme is stored with some data in locations 200 onwards, and it is required to add the constant 0.314159 to the contents of accumulator 3, and then continue with the programme, then the 55 order may be used to jump round the data so:-

	F	A	B	N	
200	10	3	0	202	$x_3' = x_3 + .314159$
201	55	0	0	203	jump round data
202	+	.314159			(data)
203	60	3	0	301	store x_3 in 301

2.5.2 Sometimes it is necessary to programme two alternative courses of action depending on whether a particular condition is satisfied or not. If the condition is satisfied the jump is as described above, that is, to the location specified by N; otherwise no jump occurs and the computer proceeds sequentially as usual. If the condition is satisfied the address specified by the N digits of an order is transferred to the control register and becomes the control address. The instruction in location N is obeyed, the control address being increased by one to (N + 1) etc., the procedure then following the normal pattern. Suppose a jump instruction is held in location L, then the two possibilities may be represented diagrammatically as below:-

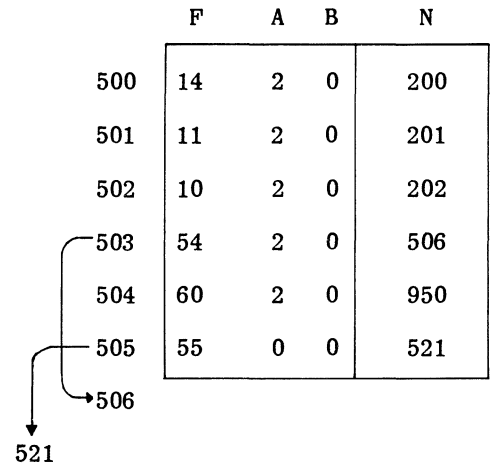


2.5.3 Sometimes a jump instruction is referred to as a 'transfer of control'. Some of the jump instructions are summarised below:-

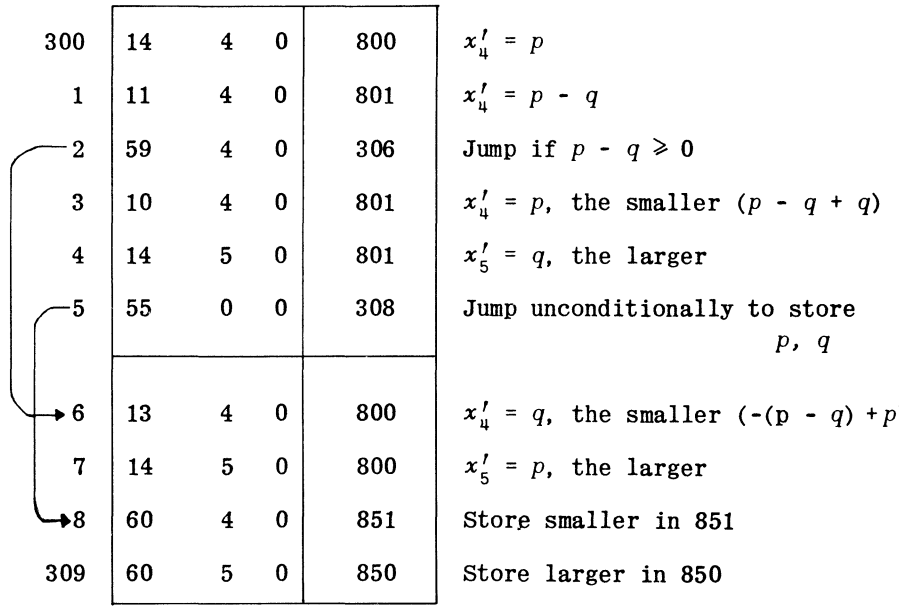
F	Effect	Description
51	Jump if MSD of $a \neq 0$.	Transfer control to the instruction in location N if the m.s. digit in A is not zero, otherwise obey the next instruction.
52	Jump if $a \neq 0$.	Transfer control to N if the C(A) are not zero, otherwise obey the next instruction.
54	Jump if $a < 0$.	Jump to N if the C(A) are negative, otherwise obey the next instruction.
55	Unconditional jump.	Jump to N in all cases.
56	Jump if MSD of $a = 0$.	Jump to N if the m.s. digit in A is zero, otherwise obey the next instruction.

F	Effect	Description
57	Jump if $a = 0$.	Jump to N if the C(A) are zero, otherwise obey the next instruction.
59	Jump if $a \geqslant 0$.	Jump to N if the C(A) are positive or zero, otherwise obey the next instruction.

Example 1: c, d, e are stored in locations 200, 201, 202. Form $(c - d + e)$, (ignoring overflow). If negative, jump to the instruction held in 506, otherwise, store in location 950, and proceed to instructions in 521. These instructions are held in locations 500 - 505.



Example 2: Given two positive numbers p and q in locations 800, 801, store the larger one in location 850 and the smaller in location 851.



CHAPTER 3

THE ORDERS OF GROUP 0

3.1 THE ORDERS 00-04

3.1.1 In this group of orders the N digits are used directly as an operand. N, with its sign extended by four digits to make it a ten digit number, is added into or subtracted from the contents of an accumulator or its complement.

Acc.	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
N					N_0	N_1	N_2	N_3	N_4	N_5

The four digits in the dotted section are the sign extension of N. If N_0 is 0 to 4, i.e. if N is positive, they are all zeros, if it is 5 to 9, i.e. if N is negative, they are all 9's.

The function 04 has the effect : $a' = N$

N, extended to 10 digits, is copied into A, the original contents of A being lost.

Example:

F	A	B	N
04	5	0	123456

puts the number 123456 into the least significant end of X5, clearing the first four digits. Note however that if the first digit of a 6-digit N had been 5 to 9, the first digits of the 10-digit N used would have been made 9's.

3.1.2 The first five orders of group 0 can be summarised as follows, with N extended to 10 digits.

F	Effect	Description
00	$a' = a + N$	Add N to a.
01	$a' = a - N$	Subtract N from a.
02	$a' = -a - N$	Negate a, and subtract N from result.
03	$a' = -a + N$	Negate a, and add N to result.
04	$a' = N$	Copy N to A.

Example: Suppose c, d, are stored in locations 305, 306, and it is required to form $(c - 100)$ in accumulator 3 and $-(1001 + d)$ in accumulator 8.

14	3	0	305	$x_3' = c$
01	3	0	100	$x_3' = c - 100$
14	8	0	306	$x_8' = d$
02	8	0	1001	$x_8' = -d - 1001$

3.2 N ZERO OR NEGATIVE

3.2.1 If N = 0 in the 04 order, then $a' = 0$, i.e. this can be used to clear the accumulator specified.

3.2.2 N = 0, in either the 02 or 03 order gives $a' = -a$, negating the contents of any accumulator.

3.2.3 If N is 0, it is usual to leave the N-column blank.

3.2.4 If N is negative the left-hand digits up to the full total of 10 are filled in with 9's: this is equivalent to extending the sign of N within the computer. The programmer merely writes '-N'.

e.g. if $x_2 = 7891$, the order

F	A	B	N
00	2	0	-197

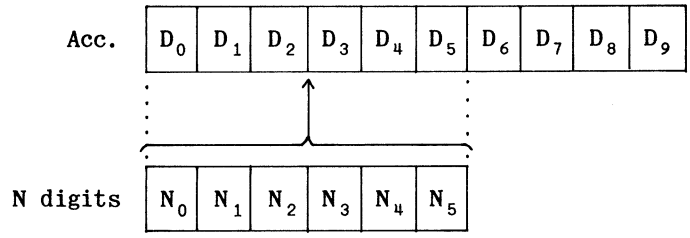
gives $x_2' = 7891 - 197 = 7694$

held in the computer as:

```
0000007891
9999999803
----------
0000007694
```

3.3 THE ORDERS 05 - 09

3.3.1 This group of functions is similar to the one above, the N digits being used directly as an operand. The difference is that the 6 digits are added or subtracted into the *most significant end* of the accumulator concerned.



3.3.2 Since the N digits may be regarded as an integer, this is equivalent to multiplying N by 10^4 and then adding or subtracting into the accumulator.

3.3.3 The function 09 has the effect $a' = 10^4 N$.

3.3.4 The six digits forming N are copied into the most significant end of the accumulator A; or the six digits N are multiplied by 10^4 and copied into the accumulator A.

09	7	0	320145

puts the number 320145 into the most significant end of X7 clearing the last four digits so that

$$x_7' = 3201450000$$

3.3.5 The last five orders of the 0-group can be defined as:-

F	Effect	Description
05	$a' = a + 10^{4}N$	Add N to m.s. end of A.
06	$a' = a - 10^{4}N$	Subtract N from m.s. end of A.
07	$a' = -a - 10^{4}N$	Negate a and subtract N from m.s. end of result.
08	$a' = -a + 10^{4}N$	Negate a and add N to m.s. end of result.
09	$a' = 10^{4}N$	Copy N to m.s. end of A.

Example 1: Suppose the negative number -3210876600 is held in X3 and it is required to negate this and add 5000000. The following order will do this

F	A	B	N
08	3	0	500

The steps performed are:-

$$x_3 = 6789123400$$

$$-x_3 = 3210876600$$

$$+10^{4}N \quad 000500$$

$$x_3' = 3215876600$$

Example 2: Two quantities P and I are stored in storage locations 800, 801. If a number T, held in 810 is 3, form P + I, if it is 5 form P - I, if it is 4 take P, and in each case store the result in location 805. These instructions are held in locations 350 onwards.

350	14	4	0	800	$x_4' = P.$
351	14	5	0	810	$x_5' = T.$
352	01	5	0	3	$x_5' = T - 3.$
353	57	5	0	359	Jump to 359 if T = 3.
354	01	5	0	1	$x_5' = T - 4.$
355	57	5	0	357	Jump to 357 if T = 4.
356	11	4	0	801	$x_4' = P - I.$
357	60	4	0	805	Store x_4 in 805.
358	55	0	0	361	Unconditional jump.
359	10	4	0	801	$x_4' = P + I.$
360	60	4	0	805	Store x_4 in 805.
361	(next instruction)				

CHAPTER 4

SHIFTING, MULTIPLICATION, DIVISION AND COLLATION

4.1 FRACTIONS AND SCALING

4.1.1 Up to now numbers in the computer have been regarded as integers, and have been restricted to lie in the range

$$-5000000000 \leqslant x \leqslant 4999999999$$

This assumes a decimal point immediately to the right of the least significant digit. This is only a convention, and we can imagine the decimal point to be between any two digits in the computer word which are convenient. The difference between 23598 and 23.598 is only a factor of 1000 = 10^3. In most problems numbers will have to be scaled to keep them within a given range in the computer. If the scaling factor is S, this will mean storing Sx, instead of the number x. Since Sirius is a decimal machine S will generally be a power of 10.

4.1.2 It is very often convenient to work with numbers in the form of pure fractions. This is the case when the decimal point is placed immediately to the left of the m.s. digit. The available range becomes

$$-.5000000000 \leqslant x \leqslant .4999999999$$

In other words, with this convention, fractions must be numerically less than ½, or equal to -½.

4.1.3 Adopting a similar convention for positive and negative fractions, as for integers, positive fractions appear in the machine as written, and negative fractions appear as the complement with respect to 1.

e.g.	+.4123	appears as	4123000000
	+.0012368	"	" 0012368000
	-.3069	"	" 6931000000
	-.4213604	"	" 5786396000
	-.000909	"	" 9990910000
	-.5	"	" 5000000000

4.1.4 As before, a number is positive if the first digit of its representation in the machine is between 0 and 4 inclusive, or negative if this first digit is between 5 and 9 inclusive.

4.2 MULTIPLICATION BY 10 OR SHIFT UP

All the 30 - 34 and the 20 - 29 functions involve multiplication by 10 and addition or subtraction. The 30 - 33, 20 - 23, 25 - 28 orders are similar to the orders 10 - 13, 00 - 03, 05 - 08 respectively; (each can be obtained by adding 20 to the function digits of the simpler orders). A zero is fed in at the least significant end of the accumulator, e.g. 0002354812 becomes 0023548120 when multiplied by 10. It should be noted that overflow may occur with these shift orders. A later paragraph will show how to deal with this situation.

4.3 THE ORDERS 30 - 33

The first four orders of this group are similar to the corresponding 10 - 13 orders except that the contents of the accumulator are first shifted up one place.

The N digits define an address in the working store, the three least significant digits only being used. These orders are described below.

F	Effect	Description
30	$a' = 10a + n$	Shift C(A) up one place and add C(N).
31	$a' = 10a - n$	Shift C(A) up one place and subtract C(N).
32	$a' = -10a - n$	Shift the complement of C(A) up one place and subtract C(N).
33	$a' = -10a + n$	Shift the complement of C(A) up one place and add C(N).

Example: X6 contains .0012341213 and the constant .2174828483 is stored in location 999. The order

F	A	B	N
33	6	0	999

forms $x_6' = -10(.0012341213) + .2174828483 = .2051416353$.

4.4 THE ORDERS 20 - 23

4.4.1 These orders are similar to those of the 00 - 03 group, except that the contents of the accumulator are first shifted up one place. N has its sign extended (as in 3.1.1) and is added to or subtracted from the accumulator.

4.4.2 The orders of this group are summarised below, with N extended to 10 digits.

F	Effect	Description
20	$a' = 10a + N$	Shift C(A) up one place and add N to A.
21	$a' = 10a - N$	Shift C(A) up one place and subtract N from A.
22	$a' = -10a - N$	Shift complement of C(A) up one place and subtract N from A.
23	$a' = -10a + N$	Shift complement of C(A) up one place and add N to A.

Example: If the number in X5 is 3624, then the instruction

23	5	0	-1

forms -10(3624) + (-1) = -36241.

4.5 THE ORDERS 25 - 28

The orders 25 - 28 are similar to the 05 - 08 orders except that the contents are first shifted up one place. The six N digits are again treated as an operand but added to or subtracted from the *most significant end* of the accumulator. The functions of this group have the following effect

F	Effect	Description
25	$a' = 10a + 10^{4}N$	Shift C(A) up one place and add N to the m.s. end of A.
26	$a' = 10a - 10^{4}N$	Shift C(A) up one place and subtract N from the m.s. end of A.
27	$a' = -10a - 10^{4}N$	Shift the complement of C(A) up one place and subtract N from m.s. end of A.
28	$a' = -10a + 10^{4}N$	Shift the complement of C(A) up one place and add N to m.s. end of A.

Example 1: The fraction 0.035 is in accumulator 6; multiply this by 10 and subtract 0.45. The order

F	A	B	N
26	6	0	450000

forms $x_6' = 10(.035) - .45 = -.10$.

Example 2: If $x_3 = .041231$ the order

27	3	0	5001

has the effect $x_3' = -.41231 - .005001 = -.417311$.

4.6 SIMPLE SHIFT

4.6.1 A simple shift up of one place i.e. multiplication by 10 can be achieved by putting the N digits equal to zero in any of the orders 20, 21, 25 or 26, e.g. the order

20	3	0	

has the effect $a' = 10a$.

4.6.2 A simple shift up together with a change of sign, i.e. multiplication by -10, can be achieved by putting the N digits zero in the 22 or 27 order, e.g.

22	3	0	

has the effect $a' = -10a$.

4.7 THE ORDER 34

The 34 order has the effect

$$a' = 10a + \text{M.S.D. of } n.$$

This shifts the contents of A up one place and puts the m.s. digit of the contents of N in the l.s. digit position of A.

Example 1: If x_4 = 0021312486 and C(499) = 4632189106 then the order

F	A	B	N
34	4	0	499

performs the following steps: -

x_4 : 0021312486

$10x_4$: 0213124860

n : 4632189106

x_4' : 0213124864

Example 2: Given a 20-digit number .02341213453100000000 in 400, 401, shift it up one place and restore it.

14	2	0	400	x_2' = .0234121345
34	2	0	401	x_2' = .2341213453
60	2	0	400	Store in location 400
14	2	0	401	x_2' = .3100000000
20	2	0		x_2' = .1000000000
60	2	0	401	Store in location 401

4.8 DIVISION BY 10 OR SHIFT DOWN. GROUP 4 ORDERS

4.8.1 There are four orders for division by 10.

4.8.2 Division by 10 is a shift of a number one place right, usually said to be a shift "down", since it becomes numerically smaller.

