

A GUIDE TO PROGRAMMING  
THE NATIONAL-ELLIOTT 803  
ELECTRONIC DIGITAL COMPUTER



COMPUTING DIVISION

**A GUIDE TO PROGRAMMING**

**THE 803**

**ELECTRONIC DIGITAL COMPUTER**

**5th Edition**

**June 1962**

**Copyright Reserved**

**Elliott Brothers (London) Ltd., Computing Division,  
Elstree Way, Borehamwood, Hertfordshire, England.**

**NCR Electronics, National Cash Register Co. Ltd.,  
206-216 Marylebone Road, London, N.W.1.**

The information contained in this guide is issued for instructional purposes only, and does not constitute a specification of the equipment described. The main part of the text is written with reference to the basic 803 computer, while an appendix describes the Automatic Floating-point unit.

## **CONTENTS LIST**

### **CHAPTER 1 INTRODUCTION**

	<b>Page</b>
1.1 General	1
1.2 The Component Parts of the Computer	1
- Figure 1. Theoretical Block Diagram	2
1.3 Form of Instructions	3
1.4 Transfer Instructions	4

### **CHAPTER 2 REPRESENTATION OF DATA AND INSTRUCTIONS**

2.1 General	7
2.2 Representation of Numbers within the Computer	7
2.3 Scaling	9
2.4 The Overflow Indicator	13
2.5 The Telecode	14
2.6 Representation of Instructions	15
2.7 Store Parity Checking	16

### **CHAPTER 3 THE INSTRUCTION CODE**

3.1 Group 0	17
3.2 Groups 1 to 3	18
3.3 Group 4	18
3.4 Group 5	21
3.5 Group 6	26
3.6 Group 7	27

<b>CHAPTER 4</b>	<b>B-LINE MODIFICATION AND SUBROUTINES</b>	<b>Page</b>
4.1	B-Line Modification	31
4.2	Subroutines	33
<b>CHAPTER 5</b>	<b>INPUT ROUTINES</b>	
5.1	The Initial Instructions	36
5.2	The Translation Input Routine, DI 2	37
5.3	Example 20: A Complete Programme	40
<b>CHAPTER 6</b>	<b>THE KEYBOARD</b>	
6.1	Description of the Keyboard	45
6.2	Manual Control of the Computer	47
6.3	Recent Additions to the Keyboard	49
<b>Appendix 1</b>	<b>The ELLIOTT Telecode</b>	<b>51</b>
<b>Appendix 2</b>	<b>Powers of 2 in Decimal</b>	<b>52</b>
<b>Appendix 3</b>	<b>The 803 Instruction Code</b>	<b>53</b>
<b>Appendix 4</b>	<b>A. Functions which can cause Overflow</b>	<b>58</b>
	<b>B. The Accuracy of the Division Process</b>	<b>58</b>
<b>Appendix 5</b>	<b>The Automatic Floating-point Unit</b>	<b>60</b>

## CHAPTER 1 INTRODUCTION

### 1.1 General

The 803 is a general purpose electronic computer. It solves problems by obeying previously prepared programmes of simple instructions automatically and at high speed: the term "general purpose" is used to indicate that programmes can be written for a very large variety of applications. Once the programme required to solve one problem has been prepared, it can be used time and again for the solution of similar problems.

The writer of a programme, or programmer, requires a detailed understanding of the problem which is to be solved, together with the ability to express the solution as a sequence of elementary arithmetical steps.

This guide comprises an introduction to programming, with particular reference to the 803, and it is specifically intended for the reader with little or no programming knowledge.

An extensive collection of programmes is available for use with the 803, and full details are given in the publication "The Elliott 803 Library of Programmes". Reference is made to some of these in this guide, and it is shown that use can be made of them to reduce the work of solving new problems.

### 1.2 The Component Parts of the Computer

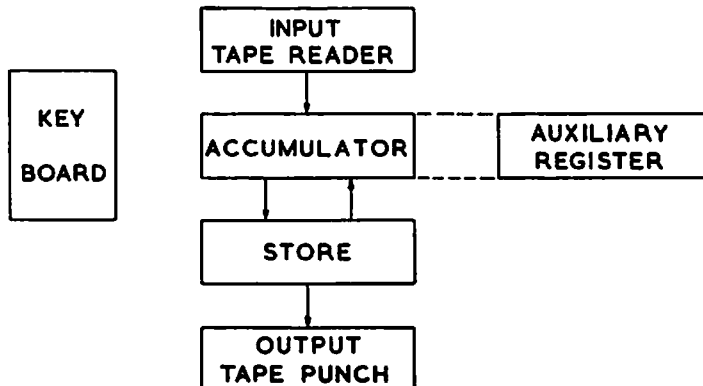
Fig. 1, overleaf, shows the parts of the computer with which the programmer is concerned. There exist also, of course, control and arithmetic units, which respectively cause and enable the computer to carry out its programme. Detailed knowledge of these devices is not essential to a programmer.

#### 1.2.1 The Store

The instructions to be obeyed and the data to be used in a calculation are placed in the store before the calculation commences, and remain there until actually required, when they pass from it to the control and arithmetic units at a suitably high speed. Without such an arrangement, the effective speed of the computer would be limited by the delays incurred in feeding each instruction and number in, and taking out each answer, including intermediate answers, by some manual or mechanical method.