4.8.3 There are two main points to be considered in connection with a shift-down. First the question of signs, remembering the convention in Sirius that the first digit determines the sign of a number. In the case of positive numbers, a zero fed in at the most significant end will maintain the sign and give a correct result. Negative numbers, on the other hand, must have as their first digit anything from 5 to 9. If a 9 is fed in at the most significant end for each shift-down, the correct arithmetical result will be obtained. Using x to denote any number and x^* its representation in the computer, the following examples should make this clear.

x	x^*	$\frac{1}{10}x^*$
-.2134800000	7865200000	9786520000
-10121	9999989879	9999998987
-.0000012136	9999987864	9999998786

4.8.4 The 44 order is a simple shift-down, and has the effect

$$a' = \frac{1}{10}a$$

4.8.5 The second point will be noticed from the numerical examples above. The l.s. digit in x^* is dropped when it is shifted down. It is not always desirable that this should happen. In the second example, a 9 is dropped off, leaving 7 as the last digit; a more accurate result would be an 8. Another function is provided which rounds off the last figure in this way. This is the 40 function. The rounding is performed by adding 5 to the l.s. digit of a number and then shifting right. In the example quoted, 5 added to the last 2 digits gives 79 + 5 = 84; when the whole number is shifted, the 4 drops off leaving the 8 as the last digit.

4.8.6 The following are examples of a rounded shift down, if the 40 order is used.

x	x^*	$\frac{1}{10}x^*$, rounded
+12378	0000012378	0000001238
-456890	9999543110	9999954311
-.2234249812	7765750188	9776575019

4.8.7 There are four orders altogether for shifting down, as summarised below.

F	Effect	Description
40	$a' = \frac{1}{10}(a + 5)$	Rounded arithmetical shift-down, with sign extension.
44	$a' = \frac{1}{10}a$	Unrounded arithmetical shift-down, with sign extension.
45	$a' = \frac{1}{10}(a + 5)$ + LSD of N	Rounded shift-down, the l.s. digit of N being copied to the m.s. digit of A.
49	$a' = \frac{1}{10}a$ + LSD of N	Unrounded shift-down, the l.s. digit of N being copied to the m.s. digit of A.

4.8.8 The full use of the 45 and 49 orders will become apparent when modification (Chapter 5) has been considered. For orindary single-length shifts into 40 and 44 orders will usually be sufficient.

4.9 THE OVERFLOW INDICATOR

4.9.1 Numbers stored in Sirius must lie within the range

$$-½ \leqslant x < ½$$

if regarded as fractions or in the range

$$-5000000000 \leqslant x < 4999999999$$

if regarded as integers. But arithmetical operations may cause numbers to arise which exceed capacity, i.e. overflow.

E.g. .312 + .418 = .730 and + .730 is outside the range, and in fact will be interpreted as a negative number of magnitude 1 - .730 = .270.

4.9.2 In this case overflow is detected by a change of sign in the result where none should occur.

4.9.3 Another form of overflow occurs when significant digits are lost, for example in the shift functions. When 1348900139 is shifted up one place, the first digit is lost. Moreover a shift may cause a number to change in sign. A negative number -.07 held as .93 when shifted up becomes .3, a positive number, the complement of the desired one.

4.9.4 All these occurrences are detected by the overflow indicator. This indicator has only two states, "clear", which is the normal state, and "set" when overflow occurs. It remains set until tested by means of either of two special orders, (the 53 and 58 functions), which then clear the indicator.

4.9.5 The computer will continue to operate even though the overflow indicator is set. If this is not foreseen by the programmer and steps taken to correct it, the resulting calculations will usually be meaningless.

4.10 THE ORDERS 53 and 58

4.10.1 These are the two functions which enable the programmer to deal with overflow. The function 53 has the effect "jump to N if overflow is set and clear the overflow indicator". That is if the overflow indicator is set, control is transferred to the address specified by N, and the indicator is cleared. Otherwise the next instruction is obeyed. For example the instruction

F	A	0	N
53	0	0	303

has the effect, "if the overflow indicator is set jump to obey the instruction held in location 303, otherwise obey the next instruction".

4.10.2 In this order and the 58 order, the A digit is put equal to zero by convention, since no accumulator needs to be specified.

4.10.3 The function 58 has the effect "jump to N if overflow is clear, otherwise clear the overflow indicator", i.e. if there is no overflow control is transferred to the address specified by N, otherwise the overflow indicator is cleared and the next instruction obeyed. e.g. if the order held in register 506 is

506	58	0	0	365

the control is transferred to location 365 if overflow is clear, otherwise the next instruction is taken from 507.

Example: Two numbers p and q are stored in locations 500 and 501. Write a programme to add these numbers and store the result in 502. If overflow should occur replace p and q by

$$\frac{p}{10} \text{ and } \frac{q}{10}, \text{ and put the sum } \frac{p+q}{10} \text{ in 502.}$$

Also store zero in register 503 if there is no overflow, or the number 1 if the result has had to be shifted.

	F	A	B	N	
200	64	0	0	503	Put zero in location 503
1	14	2	0	500	$x_2' = p$
2	10	2	0	501	$x_2' = p + q$
3	58	0	0	213	Jump to 213 if OVR clear
4	14	2	0	500	$x_2' = p$
5	40	2	0		$x_2' = \frac{p}{10}$, rounded
6	60	2	0	500	Store in 500
7	14	2	0	501	$x_2' = q$
8	40	2	0		$x_2' = \frac{q}{10}$, rounded
9	60	2	0	501	Store in 501
10	10	2	0	500	$x_2' = \frac{q}{10} + \frac{p}{10}$
11	04	3	0	1	$x_3' = 1$
12	60	3	0	503	Store in 503.
13	60	2	0	502	Store $(p+q)$ or $\frac{1}{10}(p+q)$ in 502.

4.11 MULTIPLICATION. THE 79 ORDER

4.11.1 If one multiplies together two decimal numbers each having the same number of digits one gets a product with twice as many digits e.g.

$$.38 \times .61 = .2318$$

In the same way the product of two 10-digit fractions will be a 20-digit fraction, i.e. double the length and will occupy two computer words. Two accumulators are used to contain the product, although of course the full double-length number is not always required.

4.11.2 The Sirius the 79 order is used to give a full double-length product. The m.s. half of this product can be taken as an unrounded single result (e.g. as if the answer had been taken as .23 in the example above). With integers however, provided the answer is capable of being held single length, it will be in the l.s. half. For example, in a 3 digit word machine 024 could be multiplied by 015 to give 000360, and 360 is the single-length answer.

4.11.3 One of the factors, usually called the multiplier, should be placed in X9, and the other, called the multiplicand, in XB. The resultant double-length product appears in two accumulators, XA and X9. The order may be summarised

$$79\ (a,\ x_9)' = b \times x_9$$ Place the double-length product of b and x_9 in a and x_9.

The N digits are not used, and are usually left blank.

Example 1: Two numbers p, q are held in 333 and 334. Form the double length product and store it in 350, 351.

	F	A	B	N	
401	14	4	0	333	$x_4' = p$
402	14	9	0	334	$x_9' = q$
403	79	6	4		$(x_6, x_9)' = pq$
404	60	6	0	350	Store m.s. half in 350
405	60	9	0	351	Store l.s. half in 351

Example 2: Two small integers j, k are stored in 900 and 901. Place the product in 902. Assume that this product is less than 5×10^9 in absolute value (i.e. it is a single-length number).

14	5	0	900	$x_5' = j$
14	9	0	901	$x_9' = k$
79	2	5		$x_9' = jk$
60	9	0	902	Store in 902

Example 3: Two fractions c, d are in the storage locations 703 and 704. Form the single-length correctly rounded product and store it in 705.

220	14	2	0	703	$x_2' = c$
221	14	9	0	704	$x_9' = d$
222	79	3	2		$x_3' = cd$ (m.s. half)
223	59	9	0	225	Jump if $x_9 \geqslant 0$
224	00	3	0	1	$x_3' = cd + 1$
225	60	3	0	705	Store cd (rounded) in 705

To form the rounded single-length product from the double length number in accumulators 3 and 9 it is necessary to add one to the l.s. end of accumulator 3 if the m.s. digit of accumulator 9 is 5 or more. We introduced a convention with single-length numbers, which treats the left-hand digit in a special way with regard to the sign of a number; but when double-length numbers are stored in two accumulators, the left-hand digit in accumulator 9 (i.e. the l.s. half of the number) is merely one of the digits of the product and has no special significance with regard to signs. For the purposes of testing its value however, the left-hand digit in accumulator 9 may be considered as obeying the sign convention adopted in Sirius, i.e. if this left-hand digit is 5 or more the contents of accumulator 9 would be regarded as negative. Hence the 59 order which is used to test the sign of a number can be used here to test whether to do a correction in accumulator 3. The two orders held in locations 223 and 224 will therefore perform the rounding.

4.11.4 The largest number which can be produced as the result of multiplication is when (in terms of fractions) -0.5 is multiplied by -0.5. This gives the answer +0.25. As this is within the range of Sirius, the 79 order can never set the OVR.

4.11.5 Care must be taken not to specify B = 9, if the square of a number is required, since during multiplication the original contents of accumulator 9 are destroyed.

The multiplicand in accumulator B is not destroyed, unless the same accumulator is specified both in the A and B digits. (This is permissible since the more significant half of the product does not appear in A until multiplication is complete). If A = 9 the more significant half of the product appears in accumulator 9, the less significant half being lost.

4.11.6 To summarise, B must not be equal to 9, but it is permissible to have A equal to B or A equal to 9. In the latter case the most significant half of the product appears in accumulator 9, and the least significant half is destroyed.

4.12 DIVISION

4.12.1 It is desirable that the operation of division should be the converse of multiplication, so that if the product of u and v is divided by v, we should get u as quotient, i.e.

$$\frac{uv}{v} = u$$

4.12.2 The product uv is a double length number (20 digits in Sirius) and the division operation is to divide this double length number (the dividend) by a single length one (the divisor) to give a quotient and remainder, both single length.

4.12.3 The double length dividend is formed from C(A) and C(9) taken together as a 20 digit number. The divisor is taken from B. The division operation performed is thus

$$\frac{a + 10^{-10}\, x_9}{b}$$

where x_9 is taken as positive, i.e. if regarded as a fraction the range is 0 to 1 rather than -½ to +½.

4.12.4 In general the quotient resulting from such a division will not terminate exactly, although of course as many digits accuracy as are required can always be obtained by taking the process sufficiently far. In a computer it is necessary to terminate the process at some fixed point, and in Sirius a single length, 10 digit, quotient is obtained. The remainder is then the number which, if treated as a new dividend, would enable the division to continue; it is always less than the divisor.

4.12.5 The division orders in Sirius work in an unsigned mode; this means that the correct answers will only be obtained if both divisor and dividend are positive. To deal with the case where either or both of these may be negative, it is necessary to use a subroutine for division which will ensure that these are complemented when necessary before dividing, and that the quotient is complemented if necessary after dividing.

4.12.6 If negative numbers are used by a division order they will be treated as being positive, in the range 0 to 1. Thus if we try to divide -.01 by +.4, in the machine these are represented by

```
                99900000000000000000   (the 20 digit dividend)
and                   4000000000      (the 10 digit divisor)
```

The machine will interpret this as dividing .99 by .4, which overflows, and no sensible result will be obtained, although the correct answer, -.025, is within range. Conversely, if we divide +.4 by -.01, there is no correct answer, as -40 is out of range, but the machine will divide .4 by .99, and obtain a quotient of .4040404040. This would appear to be within range, and overflow will not be set.

4.12.7 Bearing in mind the unsigned nature of Sirius division overflow will occur if the quotient would be +1.0 or more. However, as normally one will in fact be using the signed convention, the answer is really out of range if it is +.5 or more.

For example if we divide +.3 by +.4, both are positive and the quotient will appear as 7500000000. This is correct only if we ignore the sign convention. To cater for the two possibilities, two versions of the division order are provided; the 70 order will set overflow if the quotient is out of range according to the sign convention, i.e., if it is greater than or equal to .5, whereas the 75 order will set overflow only if it is greater than or equal to +1.0.

4.12.8 Once a division overflows neither the quotient nor remainder can be expected to bear any relation to the correct ones, i.e. the quotient obtained will not necessarily differ from the correct, but out of range, quotient by a whole number.

4.12.9 Both division orders place the quotient in X9 and the remainder, which is always positive and less than the divisor, in A. The division orders are summarised below:-

F	Effect	Description
70	$x_9' + \frac{10^{-10}\ a'}{b} = \frac{a + 10^{-10}\ x_9}{b}$ Set overflow if $x_9' \geqslant ½$	Divide the double-length, unsigned number, in A and X9 by the unsigned number in B setting the quotient in X9 and the positive remainder in A. Set overflow if $x_9' \geqslant ½$.
75	$x_9' + \frac{10^{-10}\ a'}{b} = \frac{a + 10^{-10}\ x_9}{b}$ Set overflow if $x_9' \geqslant 1.0$	As for 70 order but overflow is set only if $x_9' \geqslant 1.0$.

The N address is not used with either of these orders.

Example 1: Two positive integers, c and d, both less than 4999999999 are held in 300 and 301. Form the quotient and remainder on dividing c by d in X9 and X8 respectively.

F	A	B	N	
14	9	0	300	} Double length dividend to (8,9).
04	8	0		
14	7	0	301	Divisor to X7.
70	8	7		Divide, quotient to X9 and remainder to X8.

Example 2: c, d, e are positive fractions in X9, X3 and X4 respectively. Form the fraction cd/e in X9, setting overflow if the result is out of range.

79	2	3		Form cd in (2, 9)
70	2	4		Quotient to X9.

4.13 COLLATION

4.13.1 It is sometimes necessary to pick out a part of a word for separate treatment or examination. This need may arise in many different ways, but the most usual is in problems involving many small numbers, where it is common practice to "pack" two, or more, such numbers in one word. The collate order is then necessary to enable the components of the word to be unpacked.

4.13.2 The orders 66 and 68 have been provided to give these facilities; they are described as "collating" orders, or as "and" orders. They operate on words digit by digit, retaining or eliminating (i.e. replacing by zero) each digit according to the value of the corresponding digit in another word, known as the mask. The mask in general will consist of only 1's and 0's (the effects of using digits other than 1 or 0 in the mask will be explained later). The effect of a collate order can be illustrated by an example.

OPERAND	1234567809
MASK	0000001111
RESULT	0000007809

4.13.3 The effects of the collate orders are as follows:

F	Effect	Description
66	$a' = a \,\&\, N$	Retain or eliminate the digits in A, according as to whether the digits of N (10^{11}N) are 1 or 0 respectively.
68	$a' = a \,\&\, 10^{11}N$	

Example: Two numbers, c and d, are packed together in X2, c occupying the first 5 digits, and d the second 5. Place c in X3, with the last 5 digits zero, and d, with the first 5 digits zero, in X2.

F	A	B	N	
04	3	2		Place (c, d) into X3
66	2	0	11111	Retain d in X2
68	3	0	111110	Retain c in X3

The 0432 order will be explained later. It merely places a copy of X2 into X3 in this example.

4.13.4 The full effects of the 66 and 68 orders, and also the related 65 and 67 orders, are described in 8.8.

CHAPTER 5

MODIFICATION

5.1 MODIFICATION

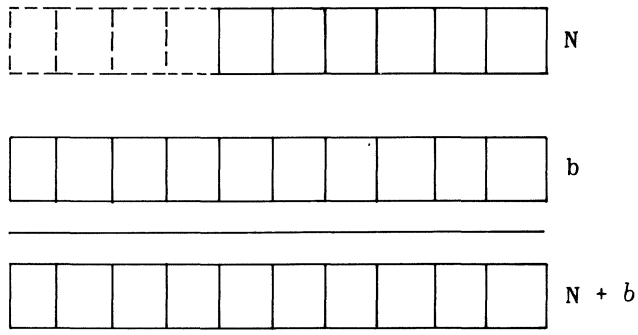
5.1.1 It is often necessary to repeat a set of instructions several times, each time operating on different storage registers. For instance, if 100 numbers in consecutive addresses in the store are to be added up, it would be possible to write out 100 orders to do it, but this would not be an acceptable solution because of the length and inflexibility of the programme. A means is therefore required for making a single order in the store refer to a set of addresses one by one.

5.1.2 In principle this can be done with existing facilities, as it would be possible to extract an order from the store, add one into its N-address, and plant it back to be obeyed. If this were done in a loop of instructions, the programme to add up 100 numbers (or indeed any number of numbers) would need half-a-dozen or so orders.

5.1.3 This technique has been used, especially on earlier computers, but is usually rather inefficient, and would be particularly so in Sirius. There is therefore an automatic way of doing it, making use of XB, whose purpose has been hitherto undefined.

5.1.4 This facility is known as modification of orders, and the effect is that b (i.e. the contents of XB) is added to N before this is used, either as an address or as an operand. If XB is zero, b is also zero (in most orders) so orders in this form are as already stated. (see however 6.6.2).

5.1.5 The process of modification is as follows. N, with its sign extended by four places to make it a ten digit number, is added to b; the result of this is then regarded as the true N to be used. The addition can thus be represented as



The four digits in the dotted lines are the sign extension of N, and are all 0's or all 9's, depending on whether N is positive or negative. This addition can never set the overflow register.

5.1.6 (N + b) only exists in the order register of Sirius; that is the process of modification has no effect on the order in the store, or on b. The order obeyed is as if B were 0, and the N address were N + b, assuming that 10 digits were available to hold this.

5.2 MODIFICATION OF THE ORDERS 10 - 14, 60, 64, 30 - 34, 50 - 59, 69

In all these orders, the N-digits specify an address, so only 4 digits (3 if the store is only 1000 words) are actually used. In the jump orders however (51 - 59, and 69) the whole of (N + b) is placed in X1, if the jump is effective.

Example: If accumulator 2 contains the number 10 then the order

F	A	B	N	
11	3	2	609	$a' = a - n$

subtracts the contents of location 609 + 10 = 619 from the contents of accumulator 3.

5.3 USE OF ACCUMULATOR 1

5.3.1 It has been remarked that accumulator 1 is the control register and holds the control address. For programming purposes this is the address of the next instruction. This can be useful particularly in the case of the jump instructions. Suppose we put 1 in the B position of the order, that is we specify accumulator 1 in the B position. Then the effect of this is to add the contents of accumulator 1, namely the control address, to the N digits (which specify an address when used with jump instructions). This can be stated:

when B = 1 the effective N address = C(1) + N

The order held in location 303

303	55	0	1	1

is an unconditional jump to C(1) + 1 = 304 + 1 = 305, remembering that the control address is that of the next instruction i.e. current instruction address plus 1.

5.3.2 If this order is used anywhere in the store it means "jump ahead two instructions". No actual address is specified. The N is in fact a relative address relating to the control address. A negative N may be used to jump back to a previous order e.g.

56	2	1	-3

means "jump back two instructions if the most significant digit in X2 is zero". Example 2 in section 3.3.5 could be re-written without reference to the locations 350 to 361, by using accumulator 1 as a modifier in this way

14	4	0	800	$x_4' = P$
14	5	0	810	$x_5' = T$
01	5	0	3	$x_5' = T - 3$
57	5	1	5	Jump 6 steps ahead if T = 3
01	5	0	1	$x_5' = T - 4$
57	5	1	1	Jump 2 steps ahead if T = 4
11	4	0	801	$x_4' = P - I$
60	4	0	805	Store x_4 in 805
55	0	1	2	Jump 3 steps ahead
10	4	0	801	$x_4' = P + I$
60	4	0	805	Store x_4 in 805
				Next instruction

5.4 LOOP STOP

If N = -1 is used with B = 1, and the instruction in location L is

	F	A	B	N
L	55	0	1	-1

then the effective N address is (L + 1) - 1 = L, i.e. a jump to the same instruction. Although strictly speaking the machine is obeying programme, in fact nothing in the store is changing, and the machine is considered to have stopped. This type of stop is called a loop stop and is often used at the end of a programme.

5.5 COUNTING

Sometimes it is required to perform an operation a given number of times, say r, i.e. the operation goes through r cycles. This can be done by putting the number r into one of the accumulators, to be used as a count. After each operation, r is reduced by 1 and the result tested to see if it is yet zero.

Example: Form the sum of 10 numbers held in registers 500 - 509, and store the sum in 525.

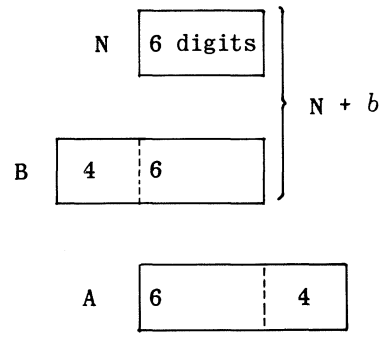
200	04	2	0	10	$x_2' = 10$ (to be used to count through the set of numbers).
1	04	3	0		$x_3' = 0$ (modifier)
2	04	4	0		Clear X4 to add in required sum
→ 3	10	4	3	500	$x_4' = 0 + C(500)$, first time
4	00	3	0	1	Add 1 to modifier in X3
5	01	2	0	1	Subtract 1 from x_2
└ 6	52	2	0	203	Jump back to add in next number if $x_2' \neq 0$
7	60	4	0	525	Store sum in 525

N.B. This example could be done more economically by using only one accumulator for both modifying and counting. The counter-modifier is set equal to 9 and we then work backwards through the store, reducing the count by 1 at each step, until it becomes negative.

5.6 MODIFICATION OF THE ORDERS 05 - 09, 25 - 29, 68

5.6.1 In these orders N is used directly as an operand, but modification is still the same, in principle. The contents of the accumulator specified in the B-digit are added to the N digits, before the execution of the order, so that (N + b) replaces N, as the order is carried out. (The order remains unaltered in the storage register which contains it).

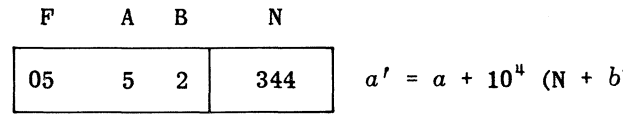
5.6.2 Since, in these orders, the 6 N digits are added to, subtracted from, or collated into, the m.s. end of C(A) (or 10 × C(A) in the case of the group 2 orders), it is only the 6 l.s. digits in B which can be added to N. The four m.s. digits are irrelevant. Modification of the 05 - 09, 25 - 29, 68 orders can be represented as



These orders can be re-written as:-

Order	Effect
05	$a' = a + 10^4 (N + b)$
06	$a' = a - 10^4 (N + b)$
07	$a' = -a - 10^4 (N + b)$
08	$a' = -a + 10^4 (N + b)$
09	$a' = 10^4 (N + b)$
25	$a' = 10a + 10^4 (N + b)$
26	$a' = 10a - 10^4 (N + b)$
27	$a' = -10a - 10^4 (N + b)$
28	$a' = -10a + 10^4 (N + b)$
29	$a' = 10a +$ M.S.D. of $10^4 (N + b)$ in L.S.D. position
68	$a' = a \,\&\, 10^4 (N + b)$

Example: If X2 contains the number 11 and X5 the number 3000000023, then the order



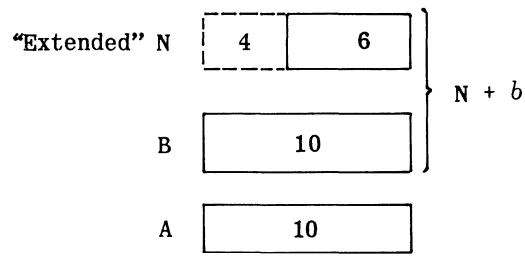
$a' = a + 10^4 (N + b)$

gives $x_5' = 3003550023$.

5.7 MODIFICATION OF THE ORDERS 00 - 04, 20 - 24, 66

5.7.1 These orders are similar to the 0-order and 2-order groups above, except that the N digits are now added to or subtracted from the l.s. end of the A-accumulator (or 10 times the A-accumulator). In this case the sign of N is extended to the full 10 digits before operating on the contents of A, and also when the order is modified the *entire* contents of the B-accumulator are added to the extended form of N, before this is added to, subtracted from, or collated into the l.s. end of C(A) or 10 x C(A).

5.7.2 Modification of the 00 - 04, 20 - 24, 66 orders can then be represented like this:



These orders can be re-written:-

Order	Effect
00	$a' = a + (N + b)$
01	$a' = a - (N + b)$
02	$a' = -a - (N + b)$
03	$a' = -a + (N + b)$
04	$a' = (N + b)$
20	$a' = 10a + (N + b)$
21	$a' = 10a - (N + b)$
22	$a' = -10a - (N + b)$
23	$a' = -10a + (N + b)$
24	$a' = 10a$ + M.S.D. of $(N + b)$ in L.S.D. position
66	$a' = a$ & $(N + b)$

Example 1: If X3 contains the number 10, X4 the number 120, then the order

F	A	B	N
00	4	3	636

gives $a' = a + (N + b)$

$x_4' = 120 + 636 + 10 = 766$

Example 2: If 1000000 is in X3 and -120000089 in X7, then the order

23	7	3	-3000	$a' = -10a + (N + b)$

gives $x_7' = 1200997890.$

5.8 ARITHMETIC IN TWO ACCUMULATORS

In the case when N = 0, the orders 00 - 04 and 20 - 24 reduce to a simplified form, allowing arithmetical operations between two accumulators. They become:-

Order	Effect	Order	Effect
00	$a' = a + b$	20	$a' = 10a + b$
01	$a' = a - b$	21	$a' = 10a - b$
02	$a' = -a - b$	22	$a' = -10a - b$
03	$a' = -a + b$	23	$a' = -10a + b$
04	$a' = b$	24	$a' = 10a$ + M.S.D. of b in L.S.D. position

Example: If small integers c, d, e are stored in locations 355, 356, 357, form $d^2 - 4ce$ and store in location 499

F	A	B	N	
14	2	0	355	$x_2' = c$
14	9	0	357	$x_9' = e$
79	3	2		$x_9' = ce$
04	2	9		$x_2' = ce$
00	2	2		$x_2' = 2ce$
00	2	2		$x_2' = 4ce$
14	3	0	356	$x_3' = d$
04	9	3		$x_9' = d$
79	4	3		$x_9' = d^2$
01	9	2		$x_9' = d^2 - 4ce$
60	9	0	499	Store in 499

Note that it is faster to form $4ce$ by doubling ce twice rather than by multiplication, even though one extra order is required.

5.9 MODIFICATION OF THE ORDERS 40, 44, 45, 49

5.9.1 There is no modification of the 40 and 44 orders, as the N digits are not used.

5.9.2 The 45 and 49 orders both refer to the least significant digit of N. This is really (N + b), and the complete effect of these orders is therefore

$$45\ a' = \frac{1}{10}(a + 5) + \text{L.S.D. of (N + } b\text{) in M.S.D. position}$$

$$49\ a' = \frac{1}{10}a + \text{L.S.D. of (N + } b\text{) in M.S.D. position}$$

In practice N will usually be zero in these orders, and their effect and description are then

$45\ a' = \frac{1}{10}(a + 5) + \text{L.S.D. of } b$ — Rounded shift-down of one place, the l.s. digit of b being copied to the m.s. digit of a.

$49\ a' = \frac{1}{10}a + \text{L.S.D. of } b$ — Unrounded shift-down of one place, the l.s. digit of b being copied to the m.s. digit of a.

Example: Shift down, unrounded, by one place, the double-length number in X8 and X9.

49	9	8		Shift x_9 down, taking the top digit from the bottom of x_8.
44	8	0		Shift x_8 down one place.

5.10 THE DUMMY ORDER 50

The 50 order is a "dummy" order. That is there is no effect on anything. It can be written:

F	A	B	N	
50	0	0		no effect

This order is sometimes used to replace orders which are found to be redundant during programme development, or may be written into the original programme at points where expansion may be necessary.

5.11 THE "WAIT" ORDER 99

5.11.1 The 99 function causes the computer to wait. It can be used to make a pause in the running of a programme. The machine may be re-started by pressing the "continue" button.

5.11.2 The order can be written as:

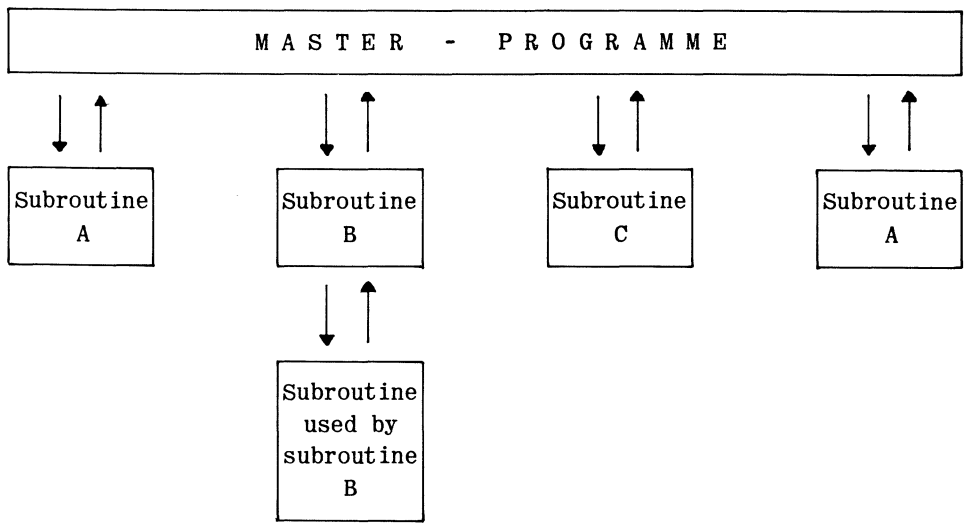
99	0	0		Wait

It is often inserted at a point where a data tape has to be read, to enable this to be placed in the tape-reader.

5.12 USE OF SUBROUTINES

5.12.1 There are a number of processes carried out by computers which are used by more than one programme, or at different points of the same programme. A piece of programme which performs a specific operation, such as printing out one number on the teleprinter, is known as a subroutine. It is in a sense subservient to the main programme, called the master programme, and it will usually perform its specified operation on numbers supplied to it by the master programme, and then re-enter (or "return control" to) the master programme.

5.12.2 One main programme may employ several subroutines, and in general some of the subroutines may themselves use subroutines, so that the whole structure may be on several levels, thus:



5.12.3 The order needed to call in a subroutine is called the "cue" to the subroutine; it is always an unconditional jump, jumping to the first order (or some other order as given by the subroutine's specification) of the subroutine.

5.12.4 When the subroutine has finished its task it must return to the master programme at the appropriate point. It normally does this by obeying a modified jump instruction; the N-address in this instruction will be zero and the modifying accumulator will contain the address of the order in the master programme to which control is to be returned. The master programme must thus ensure that this modifying accumulator, which is conventionally X2, but is really determined by the way the subroutine is written, contains the correct address at the point when the subroutine is entered.

5.12.5 A typical sequence of orders to enter a subroutine starting in word 850 say, is

		F	A	B	N	
	512	04	2	0	514	Set "link" in X2
	513	55	0	0	850	Enter subroutine
SUBROUTINE RETURNS HERE	514					

The address set in X2, or sometimes X2 itself, is called the "link". The order in the subroutine which finally returns to the master programme will be

55	0	2		Jump to the address specified in X2.

5.13 THE 69 ORDER

5.13.1 As the two orders given in 5.12.5 are needed every time a subroutine is entered, an order has been provided which has the effect of both. It is

$$69\ a' = x_1 \text{ and jump to N}$$

This places the address of the next instruction into A, and jumps. Thus in the master programme, in the same example as above, we write

512	69	2	0	850

and the subroutine will now return to 513, as 513 is placed in X2 by the 69 order.

5.13.2 The 69 order is not quite as flexible as the two separate orders, as it is only possible to make the subroutine return to the next instruction. In practice this is the usual requirement, but if it is not, it will be necessary to use two orders as in 5.12.5.

5.13.3 If X1 is specified in the A part of a 69 order, the machine will stop. This order has been banned from the code as it could have had no useful effect. This will be referred to again in Chapter 6.

5.14 THE 24 AND 29 ORDERS

5.14.1 The 24 and 29 orders belong to groups discussed in Chapter 4, but they have been left until now, as they are really only useful when modified. Their specification is as follows, regarding (N + b) as a ten digit number.

24 $a' = 10a + D_0$ of (N + b)

29 $a' = 10a + D_4$ of (N + b)

Example 1: Suppose X2 contains 1234567890 and X3 contains 9876543210.

Then the effect of

F	A	B	N
24	3	2	23

is

N	0000000023
b	1234567890
N+b	1234567913
a'	8765432101

If 29 is used instead of 24, the effect is

N	0000000023
b	1234567890
N+b	1234567913
a'	8765432105

5.14.2 In practice N is often zero with both these orders. The 24 is needed for a double-length shift up of one place, when the M.S.D. of the least significant half of the double-length number has to be transferred into the L.S.D. position of the most significant half. The 29 is less common, but is useful in subroutines to extract a digit from the D_4 position which has been set there in the main programme.

CHAPTER 6

INPUT AND OUTPUT; THE OPERATORS CONTROLS

6.1 PAPER TAPE AND TELEPRINTER EQUIPMENT

6.1.1 The equipment fitted to Sirius, so that orders and numbers can be fed into the computer and results extracted from it, uses standard five-hole punched paper tape. This tape has a row of small holes punched ten per inch along its length; these are called sprocket holes and are used to position the tape and move it. Each sprocket hole defines the position of a "character" which is made up of a maximum of five holes, each slightly larger than a sprocket hole, punched in a row across the tape and in line with the corresponding sprocket hole.