The store of the 803 is divided into many different compartments termed locations; there may be 1024, 2048, 4096 or 8192 of these. Each location can hold one word, that is: one number or two instructions. The locations are numbered from 0 upwards, the number of each location being referred to as its address. The word stored in a location, whether

it be a number or two instructions, is termed that location's content, e.g. the word stored in the location whose address is 456 is "the content of location 456", which is written "C(456)".



**Fig.1 Theoretical Block Diagram of 803 Computer**

In describing the action of the computer, the letter N will frequently be used to represent the address of a location: C(N) therefore indicates the "content of location N". In abridged notation, C(N) is written n.

Locations 0 to 3 are reserved for a special purpose, as will be explained later, but each of the remaining locations may be used for instructions or data, the allocation being at the discretion of the programmer.

### **1.2.2 The Accumulator and the Auxiliary Register**

In most arithmetical operations there are two operands. The usual manner in which 803 performs one step of a calculation is to take one number from its store, and perform some arithmetical operation with it and the number held in a device termed the accumulator. This device, which is closely associated with the arithmetic unit, is capable of holding one word. It is denoted by A, and its content by C(A). In abridged notation, C(A) is written a.

In some operations an extension is required to the accumulator, and this extension is called the auxiliary register.

### **1.2.3 The Input Tape Reader and Output Tape Punch**

The tape reader is used for feeding instructions and data into the computer before it carries out its calculations, and the tape punch is used by the computer to pass out its results. Perforated teleprinter tape, used in these devices, is a most convenient medium; for the input tapes (programmes and data) may be readily prepared, and the output tapes (results) readily interpreted, using standard teleprinter equipment.

Punched cards may also be used as an input medium. A separate publication describes the method of use.

#### 1.2.4 Keyboard

This carries the push buttons, indicator lamps and other devices which enable the operator to control the computer and check that it is operating correctly.

#### 1.3 Form of Instructions

Each instruction is composed of two parts, denoted by F and N, of which:

F specifies the function, i.e. the operation to be performed, and N is a number, which is either:

- (i) the address of a location in the store, or
- (ii) a further specification of the function.

To distinguish between the two uses of the number part of an instruction, we use N for the general case and for use (i), and N for use (ii).

##### 1.3.1 Examples of Instructions

Name of Instruction	Function	Number	<u>Effect</u>
	F	N	
Replace	30	N	The existing C(A) is deleted, and replaced by a copy of C(N).
Subtract	05	N	C(N) is subtracted from the existing C(A), which is deleted and replaced by the difference.

Note: neither of the above actions affects C(N) itself.

Exchange	10	N	C(A) and C(N) are interchanged.
Double C(A)	55	<u>N</u>	C(A) is doubled <u>N</u> times. e.g. if <u>N</u> = 5, the new C(A) will be 32 times the old.

##### 1.3.2 Example 1

Suppose that, at some stage in a calculation, the two numbers x and y are held in locations 5 and 6, and that it is required that  $(8x - y)$  be placed in location 7, for use later on. The instructions which the computer must obey to effect this are:

<u>F</u>	<u>N</u>	<u>Contents after obeying each instruction</u>			
		C(A)	C(5)	C(6)	C(7)
		Initially.... ?	x	y	?
30	5	x	x	y	?
55	3	8x	x	y	?
05	6	(8x-y)	x	y	?
10	7	?	x	y	(8x-y)

As mentioned in 1.2.1, instructions are held in the store, two instructions occupying one location. The instructions in the above example could be held, for instance, in locations 100 and 101. To express this we would write:

Address	Instructions			
	F1	N1	F2	N2
100	30	5	55	3
101	05	6	10	7

The computer is designed to obey the two instructions in one location, and then proceed with those in the next location, and so on, automatically.

### 1.3.3 Exercise 1

Given the same starting point as that in Example 1, write the programme needed to cause the following to be placed in location 7.

(a)  $64x + 8y$  (see below)

(b)  $2x - 4y$

In (a), use will be needed of the following function:

<u>Name</u>	<u>F</u>	<u>N</u>	<u>Effect</u>
Add	04	N	C(N) is added to the existing C(A), which is deleted and replaced by the sum.

## 1.4 Transfer Instructions

### 1.4.1 Conditional Transfers

Situations of the following type arise in many problems:

Two unequal positive numbers, p and q, are held in locations 8 and 9, and these must be rearranged, if necessary, so that the lesser is in location 8, and the greater in location 9.

In order that the computer may determine whether a rearrangement is needed, it is designed to carry out what are termed conditional transfer functions, of which the following is typical:

<u>Name</u>	<u>F</u>	<u>N</u>	<u>Effect</u>
Transfer to first instruction if C(A) is negative.	41	N	<p>If the number in the accumulator is positive or zero, the computer takes no action, and proceeds to the next instruction in normal sequence.</p> <p>But if the number in the accumulator is negative, the computer breaks normal sequence, proceeds to obey the first instruction in location N, and then continues to obey instructions in sequence from that new point.</p>



### Example 2

The scheme to solve the problem posed above is:

Form the difference (p-q). If this is positive, a rearrangement is necessary, so carry it out. But if (p-q) is negative, no rearrangement is required, so the rearrangement must be omitted.