6.1.2 The tape is read into the computer by one (or more) Ferranti High-Speed photo-electric tape-readers type TR5, which can read at a maximum speed of 300 characters per second. Output is via a Teletype punch (again more than one can be attached) operating at up to 60 characters per second; this produces five-hole punched paper tape. This output tape can then be printed out on a page teleprinter, either immediately, or later. Teleprinters print at 10 characters per second (older types at 7) so if there is a lot of output from a particular programme it may be impossible for one teleprinter by the side of the computer to keep up; it would then be usual to take the tape away from the computer when the programme had finished and print it out on other equipment.

6.1.3 The input tape for the computer is prepared either on a keyboard perforator or on a page-teleprinter with re-perforator attachment. Both have keyboards rather like those on typewriters; when a key is operated the corresponding character is punched and the tape is advanced to the next sprocket hole position. There is a facility for back-spacing the tape so that a wrongly punched character can be over-punched by the "Erase" character, consisting of 5 holes, and therefore capable of obliterating any other character. The keyboard perforator produces just a tape; the page teleprinter with reperforator attachment produces a printed page in addition.

6.1.4 There are 32 (2^5) possible characters with the five-hole code used, which is identical with that used for the Ferranti Pegasus and Mercury computers. Most of the 32 tape characters correspond to two alternative printed characters on the teleprinter type-face, in much the same way as a typewriter prints both capital and small letters from one set of keys. These alternative states of the teleprinter are called figure-shift and letter-shift, and there are two special tape characters called by these names which set up the teleprinter. Thus, if a figure-shift character is read from a tape being printed out on a teleprinter, every symbol thereafter will be printed according to the figure-shift code, until the letter-shift symbol is read. Then the situation is reversed and everything appears in letter-shift until a figure-shift is read.

6.1.5 Figure-shift is often denoted by ϕ, and letter-shift by λ.

6.1.6 In addition to ϕ and λ there are two characters which cause no printing; they control the movement of the paper in the teleprinter. Carriage return (CR) causes the next character to be printed at the left margin of the same line; line feed (LF) causes the next character to appear on the next line. CR is almost always followed immediately by LF to ensure the next character appears at the beginning of the next line. CR and LF are available only in figure-shift.

FIG: 6.1

Character on tape	Letter-Shift	Figure-Shift	Value inside Computer
100.00	P	0	00
000.01	A	1	04
000.10	B	2	05
100.11	S	3	09
001.00	D	4	10
101.01	U	5	14
101.10	V	6	15
001.11	G	7	19
010.00	H	8	20
110.01	Y	9	24
110.10	Z	+	25
010.11	K	-	29
111.00	.	.	30
011.01	M	LF	34
011.10	N	Space	35
111.11	Erase		39
000.00	Figure-Shift		50
100.01	Q	>	54
100.10	R	⩾	55
000.11	C	*	59
101.00	T	→	60
001.01	E	(	64
001.10	F	)	65
101.11	W	/	69
110.00	X	×	70
010.01	I	≠	74
010.10	J	=	75
110.11	Letter-Shift		79
011.00	L	v	80
111.01	?	n	84
111.10	£	CR	85
011.11	0	,	89

6.2 THE TAPE CODE

The code used as between the printed symbols, the paper tape, and the values that arise when a character is read into the computer are shown in Figure 6.1. It is not strictly necessary for the user to have any knowledge of the appearance of the characters on the tape, as tape can always be printed out on a teleprinter, but in practice the user soon learns to recognise the more common characters.

6.3 THE INPUT AND OUTPUT ORDERS

6.3.1 The 71, 72 and 73 orders are provided to enable the programmer to operate the reader and punch. It will be noted from Fig.6.1 that each tape character is represented by two decimal digits in the computer; these two digits are always the two most significant in a word. The action of the three orders is as follows.

71 Read the character now over the tape reader photocells into the two most significant digits of XA, then step the tape on to the next character. The 8 least significant digits of XA are made zero.

72 Punch out the character determined by the two most significant digits of XA. The eight least significant digits of XA are irrelevant.

73 Perform a 72 order, then a 71, using the same XA.

XB and N are irrelevant in these three orders (unless more than one reader or punch are in use, see 8.11). The 73 order is in practice rarely used.

6.3.2 The input/output orders may be summarised thus -

71 $a' = (\text{TAPE})$

72 $(\text{TAPE})' = a$

73 $(\text{TAPE})' = a$, then $a' = (\text{TAPE})$.

Example 1: Read tape, ignoring blank tape (i.e. figure-shift) until a character other than blank is found. Leave this character in X9. (The code for blank tape is 50, as in Table 6.1).

	F	A	B	N	
240	71	9	0		Character to X9
241	06	9	0	500000	Subtract 50 from m.s. end
242	57	9	0	240	Return if blank
243	05	9	0	500000	Restore character in X9

Example 2: Punch CR/LF.

09	9	0	850000	Set CR code in X9
72	9	0		Punch X9
09	9	0	340000	Set LF code in X9
72	9	0		Punch X9

6.4 SPEED OF INPUT AND OUTPUT

6.4.1 As mentioned above, the tape reader has a maximum physical speed of 300 characters per second; this means that, once a tape read order has been initiated the next one cannot start until 3-1/3 milliseconds have elapsed. After reading the character the computer goes on to obey subsequent orders, but if one of these is found to be another tape read order, and 3-1/3 ms have not elapsed, the computer will be held up temporarily until the reader has completed the action of stepping on the tape.

6.4.2 A similar situation exists with the tape punch, except that the speed is 60 c.p.s. (110 c.p.s. on later models), and the minimum time between punching successive characters is consequently 16-2/3 ms. (or 9.1 ms.).

6.5 CODE CONVERSION ON INPUT/OUTPUT

6.5.1 When reading or punching decimal digits it is necessary to convert between the value inside the computer as given in Fig.6.1 and the actual value of the digit itself. This can be done very simply as follows.

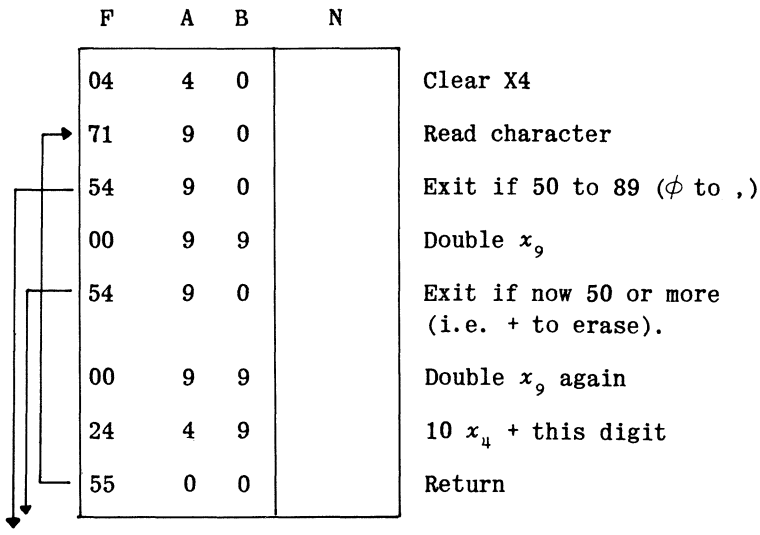
6.5.2 Input

Form 4 times the value, and take the first digit of the result, thus:-

Tape Character	Value as read	4 times Value
0	00	00
1	04	16
2	05	20
3	09	36
4	10	40
5	14	56
6	15	60
7	19	76
8	20	80
9	24	96

The digit required thus arises in the M.S.D. position, from which it can be conveniently picked off and added into the L.S. end of another accumulator.

Example: Write an input loop producing a decimal number in X4. The tape is assumed to be in figure-shift.



The loop is left whenever a character other than a decimal digit occurs. Further programme is required to deal with the various symbols which could occur. The jump addresses are left blank as the programme is incomplete.

6.5.3 Output

Form 25 times the value of the digit, plus 15; this is a three digit integer of which the top two give the correct value for sending to the punch.

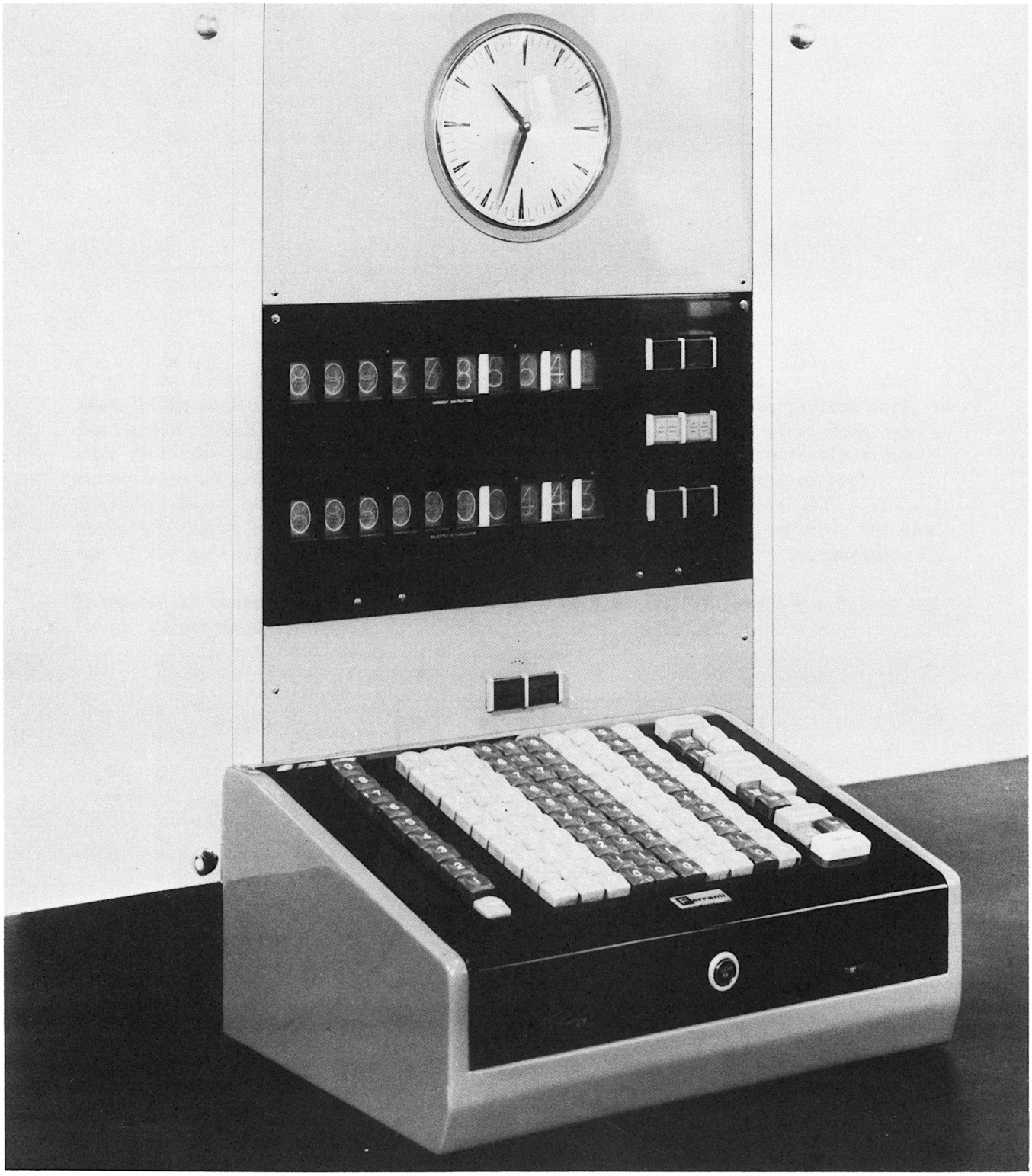


FIG: 6.2 THE CONTROL UNIT AND DISPLAYS.

Decimal Digit	25d + 15	Top two digits
0	015	01
1	040	04
2	065	06
3	090	09
4	115	11
5	140	14
6	165	16
7	190	19
8	215	21
9	240	24

It will be noted that the two digits produced by applying this rule are not always exactly the same as the values given in Fig.6.1. The rule does nevertheless work, as the series of values in Fig.6.1. has many gaps, and each value not given does lead to some character being punched when an output order is given. The apparently wrong values derived from applying the 25d + 15 rule do in fact lead to the correct character being punched. This will be more fully explained in Chapter 8. A similar situation cannot of course arise on input; whatever character is read from the tape, one of the 32 values listed in Fig.6.1. will appear in the specified accumulator.

Example: X6 contains a decimal digit, value 0 to 9 at its l.s. end. Punch this out on the output tape.

F	A	B	N	
09	9	0	025000	.025 to X9
79	6	6		25d to m.s. end of X9
05	9	0	015000	25d + 15 to m.s. end of X9
72	9	0		Punch X9

Note the position in the word in which the constant 25 has to be placed to ensure 25d appears in the desired position.

If it is desired to make efficient use of a punch with speed greater than 125 c.p.s it is worthwhile to avoid the multiplication order. See programme sheets of library subroutine S5.

6.6 CONTROLS

6.6.1 All the control buttons used by the programmer during development of programmes and normal operation of the computer are mounted on a unit which can be moved about on the desk. The layout of the buttons is shown in Fig. 6.2. There are also two rows of 10 decimal digits built into the main machine facing the operator, together with a number of lights; the two rows of decimal digits represent two computer words, and the lights indicate various states of the machine.

6.6.2 The main bank of 100 buttons, arranged in 10 columns of 10, are called the keyboard, or the handkeys; they represent one computer word, and the buttons in each column of 10 are interlocked so that only one can be depressed at a time*. There is

* Note that it is mechanically possible to have more than one button in a column down at once if more than one is pressed simultaneously. This should not be done, as it can lead to certain forbid-

a bar available to clear the whole keyboard in one operation; otherwise each column is set individually. During the course of a programme the computer may "read" the keyboard, i.e. a word corresponding to the setting of the keyboard may be obtained. There are in fact two groups of orders in which the keyboard appears as an operand, thus -

Function	Operation
15	$a' = a + (K.B.)$
16	$a' = a - (K.B.)$
17	$a' = -a - (K.B.)$
18	$a' = -a + (K.B.)$
19	$a' = (K.B.)$
35	$a' = 10a + (K.B.)$
36	$a' = 10a - (K.B.)$
37	$a' = -10a - (K.B.)$
38	$a' = -10a + (K.B.)$
39	$a' = 10a$ + M.S.D. of (K.B.) in LSD position

With all these orders XB must be 0, and N is irrelevant. (The effect of these orders if XB is not zero will be described in Chapter 8). In practice the programmer usually wishes merely to read the word on the keyboard straight into an accumulator, and so will use the 19 order.

6.6.3 The bank of buttons on the right of the control unit consists of 5 pairs and 5 single ones. The pairs are each interlocked so that one or the other must be depressed, and in effect are two position switches; the single ones are all spring loaded so that they can be used to give signals to the computer to initiate some specified action.

6.6.4 When the computer is ready for normal use by a programmer, all the five pairs should be depressed to the left. The effect of these 5 pairs is as follows.

ISOLATE 0-99/FREE 0-99

ISOLATE 100-199/FREE 100-199

These two pairs enable two sections of 100 words each of the store to be made so that no order writing into them can have any effect. These two sections taken together are used by the Initial Orders, to be described in Chapter 7, and it is desirable, especially with programmes being tried out for the first time, that whatever else they may do, they cannot accidentally destroy the Initial Orders. A correct programme will almost certainly in practice work perfectly well whichever way these buttons are set, but is nevertheless recommended that as a matter of policy, the Initial Orders should normally be kept isolated.

No K.B. WAIT/K.B. WAIT

There is a facility in the machine which causes the computer to stop whenever control reaches an address specified on the four l.s. digits of the keyboard. This is useful when developing programmes, as it enables one to run up to a certain predetermined point at full speed, then perhaps to look at various accumulators, or to see which way a critical jump order is going. Normally however, this facility is not required, so it is only operative if K.B. WAIT is depressed.

AUTO/MANUAL

If this button is in the AUTO position, the computer will obey programme from the store in the manner described in earlier chapters. It is sometimes useful however to make the

computer obey one or two orders which are not available in the store or on tape; this can be done by pressing the MANUAL button. In this state the machine will obey orders from the keyboard. This mode of operating the computer is not usually to be encouraged for the programmer, as

(a) It is very easy to make a mistake.