The programme is:

Address	Instructions				
	F1	N1	F2	N2	
150	30	8	05	9	Form (p - q) in accumulator.
151	41	153	30	8	If negative, omit these steps,
152	10	9	10	8	if positive, interchange p and q
153	Next part of programme.				

#### 1.4.2 Unconditional Transfers

In the case instanced above, the computer is required to determine whether "to do something" or "not to do something". Where the problem requires a discrimination between "doing one thing" and "doing another", an unconditional transfer function may be useful as a complement to the conditional transfer. Such a function is:

<u>Name</u>	<u>F</u>	<u>N</u>	<u>Effect</u>
Transfer unconditionally to first instruction	40	N	The computer breaks normal sequence, proceeds to obey the first instruction in location N, and then continues to obey instructions in sequence from that new point.

### Example 3

Two different positive numbers are held in locations 24 and 25, and it is required to make both equal to the greater.

The scheme is:

Compare C(24) and C(25). If C(24) is greater, place a copy in location 25 and proceed to next part of programme. If C(25) is greater, place a copy in location 24, and proceed to next part of programme.

The unconditional transfer is used to effect the underlined step, as the programme shows:

Address	Instructions				
	F1	N1	F2	N2	
250	30	24	05	25	
251	41	253	30	24	If C(24) is greater, place a copy in location 25, and skip the next two instructions
252	10	25	40	254	
253	30	25	10	24	If C(25) is greater, place a copy in location 24.
254	Next part of programme.				

### **Other Uses of Unconditional Transfers**

It is not always convenient to have two parts of a programme in adjacent sections of the store. For instance, in Example 3 the next part of the programme might be in locations 500, 501.....

In such cases, an unconditional transfer instruction would be inserted at the end of one section to cause the computer to transfer to the next.

#### **1.4.3 Exercise 2**

(a) Expand the programme of Example 2 to deal with the case where C(8), C(9) and C(10), all different positive numbers, have to be assembled in ascending order of magnitude, using the following scheme;

Compare C(8) and C(9); if C(8) is greater, rearrange. Repeat for C(8) and C(10). Repeat for C(9) and C(10).

(b) Place your instructions in locations 250 upwards and assume that the next part of the programme is in location 700.

(c) Examine the effect achieved if two of the numbers are equal.

## CHAPTER 2 REPRESENTATION OF DATA AND INSTRUCTIONS

### 2.1 General

All data and instructions are held in the computer in the form of binary numbers, that is to say, they are represented by groups of 1's and 0's. Although ability in binary arithmetic is not essential to a programmer, an understanding of the notation is required, and a brief explanation follows.

Since there exist only the two digits 0 and 1, counting in the binary scale proceeds thus:

One	1
Two	10
Three	11
Four	100
Five	101
Six	110
Seven	111
Eight	1000

The digits represent, from right to left, the powers of 2, i.e. 1, 2, 4, 8 etc., just as in the decimal scale the positions represent the powers of 10, i.e. 1, 10, 100 etc. The binary number 1101 therefore represents.

$$1.2^0 + 0.2^1 + 1.2^2 + 1.2^3 = 1 + 4 + 8 = 13$$

Similarly, in the representation of fractions, the digits to the right of the binary point represent negative powers of 2, i.e.  $1/2$ ,  $1/4$ ,  $1/8$  etc., thus:

<u>Binary</u>	<u>Vulgar</u>	<u>Decimal</u>
.1	$1/2$	.5
.11	$1/2 + 1/4$	.75
.101	$1/2 + 0 + 1/8$	.625

### 2.2 Representation of Numbers within the Computer

In any calculating device, some limit is imposed upon the range of numbers which can be directly represented. For example, on the lower scale of a standard slide rule the numbers run from +1 to +10. Yet it is quite practicable to multiply .003 by -140 on the rule and obtain the result -.42, provided appropriate scale factors are applied to each operand and to the apparent result. Similarly, there is no real bar to using any device for calculating with numbers outside the range which can be directly represented.

In 803 this range is from -1 to just less than +1, achieved as follows:

- (a) In each word there are 39 digits.
- (b) The binary point is assumed to lie between the left-hand digit and that next to it.
- (c) Positive numbers, between 0 (which is regarded as positive) and the upper limit,  $1 - 2^{-38}$ , are represented directly by words beginning with 0, thus:

(i) 0 0000 0000 0000 0000 0000 0000 0000 0000 0000 00

Zero

(ii) 0 1010 1000 0000 0000 0000 0000 0000 0000 0000 00

$$1/2 + 1/8 + 1/32 = 21/32 = .65625$$

(iii) 0 0000 0000 0000 0000 0000 0000 0000 0000 0000 01

$2^{-38}$ , the smallest possible positive number.

(iv) 0 1111 1111 1111 1111 1111 1111 1111 1111 1111 11

$$2^{-1} + 2^{-2} + 2^{-3} + \dots + 2^{-38} = 1 - 2^{-38},$$

the largest possible positive number.