(b) There is no record afterwards of what has been done.

(c) Much computer time can be taken up, which may well be wasted if a mistake is made.

The AUTO/MANUAL button should only be operated if the WAIT button (see below) is down.

WAIT/RUN

This is perhaps the most important button-pair. If WAIT is depressed the machine will not obey orders, but merely wait until some action is taken by the operator. A programme which is running can be stopped at any time by pressing WAIT.

6.6.5 Of the five single buttons, only two are in common use.

CLEAR CONTROL

This is usually the first button to be operated when a programmer begins to use the machine. Its effect is merely to clear X1, the control address, ensuring that when the machine begins obeying orders, it will enter the Initial Orders; these are arranged to begin at word 0.

As it would be harmful to clear control during the operation of a programme, it has been arranged that this button is inoperative if the RUN button is down.

CONTINUE

This button has two uses. If the machine has stopped because of a 99 order or a keyboard wait, pressing this button will cause it to go on from the next order. When the WAIT button is down however and the machine not obeying programme, pressing this button will cause one order to be obeyed; this will be the next order in the programme if the machine is on AUTO, or the order on the keyboard if it is on MANUAL. Used in the second way, it may be called a single-shot button; this mode of operation can be used in developing a programme, but, like manual operation, should not be used if it can be avoided, as considerable amounts of machine time may be used to little or no purpose.

6.6.6 The effects of the other three single buttons, PRIMARY INPUT, INTERRUPT, and ACCEPT INST. are explained in **8.5**, **7.14.6** and **8.11** respectively.

6.7 MONITORING FACILITIES

6.7.1 The upper display always shows the instruction currently being obeyed. If the machine is obeying a programme it will not usually be possible to read this display as it will be changing too rapidly. When the machine is stopped for any reason this display will show the order which was last obeyed. It will be shown in modified form, i.e. the N digits will be the actual value that was used in the execution of the order, not necessarily the value written in the order.

6.7.2 The lower display can show the contents of any accumulator, as selected by pressing one of the buttons in the lefthand bank on the control unit. This column of buttons is interlocked in the same way as the columns of the keyboard. Again if the machine is running it may not be possible to read the display, but if an accumulator which is changing only rarely is selected, it may be possible. It is frequently useful to select X1, the control address, on this display.

6.7.3 The bottom button in the display selection bank is labelled DISPLAY. This really corresponds to accumulator 0 used as a destination; this can only happen when X0 is specified in the A position of an order. Thus the order

F	A	B	N
14	0	0	432

causes the contents of register 432 to appear on the monitor, provided the DISPLAY button is down. Note that the order must be obeyed at a point in time when the DISPLAY button is down; it is not sufficient for the order to be obeyed when some other accumulator is selected, and to then select DISPLAY.

6.8 INDICATOR LIGHTS

6.8.1 There are a number of indicator lights to the right of the displays. The functions of these are as follows:

Group 1 (Red)

The first four of these (the left-hand group) should all be off before normal operation of the machine begins.

Not full speed

There is a facility on Sirius to make the machine run at slower speeds than normal; this is sometimes useful during development of programmes. The facility is controlled by the left-hand knob on the front edge of the control unit, and speed can be varied from full down to a few orders a second. This light is on whenever the machine is set to run at any speed below full.

Parity Off

This light will be on if the store parity check has been rendered inoperative by a switch at the back of the machine. The facility to inhibit this check is only for use by the maintenance engineer, and the programmer should not use the machine if this light is on. (The working of the parity check is explained in **8.3.4**).

Primary Input

There are a few orders built into the machine to enable the Initial Orders to be read in. This will be described more fully later. This light is on whenever the machine is obeying these orders rather than orders from the store or the keyboard.

Margins On

This light will be on if any of the engineers marginal testing facilities are switched on either in the computer itself or in the store cabinet.

Parity Stop

This light comes on if a parity stop occurs. This indicates either a machine fault, or that the programmer has referred to a non-existent store address, or that he has attempted to read from an address not written into since the machine was last switched on. Which of these is the reason can usually be established by manual inspection of the last two or three instructions obeyed.

6910 Stop

This light comes on when the machine tries to obey a 69 order with X1 specified in the A position. This will usually be a programme fault.

It should be noted that Parity Stops and 6910 Stops can only be cleared by use of the ACCEPT INSTRUCTION button (See **8.11**).

Group 2 (White)

Input Busy

This light is on whenever the computer is calling for a character from an input channel, and is held up for it. It therefore flickers continuously during most input programmes. If it comes on steadily it usually implies a defective tape reader, or, more commonly, a programme fault leading to a non-existent tape reader being called for. This can be recognised by examining the read instruction in the upper display.

Output Busy

This behaves very similarly to the Input Busy light, indicating the state of the output channels. Note however that this light can be lit due to a programme attempting to use the punch when this is switched off.

90 Group Wait

This comes on whenever a group 9 instruction (normally 99) is encountered. The programme carries on on pressing the CONTINUE button.

Keyboard Wait

This comes on whenever the machine stops on a keyboard wait (See **6.6.4** above).

Group 3 (Green)

These six lights give information to the maintenance engineer. The light labelled OVR SET is the only one normally of any interest to the programmer. (But see **8.11.2**).

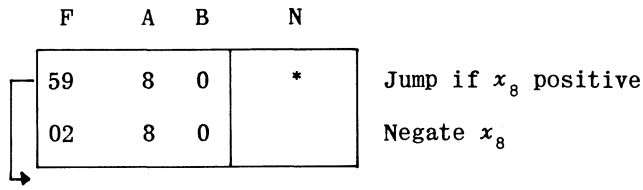
CHAPTER 7

THE INITIAL ORDERS

7.1 THE NEED FOR PARAMETERS

7.1.1 When writing any programme consisting of more than a few orders it is inconvenient to have to decide where each part of it is to be stored at the time when the orders of the programme are being written down. In these circumstances it is not possible to fill in all the addresses in the orders, as many of these depend on the actual position of the piece of programme in the store.

7.1.2 Consider for example the orders required to find the modulus of the number in X8.



The address marked with an asterisk cannot be filled in until it is known exactly where the orders are to be stored.

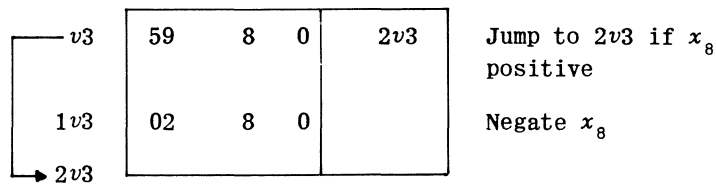
7.1.3 It is of course possible to leave blank all the addresses which cannot be filled in until the whole programme is written and all decisions on where it is to be stored have been taken. The process of filling in the correct addresses is however very tedious, and one in which it is very easy to make errors. Also, if it is later found that one section of the programme has to be lengthened by a few orders many of these addresses may have to be changed. To avoid these difficulties parameters have been introduced.

7.2 PARAMETERS

7.2.1 Parameters allow the programmer to fix on certain reference points in his programme, and to refer to these by labels. These labels will only be changed into the true addresses at the last possible moment, as the programme is read off paper tape and placed in the store of the computer.

7.2.2 Parameters in Sirius are denoted by a small letter "*v*" followed by a number. Thus the parameters available are *v*1, *v*2, etc. These can be written in the N-position of a Sirius order instead of an absolute address, and each will be changed into an absolute address, (as allocated by the programmer) as the tape is read. It is necessary to be able to refer to addresses close to one fixed by a parameter; these will be referred to by writing, for example, 2*v*1, if it is necessary to specify an address 2 further on from that known as *v*1. Note that 2*v*1 is not to be interpreted as 2 times *v*1.

7.2.3 It is now possible to rewrite the two orders above using a parameter. Suppose the address of the first order to be denoted by *v*3



7.2.4 The numbers before a parameter can be negative if required. Thus to jump unconditionally to the order 10 places before that labelled *v*4, we can write

F	A	B	N	
55	0	0	-10*v*4	Jump to -10*v*4

7.3 SETTING PARAMETERS

7.3.1 At some stage all parameters referred to on a tape have to be set to the required value. This is done by writing down an equation of the following form:

*v*3=324

This will cause the parameter *v*3 to be set equal to 324. Once this has been read an order such as

14	2	0	7*v*3

will be placed in the store as

14	2	0	331

7.3.2 It is essential that no parameter be referred to on the programme tape before it has been set by an equation such as the above. In general it is desirable for all the parameters used to be set by a series of these equations at the beginning of the programme tape. If it then becomes necessary to reallocate the store used by the various parts of the programme, only these equations, all occurring together, need be changed.

7.4 THE PURPOSE AND MODE OF ACTION OF THE INITIAL ORDERS

7.4.1 The Initial Orders are the principal means by which programmes are fed into the computer; they form a programme occupying the first 200 words of the store. This programme is normally safeguarded against accidental overwriting as described in **6.6.4.**

7.4.2 The Initial Orders fall into two fairly distinct sections; the first, usually known as INPUT, is designed to read tapes punched in a convenient code and to convert the information on them into the form required by the computer. The information punched on the tape consists mainly of orders and numbers intended to be stored; there are in addition certain "directives", which are not stored in this sense, but are used to control the operation of INPUT. The second part of the Initial Orders is the MONITOR programme; the purpose and use of this will be described later in this chapter.

7.4.3 The orders and numbers which are to be placed in the store are punched from the programme sheets in a natural way which is described in detail below. The way in which the punching is done is such that the printed version of the tape corresponds closely to the original programme sheet, i.e. each order or number will be on a separate line, and there will be spaces between different parts of orders.

7.4.4 To start off a particular programme, the tape is placed in the reader, and the CLEAR CONTROL button pressed, with the WAIT/RUN button on WAIT. Then on pressing RUN, (and also CONTINUE if the machine is already on a stop) the machine will begin to obey the INPUT programme, and this will immediately start to read tape.

7.4.5 The basic information which must always be given to INPUT on any programme tape is as follows:

1. Where it is to put the orders or numbers it is about to read.

2. When it has reached the end of a tape, and what it should then do.

7.4.6 When one order or number has been read, the resulting word is placed in the store at an address known as the "Transfer Address" (T.A.); the T.A. is then increased by one, and INPUT goes on to read the next order or number. Thus orders and numbers as read from the tape are stored in consecutive locations. It is necessary however to be able to set the T.A. at the beginning of the tape, or whenever a gap is to be left between groups of words.

7.4.7 The T.A. in Sirius is treated as parameter 0, i.e. $v0$, which can be written as just v. It can thus be set in the same way as a parameter, by an equation e.g.

$$v=220$$

This directive will cause the order or number following to be placed in word 220, the next one to 221 and so on. The only difference between ordinary parameters and the T.A. is that the latter gives the address of the word about to be stored, and is stepped on by one automatically after each word. The T.A. can therefore be referred to in orders e.g. the following order is a loop stop.

F	A	B	N	
55	0	0	v	Jump to the address of this order.

7.4.8 When allocating storage space it is necessary to allow for that used by INPUT. The programme for this always occupies addresses 0-199, and a certain working space beyond this is used by INPUT to hold the parameters. The T.A., $v0$, is held in 200, $v1$ in 201, and so on as far as necessary. The user must therefore not begin his programme until some way beyond 200; he might start at 220, or, if a lot of parameters were to be used, at 250. These particular addresses allow space for 19, or 49, parameters, as well as the T.A. If necessary, the programme can later use the space which was reserved for parameters, once it has been assembled in the store. Also, if the whole store is required, it is possible to use the 200 words which INPUT normally occupies. This is explained more fully in 8.7. It is not possible, of course, to use INPUT to read programme into locations 0-200, or into any space occupied by parameters (unless no further reference occurs on the tape to the parameters concerned).

7.5 PUNCHING OF NUMBERS

Numbers, punched as either integers or fractions, may be read in by INPUT, and will be stored in accordance with the T.A., The following are specimens of what may be punched:

On Tape	Stored in Machine
125 CR/LF	0000000125
-125 CR/LF	9999999875
0.312 CR/LF	3120000000
Sp Sp +.312 CR/LF	3120000000
-.312 CR/LF	6880000000
7654321012 CR/LF	7654321012
87654321012 CR/LF	7654321012
5000000000 CR/LF	5000000000
-0.5 CR/LF	5000000000
+0.5 CR/LF	5000000000

These do not form a comprehensive list of all possible combinations, as this would be very long indeed. They do however indicate some of the following rules.

1. All numbers must be terminated with CR/LF.
2. The + sign is optional with positive numbers.
3. Initial Spaces before a number begins (with a sign, a . or a digit) will be ignored. This makes it possible to lay out the data print-outs if desired.
4. Er (the Erase symbol) is ignored throughout except between CR and LF.
5. In addition to Space, the characters LF and CR/LF will be ignored before a number.
6. There is no check on overflow. A sensible answer cannot be obtained if more than 10 digits are punched with either integers or fractions. The fact that no warning is given of this makes it undesirable in general to use INPUT for reading large quantities of data.

7.6 PUNCHING OF ORDERS

7.6.1 The simplest type of order to punch is one in which the N address is zero. This can then be omitted and the order will consist of four digits, for example:

0490 CR/LF

The four digits must be punched without any intervening spaces and must be terminated by CR/LF. As with numbers, Erase can occur anywhere except between CR and LF, and the order may be preceded by Space, LF or CR/LF. There would be no advantage in having any spaces before an order, however.

7.6.2 If there is anything to be punched in the N position it must be preceded by Space. (This serves as a signal to INPUT that what follows is to be placed in the N position of the order; it must be remembered that this in fact occurs before the FAB when the order is assembled in the word). Thus a simple order with an N-address could be punched as:

0490 Sp 123 CR/LF

This will appear in the store as 0001230490.

7.6.3 Parameters are punched as they are written, using the small figure-shift *v*. Thus possible orders are:

0490 *v*10

0490 11*v*16

0490 -11*v*16

The last two N-addresses should be thought of as 11 + *v*16 and -11 + *v*16 of course. It is possible to punch the + sign, but this is unnecessary and contrary to convention. Parameters are sometimes required negatively and the following type of order is permissible;

0490 -*v*11

Note however that -0*v*11 is interpreted as -0+*v*11, i.e. *v*11.

7.6.4 It is occasionally useful to punch a . in an order. This occurs most frequently with 05 - 09 and 25 - 29 orders. An example is:

0950 .23

This is equivalent to

0950 230000

and appears in the store as

2300000950

7.6.5 In a few orders the N-address is made up of two, or even more, parts of the type already described. In fact, as INPUT just adds together all the parts of an order, (including FAB treated as one part), there is no limit on the number of parts than can be punched. The following are examples, showing what is stored in each case.

$v1=440$
$v6=250$

1086 .2$v1$	2004401086
2874 -$v6v1$.62	6201902874
1234 -1-$v6$.2	1997491234

Orders as complicated as the last two will be very uncommon in practice.

7.7 DIRECTIVES - WARNING CHARACTERS E, J, N, Z.

7.7.1 An equation punched on a programme tape is an instance of what is known as a "directive". The essential feature of a directive is that it is not itself stored, but gives information to INPUT concerning the programme. There are several other directives, all of which are introduced by a single letter. This letter is called a "warning character". All warning characters must be punched as:

Letter-shift, Warning character, Figure-shift.

Erase will not be ignored between the Letter-shift and Figure-shift.

7.7.2 The most important warning character is E. This is followed by an address, and instructs INPUT to enter (E for Enter) the programme already in the store at this address. This directive will thus appear on the programme tape after all the orders of the programme, usually the end of the programme tape. After reading the directive, but before obeying the first order specified, INPUT will come to a 99 wait, in order for any necessary tape changing to be done. A small example of a complete programme tape can now be given:

$v=250$
5500 v
E 250

This programme consists merely of a loop stop in word 250, but this does illustrate the basic layout of a typical programme tape.

7.7.3 The warning character J (for "Jump") has the same effect as E, except that the 99 wait does not occur. This is useful for entering what are known as "interludes". An interlude is a programme, usually very short, which is obeyed during INPUT to do some small piece of calculation which does not form part of the main programme. It is also possible to use J to enter a programme which does not read tape, or which has its data on the end of the programme tape, but in general the E-directive is the recommended way of entering a programme.

7.7.4 It is good practice for each tape used in a problem to have a name. The warning character N (for Name) is used for this. The name of the tape is punched, usually at its head, preceded by a letter N: the name is terminated by a few blanks. When the N is read by INPUT the name following it is copied on to the output tape: in this way a record is kept of the names of all tapes fed into the machine.