- (d) Negative numbers, between -1 and  $-2^{-38}$ , are represented by words beginning with 1, on the basis that the negative number (-x) is represented by  $(2 - x)$ , thus:

(v) 1 0101 1000 0000 0000 0000 0000 0000 0000 0000 00

$$\text{this is } 1 + 1/4 + 1/16 + 1/32 = 1 \frac{11}{32}$$

which is  $(2 - \frac{21}{32})$ , so the word represents  $-\frac{21}{32}$ . (See footnote)

(vi) 1 0000 0000 0000 0000 0000 0000 0000 0000 0000 00

represents -1, the largest possible negative number.

(vii) 1 1111 1111 1111 1111 1111 1111 1111 1111 1111 11

represents  $-2^{-38}$ , the smallest possible negative number.

---

A quicker way of ascertaining the value of this negative number is to write -1 for the first digit, and then  $+ 1/4 + 1/16 + 1/32$  for the other "ones": adding gives  $-21/32$  directly.

(This technique gives rise to an alternative explanation of the representation of negative numbers, namely that the first digit is -1 and the remainder are + fractions. This is misleading, for the computer treats all digits as positive.)

This method of negative number representation is most satisfactory, for if the computer is made to add the two words representing  $5/8$  and  $-1/2$  it actually adds  $5/8$  to  $1\ 1/2$  and tries, therefore, to produce the number  $2\ 1/8$ , which would be

10 0010 0000 0000 0000 0000 0000 0000 0000 00

But there is no room in the word for the extra digit on the left: this is therefore lost, and the result appears as

0 0010 0000 0000 0000 0000 0000 0000 0000 00

which is the proper representation of  $1/8$ , the correct answer.

Similarly, all additions and subtractions are carried out correctly, provided that the true answer lies between  $-1$  and  $1 - 2^{-38}$ . Multiplication and division are also correctly performed.

### 2.2.1 Sign Digit

The left-hand digit indicates the sign of a number, being 1 for negative numbers and 0 for zero and positive numbers. For this reason it is called the sign digit.

### 2.2.3 Complementing

Given the computer form of any number except  $-1$

e.g. 0 1110 1000 0000 0000 0000 0000 0000 0000 00

or 1 1111 1111 1111 1111 1111 1111 0101 0011 0110 11

to find the computer form of the equivalent negative number:

- (i) Working from right to left, write down 0 for all 0's, if any, found before the first 1 is encountered.
- (ii) Write down 1 for this 1.
- (iii) Thereafter write 1 for 0 and 0 for 1.

Thus the negatives or "complements" of the above examples are

1 0001 1000 0000 0000 0000 0000 0000 0000 00

and 0 0000 0000 0000 0000 0000 0000 1010 1100 1001 01

### 2.3 Scaling

(This section should be treated lightly at first reading, and re-read later when needed.)

Most problems require calculations to be done with numbers which exceed unity in magnitude. It is therefore generally necessary to scale all numbers by a suitable factor, as in the example of the slide rule. We now examine this question and look at number representation from a fresh angle.

Consider the number 53 which, in binary, has the six digits 110101. If we wish to construct a computer word to represent 53, we must add thirty-three zeros to make up a set of thirty-nine digits. These may all be placed to the left of the 110101, or some may be placed to the right and some to the left, thus:

(a) 0 0000 0000 0000 0000 0000 0000 0000 1101 01

(b) 0 0000 0011 0101 0000 0000 0000 0000 0000 0000 00

(c) 0 1101 0100 0000 0000 0000 0000 0000 0000 0000 00

Of these, (b) is an intermediate case, chosen at random, but (a) and (c) represent extremes. For if we try to place the 110101 any further to the right than (a), some digits will be lost, whilst if we place it any further left than (c), say:

(d) 1 1010 1000 0000 0000 0000 0000 0000 0000 0000 00

or (e) 0 1010 0000 0000 0000 0000 0000 0000 0000 0000 00

either the result is "negative", as in (d) or digits are lost, as in (e), or both.

To enable us to describe the varying positions of the digits representing 53, we associate the values

$$2^0, 2^{-1}, 2^{-2}, 2^{-3} \dots 2^{-38}$$

With the 39 digits of the word. We then inspect and determine in which position the units digit of the "53" lies, and we say that the word is, in case (a):  $53 \times 2^{-38}$ , and in (b):  $53 \times 2^{-12}$ , and so on.

But it must be understood that what these expressions mean is that in (a) the word represents 53, so placed that its units digit appears at the right-hand end, while in (b) the word represents 53, so placed that its units digit is 12 places to the right of the sign digit, with 26 places further to the right. For it is as important to know what the word represents, and how, as what it is. To force this point home, although (b) is to us  $53 \times 2^{-12}$ , i.e. 53 represented to scale  $2^{-12}$ , it could equally well be  $106 \times 2^{-13}$ , or  $1696 \times 2^{-17}$  or  $6.625 \times 2^{-9}$ .

The symbol "x" is used in this text as the "represented to scale" symbol in preference to the more frequently used "." to avoid confusion: for "53.2<sup>-12</sup>" could be read as "fifty three point two to the minus twelve".

### 2.3.1 Arithmetic Operations on Scaled Numbers

If the computer is made to carry out an arithmetic operation with the word (b), it will treat it as though it were in fact the number

(b)  $53 \times 2^{-12}$  ( $= 106 \times 2^{-13} = \text{etc.}$ )

And if it is required to add it to

(f)  $7 \times 2^{-12}$  ( $= 14 \times 2^{-13} = \text{etc.}$ )

it will produce the answer

(g)  $60 \times 2^{-12}$  ( $= 120 \times 2^{-13} = \text{etc.}$ )

Similarly, provided the two numbers with which we are dealing are expressed to the same scale, the computer will give the correct answer, in the same scale, in all additive and subtractive operations.

But if (b) and (f) are multiplied, the answer obtained will be

(h)  $371 \times 2^{-24}$  ( $= 1484 \times 2^{-26} = \text{etc.}$ )

To bring this to the same scale as the original operands will require doubling, and the number of doublings will depend on the scale factor in use, for to bring  $371 \times 2^{-24}$  to scale  $2^{-12}$  requires 12 doublings but from  $1484 \times 2^{-26}$  to  $1484 \times 2^{-13}$  requires 13 doublings.

### 2.3.2 Choice of Scale

The choice of scale to use in any part of a calculation depends upon two premises, namely: the maximum magnitude of the numbers to be dealt with, and the accuracy to which the calculation must be performed. This is best shown by example:

(a)  $53 \times 2^{-38}$ : 0 0000 0000 0000 0000 0000 0000 0000 0000 1101 01

(b)  $53 \times 2^{-18}$ : 0 0000 0000 0000 1101 0100 0000 0000 0000 0000 00

Now (a) is such that a change of 1 in the last place corresponds to a change of 1 in the number represented. This scale is therefore suitable when dealing with integers, it is termed "integer scale", and the scale factor  $2^{-38}$  is often omitted in writing when it is obvious that this scale is intended. The maximum positive capacity is ( $2^{38} - 1$ )

$$= 274,877,906,943$$

In (b) the right-hand digit is 20 places to the right of the unit of the "53". A change of 1 here is therefore equivalent to a change of  $2^{-20}$ , or about one millionth, in the number represented. This scale is therefore suitable where such a degree of accuracy is required. The maximum positive number which can be represented is, in "non-computer" binary:

11 1111 1111 1111 1111 . 1111 1111 1111 1111 1111

Which is evidently just less than

$$100\ 0000\ 0000\ 0000\ 0000 = 2^{18} = 262,144$$

Attention is invited to the table of powers of 2 in Appendix 2.

### Fraction Scale

When all numbers to be used are less than unity in magnitude, they may be represented at full scale. Thus

0 1010 0000 0000 0000 0000 0000 0000 0000 00

represents .625 "as a fraction". In this scale, the least change which can be represented is  $2^{-38}$  which is approximately .000 000 000 003 638.

If such a degree of accuracy is inadequate, a higher scale still may be used, but capacity is proportionately reduced.

#### 2.3.3 Exercises

3. What is the smallest step, and what is the maximum capacity, in scale  $2^{-6}$ ?

4. What (power of 2) scale should be used to obtain accuracy to about one thousandth? What is the maximum capacity in this scale?

#### 2.3.4 Other Scales and Methods of Storing Numbers

We have, so far, only discussed scales in which the factor is a power of 2. That such scales may be more convenient than other is suggested by the multiplication problem of 2.3.1, where rescaling is done by doubling, a function provided in the computer. If the scale factor is not a power of 2, rescaling is somewhat more difficult. But other scale factors may be used, sometimes with advantage, where this consideration is of little or no importance.

In rare cases in which it is not possible to find a suitable scale factor for the solution of a particular problem, because the required accuracy and capacity cannot both be obtained simultaneously, recourse has to be had to other methods of number representation, termed "multiple-length" and "floating-point". The 803 library contains programmes which enable these methods to be used. See also Appendix 5.

When it is necessary to store a lot of fairly short numbers in a limited number of locations, recourse may be had to a process termed packing, that is, making one word represent two or more numbers.

#### Example 4

Taking the simplest case, where two positive integers, say 127 (1111111) and 341 (101010101) have to be stored together, we may proceed as follows:

Add  $127 \times 2^{-19}$  to  $341 \times 2^{-38}$  and store the result. Thus:

```
0 0000 0000 0000 1111 1110 0000 0000 0000 00
+ 0 0000 0000 0000 0000 0000 0000 0000 0101 0101 01
= 0 0000 0000 0000 1111 1110 0000 0000 0101 0101 01
```



in which it is seen that the two numbers have remained separate from each other.

When either number is required for arithmetic purposes, a copy of this word is taken from the store, and the unwanted number deleted by a function provided for the purpose.

## 2.4 The Overflow Indicator

Whenever the 803 carries out an arithmetic operation, it automatically tests the numbers being used to determine whether capacity is being exceeded. If this is so, a special device called the overflow indicator is set. The following operations, for example, would cause this; in all cases, the correct answer is outside the proper range, so the computer cannot represent it:

<u>Operation</u>	<u>Correct Result</u>
$1/2 + 1/2$	1
$-1 + -1/2$	-1 1/2
$-3/4 - 3/8$	-1 1/8
Negate (-1)	1
$-1 \times -1$	1
$3/4 \div 3/8$	2
Double 5/16 twice	1 1/4

Once it has been set, the overflow indicator remains set until one of the two "test overflow indicator" instructions (43 and 47) is obeyed. These are conditional transfer instructions, by means of which the computer can be programmed to do one thing if overflow takes place and another if it does not. It is customary to insert one or other of these instructions at intervals in any programme in which there is a risk of capacity being exceeded. Since the overflow indicator, once set, remains set until tested, it is not necessary to test after every operation.