7.7.5 Like the other warning characters N must be punched as $\lambda N \phi$; INPUT will then read and copy the tape until a non-blank character is encountered (note that Erase is

not ignored here); it then punches this character and all succeeding characters on the input tape until it reaches two consecutive blanks, of which it punches the first. Note that N must always be followed by at least one ϕ. This means that, if the name sequence is to start in letter-shift, the input tape must be punched λ N$\phi\lambda$. Normally however the name begins with CR/LF, which must be in figure-shift.

7.7.6 For example, if the name of the tape is

LRGW 13A

the following should be punched:

λNϕ CR/LF λLRGWϕ Sp 13 λA$\phi\phi$

Two ϕ s are sufficient to terminate the name; in practice several are usually obtained by pressing the run-out key for a second or two.

7.7.7 It is often necessary to cause the machine to halt during input without entering the programme: in these circumstances E is not what is wanted, and the warning character Z has been provided. This is not followed by any address, and on being read causes a 99 wait. On pressing the CONTINUE button, INPUT will carry on reading tape.

7.8 USE OF INPUT AS A SUBROUTINE

7.8.1 It is possible to use INPUT as an ordinary subroutine during the course of a programme - e.g. to read in further numbers. INPUT can be entered by the following order:

6920 144

This sets a link in X2, and INPUT immediately begins to read tape in the ordinary way. It will return to the Master Programme when the warning character L is read from tape (L for Link). Like Z, L is not followed by an address. The T.A. (and any other parameters) can be set from the tape read by INPUT as usual or alternatively they can be set by the Master Programme before entering INPUT. The T.A. can be placed in word 200, ν1 in 201 and so on. INPUT used as a subroutine uses all the accumulators.

7.9 USE OF MORE THAN ONE READER OR PUNCH

7.9.1 It is possible to make INPUT use any reader or punch when more than one of either of these is attached to a particular machine. (See 8.11 for how extra readers and punches are called for by programmes).

7.9.2 When INPUT is entered at word 0, i.e. after pressing CLEAR CONTROL, the reader and punch used are specified by the two most significant digits of the keyboard. D_0 selects the input channel and D_1 the output channel. Thus if these two digits are clear on entry to INPUT, the first tape reader and the first punch will be used.

7.9.3 When INPUT is entered as a subroutine the selection is done in a similar manner by the two most significant digits of X2. Thus to enter INPUT to use tape reader 1 and punch 2, the following orders can be used.

0510 .12

6920 144

If INPUT is only going to be used to read in, the 2nd digit of X2 is immaterial.

7.9.4 The warning character T (T for Tape-Reader) can be used to select a new input channel at any point of a tape. T must be followed by a single digit. Thus if T2

occurs on a tape being read on tape reader 0, INPUT will begin to read on input channel 2 at this point. If later T 0 occurs on this tape, INPUT will revert to the first reader.

7.10 PRINTING OUT PARTS OF THE STORE

7.10.1 As explained in Section 6.7 it is possible to examine any word in the store by using the DISPLAY facility. This operation is not to be encouraged when there are more than one or two words to be examined, as it is relatively slow, and a proper record of what was seen cannot be guaranteed, as it is very easy to look at the wrong word, or write down the contents wrongly. A facility whereby chosen words can be printed out in a simple form has therefore been provided as part of the Initial Orders.

7.10.2 This is done by the warning character P (P for Print or Punch), followed by an address. This will cause the contents of this address to be punched out, on the selected output channel, as 10 decimal digits. As an example, if P 252 is read off the input tape, the following might be punched:

v=252

1234567890

The "v=252" gives the address and 1234567890 is the contents.

7.10.3 Alternatively P may be followed by two addresses separated by a dash (minus). Then all words between these two addresses, inclusive, are punched. Each word whose address is a multiple of 10 will be preceded by an extra CR/LF, so that the words will be laid out in blocks of 10. Thus the effect of P 257-261 might be:

v=257

1234567890
9876543210
1414141414

3213213213
1111111111

7.10.4 The method of indicating addresses has been chosen so that the output tape can later be fed back into the machine. This facility is useful when certain numbers have to be carried forward from one programme to another, to be run on a different occasion, and when the numbers in themselves are not of great interest outside the machine.

7.11 CHECKSUMS

7.11.1 The majority of errors which occur on the input of paper tape to the computer arise through the loss or gain of a single hole in the tape. The paper tape code, as shown in Figure 6.1, has therefore been chosen so that the most commonly used characters (the first 16 figure-shift characters) each have an odd number of holes. These are referred to as the "checked" characters, and have the property that it is impossible for any one of them to be changed into any other by the loss or gain of a single (or odd number) of holes.

7.11.2 In general, INPUT only accepts "checked" characters, or where "unchecked" characters are accepted, they have to be punched in combination with some other characters. Thus CR (which has an even number of holes) must always be followed immediately by LF, and Letter-shift must always have a Figure-shift exactly two characters later. Any apparently wrong combination will be rejected by INPUT, a loop stop occurring, so that there are safeguards against most reading errors.

7.11.3 There is however a further type of fault which occurs when a character on the tape is read twice, or omitted altogether. This fault is not common, but cannot usually be detected by the method just described. It is therefore desirable to have a further check on the accuracy with which any given tape has been read, and this is done by having a checksum at the end of it. INPUT always builds up a sum of everything it reads, and at the end of a tape a warning character C (C for check) is punched, followed by this sum. The action of C is to bring INPUT to a stop if the sums do not agree, and to go on reading tape otherwise.

7.11.4 If C is to be used at the end of a tape, it is necessary for the checksum to be clear at the beginning. This is done by the warning character K, not followed by an address. K should be punched at the beginning of any tape which is to have a checksum, either before or after any N-sequence which may be present. The checksum is also cleared whenever INPUT is entered via the CLEAR CONTROL button, or as a sub-routine, and when a T-directive is encountered.

7.11.5 The actual sum formed for checking is not a straight-forward sum of the words read, but depends on the number of characters and their value in a fairly complicated manner. It is not necessary however for the user to understand exactly how the checksum is formed beyond knowing that it gives a check which is effectively foolproof against any type of reading error.

7.11.6 At some stage the checksum has to be put upon the input tape. To do this the 18 symbols as follows should first be punched on it, just before the E, J or Z ending the tape.

CR/LF λCφ - φφφφφφφφφ CR/LF

This tape should be read in and will come to a loop stop after reading the above characters. The 10 digits of X4 should then be noted down, and hand-punched over the 10 figure-shifts. The tape should then be capable of being read from the beginning and passing through this sequence, and is then a checked tape.

7.11.7 The checksum facility is of course optional, but should normally be used with programme tapes which are to be used for productive work. It is not usually worthwhile to put the checksum on the tape until the programme is fully developed.

7.12 PUNCHING CONVENTIONS AND FAULTS

7.12.1 As explained INPUT is arranged to come to a loop stop if certain combinations of characters are read from tape. Such a loop stop may indicate either a tape-reader fault or a punching error. Most of the punching rules have already been given, but there are a few further points to be noted.

7.12.2 Blank tape (figure-shifts) can be read anywhere *between* words, but not within them. Once blank tape has been encountered however, INPUT will not accept any item which does not begin with CR/LF or λ. When blank tape is being read, Erase, Space and LF are ignored, but will not introduce a new word or directive. This point is important, as the omission of CR/LF before a new item after blank tape is very common error.

7.12.3 All addresses after warning characters must be terminated by CR/LF. This is not necessary with warning characters which do not have addresses. In general, directive addresses are made up of sub-items in the same way as ordinary words, except that Space does not cause what follows to be shifted up four places, and acts merely as a termination of a sub-item. Thus the following could be punched:

E.2v4 (Equivalent to Ev4, but .2 appears in X1 on entry).

E v3-v4
E 2 3 (Equivalent to E5, because of the Space between 2 and 3)

Note that - cannot be used in the P directive except to separate two addresses. Thus:

P -2v3

is not possible.

7.12.4 Spaces are not permitted immediately preceding the = symbol in an equation.

7.13 STOPS IN INPUT

7.13.1 There is one 99 wait order in INPUT, which is obeyed by both E and Z directives. This order is in 81, and the control number in X1 appearing on the monitor is therefore 82. The N-digits of the upper display show the address at which the entry is being made for E and 150 for Z. For example, if E 253 is encountered, the upper display shows:-

0002539904

7.13.2 When any sequence of characters is rejected as an error, INPUT jumps to loop stop order, 6901 -1, in location 91. This order causes a continuous note on the hooter. The M.S.D. of the upper display indicates the cause of error, as follows:-

1 Blank tape after warning character requiring address

2 Sum-check failure

3 = not preceded by v, or is preceded by space

4 CR not followed by LF

5 Warning character not followed by figure-shift

6 Figure-shift in middle of digits

7 Impermissible character

8 Digit in blank tape

9 Letter-shift or Line Feed amongst digits.

7.14 THE MONITOR ROUTINE

7.14.1 The Monitor Routine forms the second part of the Initial Orders, and is mainly for use when developing programmes. Its general mode of operation allows a programme already running on the machine to be interrupted by the operator, and for return to be made to this programme exactly where it was stopped, after the contents of various registers have been punched out or displayed.

7.14.2 The Monitor Routine is entered by pressing the button labelled INTERRUPT. It will almost immediately come to a 9 group order stop (the order is actually 9090909090 for easy recognition on the display). At this point the keyboard should be set up to call for the operations required, the digits being used as follows:

D_0 Select mode of operation (see below)

D_1 Number of punch to be used

D_2 - D_5 A main store address

D_6 - D_9 A parameter number

7.14.3 There are five modes of operation specified by D_0

D_0=0 Punch continuously starting at the specified address

D_0=1 Punch one location only, from the specified address

D_0=5 DISPLAY the contents of the specified address

D_0=8 Special monitoring (see (iv) below)

D_0=9 Return to the original programme

The address specified will usually be an absolute address, as set on D_2-D_5. In this case D_6 to D_9 must be clear. It is desired to add a parameter to this absolute value, the parameter's number should be set on D_6 to D_9. Thus $v0$ cannot be used, but there is no effective upper limit on the parameter number.

7.14.4 When the keyboard has been set up, the CONTINUE button should be pressed.

(i) If D_0=0, words will be punched out from the specified address, until D_0 is changed. Then the Monitor Routine will again come to the 9090909090 stop.

(ii) If D_0=1, one word will be punched out from the specified address, then the 9090909090 stop will occur. If the CONTINUE button is again pressed, the contents of the next location will be punched, and so on, until the keyboard is changed. If the change is just setting D_0=0, continuous punching from the address reached (i.e. somewhere beyond the address still specified on the keyboard) will begin, and (i) above applies. Any other change will cause a completely new operation to begin.

(iii) If D_0=5, the contents of the specified address will be displayed, (provided the DISPLAY button is down), then the 9090909090 stop will occur. If the CONTINUE button is again pressed, the contents of the next location will be displayed, and so on. Any change to the keyboard will cause a new operation to be initiated.

(iv) If D_0=8, control will jump to word 989 in a 1000 word store machine (in general to the address 10 words before the last word in the store). This is for use only if the user has provided a special monitoring programme of his own, to print out certain locations in a special way for example. This monitoring programme must begin in 989; this is effectively the last word of store available for ordinary use, so it will have to contain an unconditional jump to where the monitoring programme really begins. To return to the point at which interruption occurred in the main programme the following orders should be used:

```
1450 -5v9890
5500 42
```

The first order is only needed if the monitoring programme has used X5.

(v) If D_0=9, control will be returned to the original programme, with the accumulators and overflow indicator set to their values when the INTERRUPT button was pressed.

7.14.5 On entering the Monitor Routine, the accumulators (including the control register) and a word indicating the state of the overflow indicator are placed in the last 10 words of the store. This is necessary so that they can be later restored on return to the main programme, but it is often very useful to print or display them. Thus on a 1000 word store machine, the locations 990 - 999 might be punched, and would give information as follows:

990	Zero if OVR clear at point where programme was left, but all 9's if set.
991 - 999	Values of X1 to X9.

It should be noted that these 10 locations will always be overwritten once INTERRUPT is pressed.

7.14.6 The exact action of the INTERRUPT button is as follows. The contents of X1 is placed in word 1 of the store (this can be done whether or not 0-99 are isolated), and a jump to word 2 occurs. The Monitor Routine therefore begins in word 2, and the contents of word 1 are immediately moved to word 991 (in a 1000 word store machine). If the INTERRUPT button is pressed when the Monitor Routine is already in operation, all the original accumulators etc. will be lost, and it will not be possible to return to the original programme at the exact point where it was left.

7.14.7 The form of the punching done by the Monitor Routine is in general the same as that for the warning character P. Thus each new operation will cause v=A to be punched where A is the address of the word which follows, each word will appear as 10 digits, and there will be an extra line feed before each word whose address is a multiple of 10.

7.15 GENERAL DESCRIPTION OF ASSEMBLY

7.15.1 A complete programme is made up of two parts, a main programme written specially for the job, and a selection from the available library subroutines. The task of linking these parts together, both when writing the main programme and also when producing the programme tape, can involve a considerable amount of effort, and the Assembly programme has been provided to reduce this as much as possible. In particular, it makes it unnecessary for the user to decide where each individual subroutine is to be stored, or to edit the tapes of library subroutines into his own programme tape.

7.15.2 The function of Assembly is to select from the library those subroutines needed by the programme. The library subroutines are all punched on one long tape. Assembly scans this tape, selects the subroutines needed and places them in the store, the other subroutines on the library tape being read but not placed in the store.

7.15.3 In order that the master programme may refer to the subroutines selected from the library Assembly arranges to set a parameter to the address of the first location occupied by each routine.

7.15.4 The procedure usually adopted when Assembly is used is as follows:-

(a) First read Assembly programme into the store; this will occupy a certain amount of space at the top of the store.

(b) Tell Assembly which subroutines are required.

(c) Instruct Assembly to select the routines and place them in the store.

(d) Read the master programme into the store. This may, if necessary, overwrite the Assembly programme.

7.16 CALLS FOR SUBROUTINES

7.16.1 The warning character S is used to indicate to Assembly which library routines are needed by the master programme. Each S directive specifies one library subroutine number and one parameter number. It is this parameter that will be set to the address of the first location occupied by the subroutine. This directive is written as indicated below where n represents the routine number and v the parameter number.

S n - v

7.16.2 The programme tape will have near its beginning several of these directives each selecting one particular library subroutine. Each such directive is known as a Call. Assembly will not accept a Call that nominates parameter v0. There may be more than one Call for the same library subroutine but Assembly will only read one copy of it into the store. All parameters nominated will, however, be set.

7.17 PRE-SET PARAMETERS

7.17.1 A pre-set parameter is a special item of data required by some library subroutines which must be provided before the subroutine can be read into the store. Such items of data may be the number of figures to be printed after the decimal point or the number of rows in a matrix; the items required (if any) for any particular subroutine will be stated in its specification.

7.17.2 The list of pre-set-parameters should be written immediately after the Call for the subroutine concerned. No other numbers or orders may be written between Calls.

If there is more than one Call for a subroutine requiring one or more pre-set parameters the list selected will be the first list provided.

7.17.3 Each subroutine that requires a list of pre-set parameters has stored with it an optional list. This list will only be used if no list for the subroutine concerned is provided on the main programme tape. If a parameter list is provided it must be of the length required by the subroutine. It is not possible to take some parameters from one list and some from another.

7.18 LAYOUT OF A PROGRAMME TAPE

7.18.1 The main programme tape must always be read on channel 0. It will usually be headed by a name sequence, and this should be followed by a T-directive selecting the reader on which the library tape is to be read. This can be any available channel. After the T-directive will come a directive setting the T.A. for wherever Assembly is to place the subroutines, and then a number of Calls, including any necessary parameter lists. The Calls are terminated by a second T-directive, identical with the first, then the main programme.

7.18.2 A specimen layout for a programme tape using Assembly is as follows:-

```
N
SPECIMEN PROGRAMME
T5
v=250
S5 - 2
S240 - 4
2
-0.11243
T5

Main Programme.
```

On the first T5 the library tape is selected, the Assembly programme (which is at the beginning of the library tape) is read to the top of the store, and is entered. It uses the Initial Orders as a subroutine to read the directive setting the T.A. and the Calls. From these Assembly builds up a list of what subroutines are required, and holds any parameter lists read. On the second T5, control reverts to the library tape, more Assembly programme is read into the top of the store, and Assembly begins to scan the subroutines on the library tape. Those not called for are ignored, and those required are stored in accordance with the T.A. Thus in the example above either S5 or S240 will begin in word 250; which one comes first does not affect the user, but it will in fact be the one which occurs first on the library tape. The main programme can then refer to the first word of S5 as $v2$, and that of S240 as $v4$.