A lamp on the keyboard is illuminated whenever the overflow indicator is set.

### 2.4.1 Deliberate Overflow

In certain cases it is possible to allow capacity to be exceeded, provided that compensating action is taken immediately.

For example, in the sequence:

Add  $1/4$  to  $7/8$ , then subtract  $3/8$ ,

the addition  $1/4 + 7/8$  produces  $1 1/8$  in the accumulator. This is the representation of  $-7/8$ , which is wrong, and the overflow indicator is therefore set. But the immediate subtraction of  $3/8$  from this gives the correct overall result  $3/4$ .

Whenever the correct result of a series of additions and subtractions is within the range  $-1$  to  $1 - 2^{-38}$  inclusive, the 803 will give it correctly, even though the overflow indicator may be set during the series. Deliberate overflow may also be used when employing doubling, halving, multiplication and division functions but greater care is needed when doing so.

## 2.5 The Telecode

Perforated teleprinter tape, or punched paper tape as it is usually termed, is a medium upon which data are recorded by patterns of punched holes. One character consists of between 0 and 5 code holes in a line across the tape, arranged in 5 possible positions. Between the second and third code hole positions there is a small hole, termed the sprocket hole, which is required for mechanical purposes.

To the computer, a hole represents a 1, and "no hole" a 0; each character therefore represents a five-digit binary number between 00000 (zero) and 11111 (thirty one). It is important to appreciate that, although we may quite readily programme the computer to deal with letters, figures and symbols in whatever way we please, the computer recognises them only as groups of five binary digits, or "bits".

To the teleprinter equipment used in preparing input tapes and interpreting output tapes, however, each of the 32 possible combinations of holes has one or two specific meanings as shown in Appendix 1.

To enable a teleprinter to distinguish between the two meanings of a character, two special characters are provided, "letter shift" and "figure shift" (ls and fs). One of these precedes each group of other characters on a tape, to indicate whether they are to be read as letters or as figures and symbols.

The character "line feed" (lf) is interpreted by a teleprinter as an instruction to print the next letters, figures or symbols on a new line, while "carriage return" (cr) causes printing to start or restart at the left hand margin. "Space" (sp) characters are inserted on a tape whenever required, to cause the teleprinter to space out the letters, figures and symbols. "Blank" (bl) has no meaning, and the teleprinter will ignore it; several inches of blanks are punched at each end of a tape to simplify handling.

The choice of characters used to represent figures is governed by the rules:

- (i) The right-hand four digits are the binary form of the figure represented.
- (ii) The left-hand digit is such that the total number of 1's is odd. This digit is termed the parity digit.

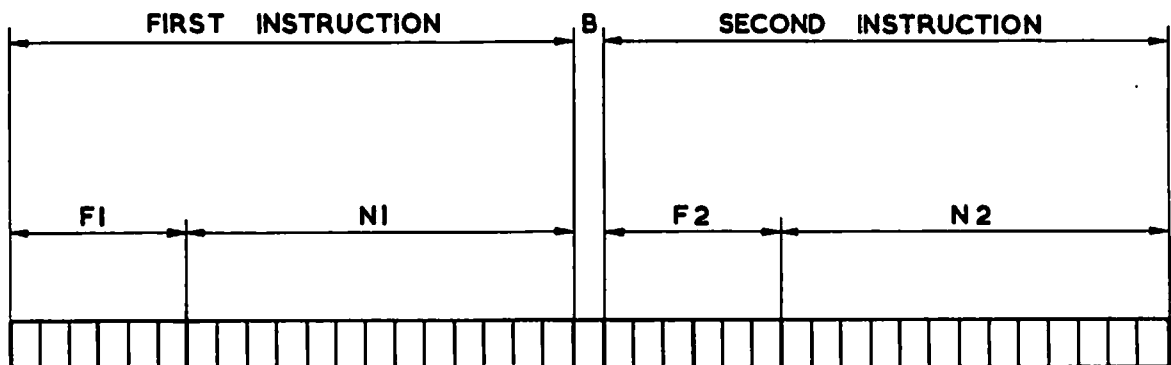
These rules are made so that, respectively:

- (i) Conversion to and from pure binary is relatively simple: all that is needed is to remove or insert the parity digit.
- (ii) The most common mechanical errors in punching, which are
  - (a) Insertion of one extra hole,
  - (b) Omission of one hole,
  - (c) Omission of all holes,
 may easily be detected in all numerical data.

## 2.6 Representation of Instructions

When a word represents a pair of instructions, each instruction is denoted by a group of 19 digits, 6 for the function and 13 for the number. The odd digit, which is the middle digit of the word, is termed the B digit: this has a special purpose which will be described later.

The layout is shown below.



Each function is denoted by six binary digits, which correspond, in two sets of three, to its two-digit octal reference number. Thus function 43 (four-three, not forty-three) is, in binary, 100011. This provides a convenient means of grouping the functions in eight groups of eight; thus Group 0 comprises functions 00 to 07, Group 1 comprises functions 10 to 17 and so on to Group 7, which consists of functions 70 to 77.

The thirteen N digits in each instruction are sufficient to provide a range of N from zero to 111111111111 = 8191 i.e. to enable the address of any location in a store of 8192 locations to be specified. This is the largest size of store which can be fitted in an 803.

Each 803 computer with a store of 4096 locations is adjusted in such a way that, if it is called upon to obey an instruction in which N is larger than the address of the last location in the store, it will ignore the left-hand N digit. Thus the instruction 04 4104 is treated as the instruction 04 8. Where the store has 2048 (or 1024) locations,

the two (or three) left-hand N digits are ignored.

## 2.7 Store Parity Checking

Whenever one word, of 39 digits, is placed in the store, an artificial 40th digit is automatically appended and stored with it: this is termed the parity digit, and is a 1 or a 0 as may be needed to make the total number of 1's in the whole group of 40 digits odd. When a word is taken from the store, the number of 1's in its 40 digits is checked. If this is found to be odd, the parity digit is deleted, and the computer proceeds normally. If it is even, the checking circuit gives rise to a special signal. In the basic 803 the effect of this signal is to stop the computer and illuminate a lamp on the keyboard.

This facility cannot be controlled by programming methods, and the subject is not discussed further.

### 3 THE INSTRUCTION CODE

The 64 possible functions are now described. The full instruction code is given in Appendix 3.

Functions in Groups 0 to 3 are termed Basic Functions, and these, together with the Transfer Functions of Group 4, have in common the fact that the N digits always specify the address of a location, although in some cases (functions 00, 01 and 06), the content of that location does not enter into the operation.

The uses of the N digits in the functions of Groups 5 to 7 are given in the appropriate sections below. The times taken to perform all functions are given in Appendix 3.

#### 3.1 Group 0

Note: None of the functions in this group affects the content of any store location.

Function 00                      Do nothing

The usefulness of this function will be appreciated when the use of the B digit is understood.

Function 01                      Negate

The content of the accumulator is negated.

Function 02                      Replace and count

The existing C(A) is deleted and replaced by a number equivalent to  $C(N) + 2^{-38}$ .

Function 03                      Collate

A new word is formed which has a 1 in each position in which there are 1's in both C(A) and C(N), and 0's elsewhere. C(A) is then deleted and replaced by this new word.

#### Example 5

The collate function may be used when it is desired to extract the wanted number from a location in which two numbers have been packed. Returning to the case of Example 4 in section 2.3.4, if the word containing two numbers is in location 300, and the integer 524287 is stored in location 276, the following instructions suffice to get the right-hand number into the accumulator on its own:

Address	Instructions				
	F1	N1	B	F2	N2
740	30	300		03	276

For the effect is to collate

0 0000 0000 0000 1111 1110 0000 0000 0101 0101 01

with the collating constant

0 0000 0000 0000 0000 0001 1111 1111 1111 1111 11

and thus obtain

0 0000 0000 0000 0000 0000 0000 0000 0101 0101 01

Function 04                      Add

C(N) is added to C(A), which is deleted and replaced by the sum.

Function 05                      Subtract

C(N) is subtracted from C(A), which is deleted and replaced by the difference.

Function 06                      Clear

C(A) is deleted and replaced by zero.

Function 07                      Negate and add

C(A) is subtracted from C(N), C(A) is deleted and replaced by the difference.

The result is the negative of that produced by function 05.

### 3.2 Groups 1 to 3

In Groups 1 to 3 the operations performed are similar to those described above, but whereas in Group 0 the result is placed in the accumulator and the store remains unaffected, the arrangements in Groups 1 to 3 vary. Appendix 3 lists these, and it can be appreciated that the availability of so wide a choice of functions enables very compact programmes to be written for 803.

To "replace" is to delete C(A) and substitute a copy of C(N), to "write" is to delete C(N) and substitute a copy of C(A), and to "exchange" is to do both at once.

### 3.3 Group 4 - Transfer Instructions

Transfer or, as they are sometimes called, "jump" instructions are inserted in a programme wherever it is desired that the computer should obey instructions in a sequence different from that in which they have been stored. Some examples of this have already been given in 1.4, and others are given below.

Function 40                      Transfer, unconditionally, to first instruction

This causes the computer to break normal sequence, carry out the first instruction in location N, and continue then in normal sequence from that new point until a further transfer instruction comes to be obeyed.

### Example 6

When the computer has completed a calculation, it must be stopped. But the speed of the computer is too great for a human operator to stop it at the right instant. An instruction to stop must therefore be written into the programme. There is no specific instruction "stop" in the code, so recourse is had to what is termed a "dynamic stop".

Suppose that the last two instructions of a calculation are in location 843. Then in 844 we may place, as first instruction, "40 844".

It is left to the reader to work out what happens.

There is a loudspeaker on the keyboard, which emits a distinctive high-pitched noise when the computer comes to a dynamic stop.

#### Function 41

Transfer to first instruction if (and only if) C(A) is negative

If the sign digit in the accumulator is a 1 (content negative) when this instruction is obeyed, the action is as for function 40. But if the sign digit is a 0 (content zero or positive), the instruction is interpreted as "do nothing".