7.18.3 On a Sirius with only one tape reader, the same procedure is effectively followed, but a certain amount of tape-changing is necessary. The Assembly programme must be read in first. Then must come the main programme tape. The first T5 will cause a 99 wait (T followed by a number is arranged to be equivalent to Z on a machine with only one input channel); at this wait press the CONTINUE button. Assembly will then read the Calls, and wait on the second T. At this point insert the library tape, and CONTINUE. A further wait will occur when all the subroutines required have been selected, and the main programme can then be read.

7.18.4 Whilst Assembly is operating a certain amount of printing will normally take place. Each time a required subroutine is formed the following will be printed:-

S n - v A

n is the number of the subroutine, v the number of the nominated parameter, and A the address of the first word to be occupied by the subroutine (i.e. the value of the

nominated parameter). The list of subroutines will be preceded by the word ASSEMBLY, and the date on which the library tape was compiled (this identifies the library tape used). This printing will only take place if the least significant digit of the keyboard is in the range 0 to 4.

7.19 SUMMARY OF THE EFFECTS OF THE WARNING CHARACTERS

7. The effects of all the warning characters are summarised below. An * indicates that an address or number must always follow the character.

C* Introduces a checksum.

E* Wait, then Enter programme.

J* Enter programme (as E, but no wait).

K Clear checksum.

L Obey Link. Should only occur on tapes being read by INPUT as a subroutine. Otherwise a jump will occur to the address which was on the keyboard when INPUT was last entered via CLEAR CONTROL.

N Introduces Name.

P* Punch out from one or more storage locations.

S* Call for a library subroutine. This should only occur when INPUT is being used as a subroutine by Assembly. Otherwise it has the effect of L.

T* Select specified input channel. (Equivalent to Z in a Sirius with only one input channel).

Z Wait.

7.20 SIZE OF STORE

7.20.1 It is often necessary for a programme to take into account the size of the store of the machine on which it is being run. For example it might wish to use some locations at the end of the store as a dump, to leave the maximum space for data. To avoid the need to have different sets of programme tapes for machines with different sizes of stores, location 90 always holds an integer giving the size of store for the machine concerned. Then only the Initial Orders tape has to be different for different machines, as programmes can refer to this word as necessary. The reference can be direct, but it is usually more convenient to do it by writing $v9890$, which effectively causes the contents of 90 to be used as a parameter.

7.20.2 An example of this technique occurs above, in **7.14.4**, where the order

1450 -5v9890

occurred. On a 1000 word store $v9890$ is 1000, so the address in the order is 995, but on a 4000 word store it would be 3995, $v9890$ being 4000. Thus in each case the last word but 4 in the store is obtained.

7.20.3 The last ten words of the store are regarded as being available for use by interludes etc., in particular they are used by the Monitor Routine. It is therefore advisable for the ordinary programmer to refrain from using them if possible.

CHAPTER 8

FURTHER FEATURES OF THE COMPUTER

8.1 REASONS FOR THIS CHAPTER

There are certain features of Sirius which are only of use in special circumstances, and to avoid the main part of this manual being unduly long, they have been left until this point. This Chapter then can be omitted at a first reading, or indeed left entirely to be used only for reference purposes.

8.2 GAPS IN THE ORDER CODE

8.2.1 It will have been noticed that of the 100 possible functions 0-99, by no means all have been covered. All the unmentioned functions do in fact have some effect, as follows.

15 $a' = a + b$
16 $a' = a - b$
17 $a' = -a - b$
18 $a' = -a + b$
19 $a' = b$

35 $a' = 10a + b$
36 $a' = 10a - b$
37 $a' = -10a - b$
38 $a' = -10a + b$
39 $a' = 10a$ + MSD of b in LSD position

8.2.2 These orders however differ in two ways from most others. Firstly, if B = 0, the number on the keyboard is obtained, and not just zero. Thus with B = 0 they are a special case, as described earlier in **6.6.2.** Secondly these orders can never set the overflow indicator, whatever the result of the operation performed. If B is not zero, all the effects of these orders can be obtained by using the corresponding 00 - 04 and 20 - 24 orders with N = 0, apart from the fact that these latter orders can set overflow. The 15 - 19 and 35 - 39 orders with B not zero, are thus not often required, and are only used in special circumstances. One reason for using them is obviously when overflow indicator setting is not wanted; another is when the N digits of the order are to be used for some purpose unconnected with the order e.g. to form a constant in which only the first six digits matter. Then the fact that these orders ignore N is useful.

8.2.3 41, 42 and 43 have exactly the same effect as 40, and 46, 47 and 48 are the same as 45.

8.2.4 61, 62 and 63 are the same as 60. A full explanation of the 65, 66, 67 and 68 orders will be given later in the Chapter.

8.2.5 The 74 gives what is known as half-signed multiplication. That is, although b is treated as a signed number, x_9 is taken as being a positive number in the range 0 to 1. If the number in X9 is in the range 0 to 0.4999999999, the 74 order will give exactly the same answer as the 79 order. The difference between the orders is best shown by an example, as follows. Suppose we have .2000000000 in X8 and .7000000000 in X9. If we multiply these using a 79 order the number in X9 is taken to be -0.3, by the sign convention, and the product is 0.2 × -0.3 = -.06, and the answer appears in the computer as

9400000000000000000

If however we use a 74 order the number in X9 is interpreted as +0.7 and the product is

$$0.2 \times 0.7 = 0.14, \text{ appearing as}$$

1400000000000000000

8.2.6 The 74 order is of use in double and multi-length working, e.g. if a double-length number is to be multiplied by a single-length one, the least significant half of the double-length number is effectively unsigned, and an order which treats it as such is useful.

8.2.7 76, 77 and 78 are the same as 71, 72 and 73 respectively.

8.2.8 All the 8 group orders are dummies.

8.2.9 All the 9 group are the same, having no effect other than causing a wait:

8.2.10 In general the user should avoid the completely redundant orders described, and use only the recommended one, as the redundant ones are liable to be changed into something else if modifications are made to the machine.

8.3 DIGIT REPRESENTATION

8.3.1 For all ordinary purposes the programmer can look on the decimal digit as being the basic unit of the Sirius word; there are however some special circumstances in which a knowledge of how this digit is represented inside the machine is useful.

8.3.2 The Sirius word consists of 40 binary (scale of two, rather than denary, scale of ten) digits, usually called bits. These are to be looked on as 10 groups of 4, each representing one decimal digit. Four binary digits however give rise to 16, i.e. $2 \times 2 \times 2 \times 2$, possibilities whereas only 10 are required. The values of these four bits are, in order 5, 4, 2, 1 and this leads to the following code.

0	0000		
1	0001		
2	0010		
3	0011		0101
4	0100		0110
5	1000	Unused	0111
6	1001		1101
7	1010		1110
8	1011		1111
9	1100		

8.3.3 The 5, 4, 2, 1 code is of course potentially ambiguous, in that for example 5 can be represented in two ways, 0101 and 1000. This difficulty is avoided by applying the convention that the 5 bit is used in the representation of any digit greater than or equal to 5. The six unused combinations are called trans-decimal digits; they can only be introduced from the keyboard (see footnote to **6.6.2**) or by machine malfunctioning. In the latter case a parity failure (see **8.3.4** below) will probably occur at the same time.

8.3.4 With each 40 bit Sirius word in the main store is associated a further bit, called the *parity bit*. This bit is not available to the programmer in any way, but it is used to give a check against any malfunctioning of the machine which might lead to bits changing in the store. Whenever a 40 bit word is written into the main store, the machine also computes whether the number of one bits in this word is odd or even; if odd the associated parity bit is made zero, if even it is made one. Whenever a

word is read from the store, the machine checks that the total number of one bits in the 40 bit word and the parity bit taken together is odd; if it is not the machine stops with the PARITY STOP light on. The check is also applied when writing into the store, on the word which is already there.

8.3.5 This check in practice is a very powerful check on the correct working of the store; as it is completely automatic it does not affect the writing of programmes in any way.

8.4 TIMING

8.4.1 Sirius is a serial machine, that is the 40 bits of a word pass through the arithmetic unit one at a time in sequence. The speed of the machine is basically determined by the frequency with which this occurs, and in Sirius this is once every two millionths of a second. A word therefore passes through the machine in 80 millionths of a second, or 80 micro-seconds, written as either 80μs or .080 ms. (ms. for milliseconds, or thousandths of a second). 80μs is the basic unit of time in the machine as far as the programmer is concerned, called the word-time. The basic frequency is usually expressed by saying the machine works at ½ megacycle, or 500 kilocycles, a second.

8.4.2 We now have to consider the timing of the store. The 9 accumulators 1 to 9 are stored in delay lines each with a cycle time of one word-time. This means they are effectively available at any time, or have zero access time. The main store however consists of delay lines each holding 50 words, cycling in 50 word-times, i.e. 4 ms. Thus any particular word in the main store is available only once in every 50 word-times. Now when the machine is obeying programme from the store, orders will normally come from consecutive addresses in it, and it is clearly desirable that as far as possible orders become available at exactly the time required. The majority of orders in fact require 3 word-times, or .240 ms. to be obeyed, and to conform with this it is arranged that addresses in the long delay lines are staggered at intervals of 3 word times. Thus the addresses in the first long line begin -

0
17
34
1
18
35
2
19
36
3
.
.

coming back to 0 again after a total of 50 addresses. So if the machine obeys an order out of word 0 taking 3 word times, the next order from word 1 is available just as it is required and no time is lost.

8.4.3 The orders taking 3 word times in all circumstances are 00 - 09, 15 - 29, 35 - 50 and 65 - 68. The other orders fall into several groups for timing purposes, and will be dealt with separately.

8.4.4 The orders 10 - 14, 30 - 34, 60 and 64 all require access to a storage location specified in the order. Such an order cannot therefore be executed until the required address becomes available. This can turn up anything from 0 to 49 word-times later. It it comes up in 0 word-times i.e. at the instant it is required, the order will take 3 word-times in the same way as the orders described in **8.4.3.** In general however a certain time will elapse before the address is found, and the order will then be completed. By this time however the next order will no longer be coming available at the instant it is required, so the machine will have to wait for this. The consequence is in fact that in 49 cases out of 50 these orders will take 53 word-times rather than 3, i.e. 4 ms. extra should be allowed for access time.

8.4.5 In general the 4 ms. access time to the store cannot be avoided, but in some circumstances it is possible to arrange that the number being extracted is in exactly the right timing position. The access is required in all cases at the third of the three-word times, e.g. an order in location 0 will be obeyed in 3 word-times if (and only if) the location to which access is required is in 34, or 84 or 134 etc. (As all the long lines are in phase addresses differing by 50 all become available at the same time).

8.4.6 The jump orders (51 to 59 and 69) require 3 word times like "ordinary" orders (i.e. those described in **8.4.3.**) if the jump is *not* made. If it is made, i.e. if the necessary condition is satisfied, access is required to the address in the main store which is being jumped to, but unlike the orders discussed in **8.4.4**, there is no question of continuing with the orders subsequent to the jump order. Thus anything from 0 to 49 word-times can be lost with effective jump orders.

8.4.7 The time loss with an effective jump will only be zero if the jump is to an order whose address differs from that of the jump order by one, plus a multiple of 50. For example, a jump order in location 23 can jump without loss of time of 74, or 124 etc. If a jump is made to an order a few locations further on (up to 16 in fact) the time will be the same as if all the intervening orders had been obeyed as if they were ordinary orders. If the jump is a few locations back, as occurs in a loop of programme, the effect is that the loop requires 4 ms. (plus allowance for any accesses etc. involved in the orders in the loop). This is true for a loop of up to 16 orders, including the jump. To obey 17 orders requires a minimum of 51 word times, so they cannot be executed in one cycle time of a long line.

8.4.8 The three orders which operate external devices, 71, 72 and 73 take 3 word times in principle, but in fact times here are usually governed by the maximum physical speeds at which the device can move. Thus if two 71 orders occur close together after a period during which no reading of the tape has occurred, the first will take three word-times, but the second may be reached before the tape has finished moving on to the next character position. When this has occurred the order will be obeyed, but by this time the following order is no longer immediately available and at least 4 ms. will be lost. The situation is similar with tape punch orders.

8.4.9 The time taken for multiplication depends on the actual number in X9. If then A is the sum of the digits of this number (e.g. if the number is 0001234567, A is 28), the time the machine actually spends multiplying is given by

$$\text{Time} = .160\ (A + 11)\ \text{ms.}$$

The time lost to the computer however depends on a similar consideration to that involved in the store access type of order, and as the machine must ultimately return to the following order, the time must be 3 word-times, plus a multiple of 4 ms. Consideration of the above formula then shows that multiplication must take 4, 8, 12 or 16 ms. over and above the usual 3 word-times. For small positive numbers the time will be 4 ms. and it is probable that the majority of multiplication orders will not exceed 8 ms. as bearing in mind that numbers have to be scaled anyway, it is not unusual to find the first few decimal digits of a number are zero.

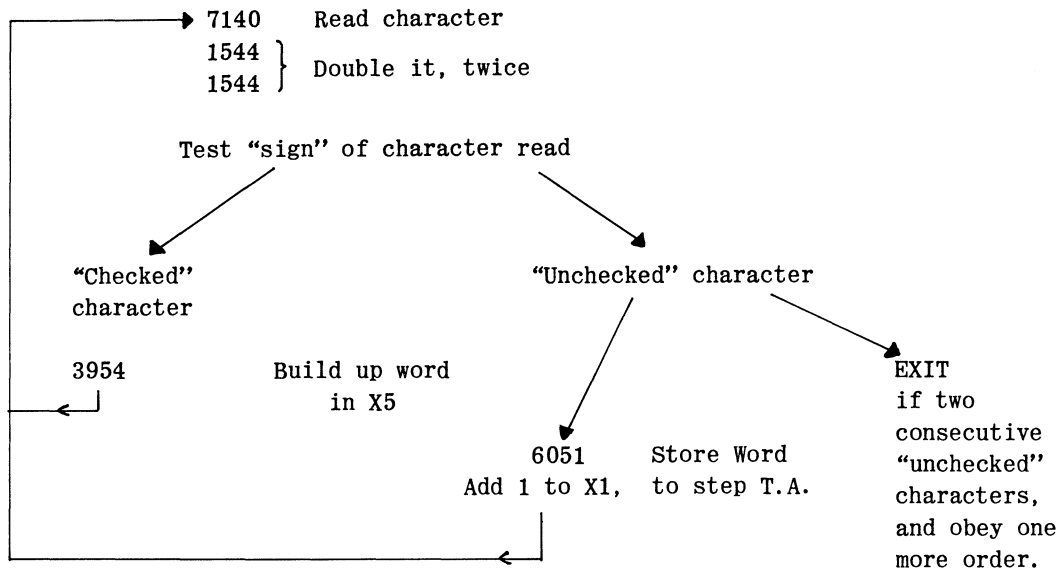
8.4.10 As the time depends only to the contents of X9, it is always advantageous to place the number likely to be the smaller of the two factors in X9, but if nothing is known about the relative magnitudes this cannot be done.

8.4.11 The time for division depends entirely on the digits of the quotient, and the formula is exactly the same as for multiplication. All the other considerations mentioned in **8.4.9** apply.

8.5 PRIMARY INPUT

8.5.1 The nickel delay line store is volatile, that is it will not hold information when the machine is switched off. The primary input facility has been provided to

enable the Initial Orders to be put into the machine, and consists of a few orders which are permanently wired in, as follows.



8.5.2 The "specification" of primary input is thus as follows.

> "Checked" characters (i.e. 0 to Erase, including all the decimal digits) are converted and a word built up in X5. This word is stored whenever an "unchecked" character (figure-shift to,) is read, according to a transfer address in X1. This is then stepped on by 1. If a second unchecked character is now read the process terminates, and the machine reverts to normal operation (depending on the setting of the MANUAL/AUTO and WAIT/RUN keys), otherwise a further word begins to build up.

Notes

1. During primary input X1 is unchanged other than when an unchecked character is read.

2. X5 is never cleared, so it is essential to have 10 characters in each word to be stored.

3. Erase is not ignored. Together with +, -, ., LF, and Space it is "converted" into a decimal digit and stored. If an error is made in preparing a tape for primary input the recommended action is to start that particular word again (after first back-spacing and erasing the error if it is an unchecked character). The effect of this when the tape is read is that the wrong characters are pushed off the top end of the word, and only the last ten digits punched remain.

4. The usual form of punching for primary input is a string of ten digit words each followed by CR/LF. The CR acts as the unchecked character terminating the word, and the LF is a checked character which gets pushed off the top end of the following word.

5. Primary input tapes must not be inserted on initial blank tape. A string of LF's is recommended, followed immediately by the first digit of the first word.

6. Any two consecutive unchecked characters can be used to terminate primary input. It is suggested that two arrow symbols be used as standard.

7. It is recommended that the machine be set to MANUAL and WAIT for primary input. In these circumstances one order from the keyboard will be obeyed at the end of primary input. If the machine is on AUTO and RUN it will enter programme just beyond the last word read in, and this may lead to chaotic effects. The keyboard should be set up as a useful order (such as an entry to the programme), or a harmless

8. Primary input is operated by setting the required initial transfer address (if reading in the Initial Orders this is zero, and so can be set by the CLEAR CONTROL button, otherwise X1 must be set to the required address by a manual order), then pressing the PRIMARY INPUT button. On completion of reading the tape, X1 should show the address following the last word read in, e.g. if the Initial Orders are being read to 0 to 199, X1 should contain 200 when the end of the tape is reached. This point should always be checked, as it is in fact a powerful check on the correctness of the reading of the tape. (Since any "single-hole" error changes a "checked" into an unchecked character or vice versa, it will cause one too few or too many words to be stored. Thus there must be compensating errors for the reading to be in error and yet X1 to be correct. As these are very rare in practice, the check is a powerful one).

8.5.3 The method of reading in the Initial Orders by primary input is therefore as follows.

1. Ensure the machine is set to MANUAL and WAIT, the keyboard is clear, that K.B. WAIT if off (otherwise a keyboard wait occurs before reading tape) and that 0-99 and 100-199 are free.

2. Press CLEAR CONTROL, insert Initial Orders tape in reader on LF, and press PRIMARY INPUT button.

3. The tape will now be read and the primary input light will be on, until almost the end of the tape when a stop will occur. At this point X1 should show 200. If not repeat.

4. If X1 is correct set both isolation buttons and the short programme at the end of the Initial Orders tape should be read in the ordinary way. This forms a sum check of the Initial Orders, and if they are correct, prints out "The Initial Orders are in". If this does not happen some error has occurred and should be investigated. If it does the machine is ready for ordinary work.

Note

If this operation is carried out with either or both isolation buttons set to ISOLATE, the computer will go through the motions of reading the tape, but the information read will not be stored. This feature applies generally; if an order 6020 150 for example is obeyed when 100-199 are isolated, the machine will not stop or give any alarm signal but will go on without having obeyed the order.

8.6 USE OF X1

X1 always contains the address of control in its l.s. 4 digits, but the m.s. 6 may be used for any purpose to which the programmer is able to put them. The following points should be noted.

1. An order such as 0410 N is effectively an unconditional jump to N.

2. An order such as 0510 N has no effect on control, as only the m.s. 6 digits of X1 are affected. 0910 N would however clear the l.s. 4 digits of X1, and thereby jump to word 0 in the Initial Orders.

3. On effective jump orders, the *whole* contents of X1 are changed (to N + b).

4. When an order is obeyed, as far as the programmer is concerned X1 contains the address of the *next* instruction.

5. If X1 is used as a modifier in a jump instruction a "relative jump" facility is obtained. Thus the order 5501 1 always jumps two instructions further on, and 5501 -1 is a loop stop. This is true wherever these orders are in the store, and this is an alternative to using preset parameters when writing programmes, such as general purpose sub-routines to be obeyed anywhere in the store. Each of these methods has advantages according to circumstances.

8.7 PROPERTIES OF THE STORE ADDRESSING SYSTEM

8.7.1 In a Sirius with just 1000 words of store, the addressing system only takes note of 3 decimal digits. This has the effect of making the store circular as it were; e.g. if an address 1002 is specified anywhere, word 2 is obtained. This is very convenient in practice, as if a programme requires more than 800 words of store, but less than 1000, it is possible to read the programme in starting at 200, to refer to addresses up to 1199, then to free locations 0-199 before running the programme. The machine then uses 0-199 for 1100-1199. The programme will of course destroy the Initial Orders but these can be put back very quickly later.

8.7.2 In a Sirius with more than 1000 words of store, the store is circular with respect to 10000. Thus in such a machine address 10243 will be interpreted as 243. If the amount of store present is less than 10000, reference to non-existent locations will lead to a parity failure, so that the programmer has an immediate warning if he attempts to refer to non-existent store.

8.8 THE COLLATE ORDERS

8.8.1 So far the 66 and 68 orders have been mentioned. 65 and 67 can be paired with these, i.e. 65 and 66 use (N + b) as a mask, and 67 and 68 use (N + b) $\times 10^4$.

8.8.2 The four orders in fact work up bit by bit from the l.s. end through each digit of a, the rule being that with the 66 and 68 orders bits are rejected from the operand (a) until a "one" bit is found in the corresponding position of the mask, and are thereafter retained; with 65 and 67 bits are retained until a one is found, and then rejected. This is done for each digit. The results clearly depend on the way digits are represented by bits in Sirius and can best be shown in a table.

Digits appearing in result for 66 and 68 orders

"Operand" digit (from A)	"Mask" digit (from N + b) 0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	1	0
2	0	2	2	2	0	0	2	2	2	0
3	0	3	2	3	0	0	3	2	3	0
4	0	4	4	4	4	0	4	4	4	4
5	0	5	5	5	5	5	5	5	5	5
6	0	6	5	6	5	5	6	5	6	5
7	0	7	7	7	5	5	7	7	7	5
8	0	8	7	8	5	5	8	7	8	5
9	0	9	9	9	9	5	9	9	9	9

Digits appearing in result for 65 and 67 orders

"Mask" digit (from N + b)

"Operand" digit (from A)	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	1	1	0	1	0	1
2	2	0	0	0	2	2	0	0	0	2
3	3	0	1	0	3	3	0	1	0	3
4	4	0	0	0	0	4	0	0	0	0
5	5	0	0	0	0	0	0	0	0	0
6	6	0	1	0	1	1	0	1	0	1
7	7	0	0	0	2	2	0	0	0	2
8	8	0	1	0	3	3	0	1	0	3
9	9	0	0	0	0	4	0	0	0	0

8.9 INPUT/OUTPUT CODES

8.9.1 To appreciate how the machine codes for tape characters arise it is necessary to consider the decimal digit representation. We shall describe the internal Sirius process in terms of an input instruction.

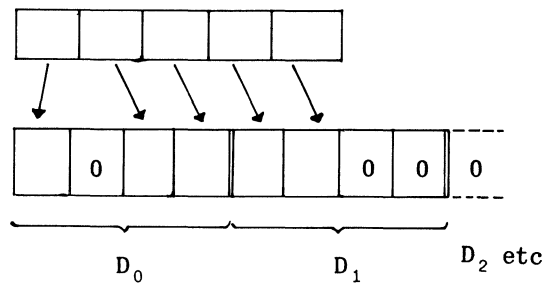
8.9.2 The five channels read from the tape provide 5 bits, a one for a hole, a zero for no hole. The 32 possibilities are as shown in the left column of Fig. 6.1. Adjustment is then made to this 5 bit binary number as follows.

1. If the number of one bits is odd, the m.s. bit (if any) is removed from the character. (All the 16 so-called checked characters have an odd number of holes).

2. If the number of one bits is even, the m.s. bit is made one, if it is not one already. (The 16 unchecked characters are all of this form).

The result of this adjustment is a rather different column of binary numbers, and in fact they correspond to 0-31 in order. These five bits are placed into five bit positions of the specified accumulator as shown.

Tape character
(as adjusted)



8.9.3 Consideration of this diagram will show that any decimal digit except 4 or 9 can occur in the first position, but only 0, 4, 5 or 9 in the second position. In fact the machine codes shown in Table 6.1 are generated.

8.9.4 On output the same correspondence is used, with the adjustment of the m.s. bit reversed. The two l.s. bits of the second character are not however used, so it is possible for them to have any values. Thus the exact codes of Table 6.1 are not necessary for correct punching, providing the first six bits are correct. The

second bit of the first character is not punched, but it is not ignored by the mechanism which adjusts the m.s. bit. It must therefore be zero.

8.10 BEHAVIOUR OF X0

8.10.1 When 0 appears in the B position of the order, zero is obtained except for orders 15 - 19, 35 - 39, 70, 74, 75 and 79; for these b is the number set up on the keyboard when the order is obeyed. B is only used as a source in Sirius, so there can be no question of what happens to numbers sent to B.

8.10.2 The situation is more complicated if A is 0. A equal to 0 as a source gives the number on the keyboard. Thus the order 6000 250 writes this number into location 250. This is not often required, and the more common way of obtaining the keyboard is by the order 1920, placing the number into accumulator 2. As a destination, A equal to 0 is DISPLAY, i.e. the result of the operation carried out by the order appears on the lower display if the button labelled DISPLAY in the left hand column of ten is down. Using A equal to 0 as a destination also gives a single pip on the hooter; by varying the lengths of loops in which this occurs notes of various pitches can be obtained. The volume of the hooter can be controlled by the right hand knob on the front edge of the control unit.

8.10.3 Thus the following orders have effects as indicated.

0000		Add the number on the K.B. to zero and send to DISPLAY (i.e. display (K.B.))
0000	123	Display (K.B.) + 123
1000	234	Display (K.B.) + contents of register 234
1400	330	Display contents of register 330
5700	400	Jump to 400 if (K.B.) is zero.

The first four of these also give a pip on the hooter.

8.11 THE ACCEPT INSTRUCTION BUTTON

8.11.1 It is a useful operation to change the contents of a given register. This can of course be done by punching a piece of paper tape, to be read by INPUT, and this is the recommended procedure, but it is sometimes justifiable to do it manually, using the facilities of the keyboard.

8.11.2 Using the fact that XA = 0 as a source gives the number on the keyboard, the order 6000 N can be obeyed manually to change the contents of address N. As it stands however it is useless as this order itself has to be on the keyboard as it is obeyed, and in general this will not be what the operator wishes to place in address N. To get around this difficulty the ACCEPT INSTRUCTION button has been provided. The machine must be on MANUAL and WAIT, and the instruction 6000 N, where N is the address whose contents are to be changed, set up on the keyboard. On pressing ACCEPT INSTRUCTION, this order goes into the order register, and will appear on the upper display. The engineers light labelled "C" will also come on at this point. The order however will not have been obeyed, so the required new contents for N can now be set on the keyboard. On pressing CONTINUE, the order will be obeyed and the contents of the keyboard will go to N. The "C" light will go off as CONTINUE is pressed.

8.11.3 Any instruction can be obeyed in this way, but there is no purpose in doing so in general. The ACCEPT INSTRUCTION button is also used for releasing the machine from

certain inoperable states which can arise from various forms of misoperation. Such states are usually recognised by the fact that the machine cannot be made to obey instructions. If such a state arises, the machine should be set to MANUAL and WAIT, and the keyboard must be clear, or have some harmless instruction on it. Pressing ACCEPT INSTRUCTION followed by CONTINUE should make the machine workable. If not there is a machine fault.

8.12 USE OF MORE THAN ONE READER OR PUNCH, AND DIFFERENT TAPE WIDTHS

8.12.1 If more than one reader is attached, the readers will be called 0, 1, 2 etc. The one which is used by a particular 71 order then depends on the N digits of the order. Thus 7120 1 will read from tape reader 1. The order is subject to modification in the usual way, so if X3 contains 1, 7123 will read from tape reader 1. It is thus possible to write a piece of input programme capable of using any tape reader, depending on the contents of a specified accumulator.

8.12.2 The same applies if there is more than one punch; the punch used is specified by the $(N + b)$ with the 72 order.

8.12.3 In practice Sirius has two input sockets. Channels 0 to 4 are on one socket, and 5 to 9 on the other. If more than two input devices are attached, some form of branched cable or junction box is required, but if there are just two, one can be plugged directly to each socket, and their numbers will be 0 and 5. The situation is the same with output devices.

8.12.4 It is occasionally necessary to use paper tape with more than 5 tracks on Sirius. The TR5 readers as normally supplied can in fact cope with up to 8 tracks, and are adjustable for 5, 7 or 8 tracks. The full codes which can be generated will not be given here, but the user can work out what happens by considering the diagram in **8.9.2** together with the following information.

1. The "3-hole" side of the tape is always in the same position. Therefore the locations of the first five tracks of any width of tape, starting from the "3-hole" edge, can be deduced from the diagram.

2. Continuing across the tape, the 6th, 7th and 8th tracks are directed into bit-positions as follows:

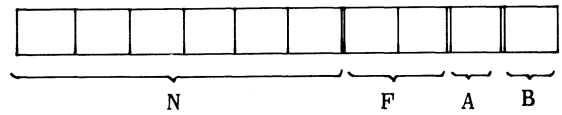
6th Track	"4" bit of D_2
7th Track	"5" bit of D_2
8th Track	"1" bit of D_1

It is thus apparent that any possible 7-track tape can be read, but 8-track tape is subject to the limitation that the 8th track cannot have a hole at the same time as the 4th and 5th tracks. This is so with most 8-track codes in use.

8.12.5 The user of tape with more than 5 tracks should note that the adjustment described in **8.9.2** is still carried out, and this must be allowed for in determining the internal representation of any tape code. In particular, with 7-track odd-parity tape, all characters coming in should be positive, and this fact can be used to check the accuracy of reading of such tape.

FERRANTI SIRIUS COMPUTER BASIC ORDER CODE

ORDER STRUCTURE
10 DECIMAL DIGITS



N is a main store address or a constant, with its sign extended if necessary.

A and B are accumulators

n, a, b are contents of N, A, B respectively

a' is the contents of A after the operation

In all orders up to 69, b is added to N before the order is obeyed. This is the basic order code, and only includes the commonly used functions.

00 $a' = a + N$ 01 $a' = a - N$ 02 $a' = -a - N$ 03 $a' = -a + N$ 04 $a' = N$	20 $a' = 10a + N$ 21 $a' = 10a - N$ 22 $a' = -10a - N$ 23 $a' = -10a + N$ 24 $a' = 10a$ + M.S.D. of N
05 $a' = a + 10^{11}N$ 06 $a' = a - 10^{11}N$ 07 $a' = -a - 10^{11}N$ 08 $a' = -a + 10^{11}N$ 09 $a' = 10^{11}N$	25 $a' = 10a + 10^{11}N$ 26 $a' = 10a - 10^{11}N$ 27 $a' = -10a - 10^{11}N$ 28 $a' = -10a + 10^{11}N$ 29 $a' = 10a$ + M.S.D. of $10^{11}N$
10 $a' = a + n$ 11 $a' = a - n$ 12 $a' = -a - n$ 13 $a' = -a + n$ 14 $a' = n$	30 $a' = 10a + n$ 31 $a' = 10a - n$ 32 $a' = -10a - n$ 33 $a' = -10a + n$ 34 $a' = 10a$ + M.S.D. of n
40 $a' = (a + 5)/10$ Arithmetical Shift down (Rounded). 44 $a' = a/10$ Arithmetical Shift down (Unrounded). 45 $a' = (a + 5)/10$ + L.S.D. of N (Rounded), 49 $a' = a/10$ + L.S.D. of N (Unrounded).	
50 Dummy 51 Jump to N if M.S.D. of $a \neq 0$ 52 Jump to N if $a \neq 0$ 53 Jump to N if OVR set 54 Jump to N if $a < 0$	55 Jump to N unconditionally 56 Jump to N if M.S.D. of $a = 0$ 57 Jump to N if $a = 0$ 58 Jump to N if OVR clear 59 Jump to N if $a \geqslant 0$
Orders 53 and 58 clear the OVR	
60 $n' = a$ 64 $n' = 0$ 66 $a' = a$ & N 68 $a' = a$ & $10^{11}N$ 69 $a' = x_1$ and jump to N 99 Wait	70 $x_9' =$ quotient, $a' =$ remainder on dividing (a, x_9) by b. Unsigned 71 a' = TAPE 72 (TAPE)$' = a$ 73 (TAPE)$' = a$ and a' = TAPE 79 $(a,x_9)' = b \times x_9$

Accumulator 1 is the control register and contains the address of the next instruction.

Accumulator 0 = 0 when used as B, = keyboard or display when used as A (with some exceptions).

(This Library Document (LD11) supersedes List CS 244).

Ferranti Ltd

COMPUTER DEPARTMENT

London Computer Centre:

68 NEWMAN STREET, LONDON, W.1

Telephone MUSeum 5040

and

21 PORTLAND PLACE, LONDON, W.1

Telephone LANgham 9211

Offices, Laboratories, and Works:

WEST GORTON, MANCHESTER, 12.

Telephone EÄSt 1301

Research Laboratories:

WESTERN ROAD, BRACKNELL, BERKS.