### Example 7

Suppose that, in some iterative process, certain steps of a calculation must be performed 6 times. This could be organised as in the programme below in which the entry point is the first instruction in location 10.

Address	Instructions				
	F1	N1	B	P2	N2
9					-5
					This is the way in which the number $-5 \times 2^{-38}$ is expressed on a programme sheet.
10	30	9		20	8
					Place -5 in location 8
11	)				
	)				The instructions to be obeyed 6 times.
15	)				
16	32	8		41	11
					Increase the content of location 8 by 1, and repeat the set of instructions if necessary
17					Next part of programme.

When the instruction 32 8 is obeyed for the first time, the number which comes into the accumulator is -5. On the second occasion it is -4, then -3, -2, -1 and finally after the sixth iteration, 0.

In the above programme "a count is kept in location 8" or "location 8 is used as a count location". The number placed in a count location at the start of such a process is called a "count constant". The action of placing a count constant in a count location is termed "setting a count".

<u>Function 42</u>	Transfer to first instruction if (and only if) C(A) is zero
--------------------	---

If the content of the accumulator is zero when the computer obeys this instruction, the action is as for function 40. But if the content is not zero, the instruction is interpreted as "do nothing".

<u>Function 43</u>	Transfer to first instruction if (and only if) the overflow indicator is set, and clear the overflow indicator.
--------------------	---

The overflow indicator is tested, and if it is found to be in the set condition, the action is as for function 40, coupled with the fact that the overflow indicator is cleared. But if the overflow indicator is found to be in the clear condition, the instruction has the same effect as "do nothing".

Appendix 4 gives a complete list of all the functions which can cause overflow.

If it is desired to use the overflow indicator to test whether some function or group of functions has caused overflow, it is necessary to ensure that the overflow indicator is clear before they take place. This may require the insertion of an instruction whose only effect is "clear overflow indicator (if set)". The instruction 43 (N + 1) placed in the second position of location N has this effect.

#### Functions 44, 45, 46 and 47

These functions are identical in operation to 40, 41, 42 and 43, except that where a transfer takes place it is to the second instruction in the specified location instead of to the first.

Any function of Group 4, except 43 and 47, may be used to produce a dynamic stop. In the case of functions 41, 42, 45 and 46 the stop is conditional.

### 3.3.1 Exercises

5. Write the two sets of instructions which cause the computer to compare C(416) and C(417) and then clear location 418 if and only if they



are

(a) Unequal

(b) Equal

6. (a) Why can neither function 43 nor 47 be used to produce a dynamic stop?

(b) Write the instructions which cause the computer to add C(735) and C(736) and then stop if the sum is in excess of capacity, but go on to the next part of the programme otherwise, with the sum in the accumulator.

7. C(24) and C(25) are two positive numbers. Write a programme which places a number equal to the greater in location 26, using any functions you require from Groups 0 to 4. (Target: 5 instructions).

### 3.4 Group 5 - Multiplication, Division and Shift Functions

When two fractions of equal length (e.g. +.23 and -.47) are multiplied, the answer is, in general, twice as long as either (-.1081). Similarly, if two computer words each of 38 fractional digits and one sign digit are multiplied, the full product has 76 fractional digits and one sign digit. Provision is made in 803 to "extend" the accumulator in this and certain other cases to accommodate one such double-length number. This is held in such a way that the sign digit and the next 38 digits are in the accumulator itself, while the remaining 38 digits fill the extension, which is termed the Auxiliary Register (A.R.)

Group 5 functions are the only ones which effect the A.R.; none of them affects the store.

In the case of functions 52, 53 and 56, (multiply and divide), the N digits specify the address of a location in the store. In function 57 (read A.R.), the N digits are not used. In functions 50, 51, 54 and 55 (shifts), the seven right-hand N digits specify a number N: this N has therefore a range of from 0 to 127 inclusive, and if N = 145 say, then N = 17.

#### Function 50

Halve the double-length number N times

The digits of the double-length number are shifted N places to the right, the N right-hand digits being lost. If the original number is positive or zero, N zeros are inserted at the left-hand end: if it is negative N ones are inserted.

Thus the sign of the number is maintained, and the effect is a true division by  $2^N$ , subject to any error occurring through loss of right-hand digits.

### Example 8

If the double-length numbers  $1/2$  and  $-1/2$

i.e.:     0 1000 0000.....0000 0000

and        1 1000 0000.....0000 0000

and halved 4 times, the results are

0 0000 1000.....0000 0000

and        1 1111 1000.....0000 0000

which are  $1/32$  and  $-1/32$  respectively.

### Exercise 8

Verify that if the double-length number  $10 \times 2^{-76}$  be halved twice, the result is  $2 \times 2^{-76}$ .

Function 51                   Shift C(A) right N places.   Clear A.R.

The digits in the accumulator are shifted N places to the right, the N right-hand digits being lost, and N zeros are inserted at the left. The A.R. has no part in this process: it is cleared as a separate operation.

### Exercise 9

Verify the following:

(a) If the number in the accumulator is  $-3/4$ , and the function 51 1 be performed, the result is  $+5/8$ .

(b) If the double-length number is  $27 \times 2^{-39}$ , and the function 51 2 be performed, the resulting double-length number is  $3 \times 2^{-38}$ . ( $27 = 11011$ ).

Function 52                   Multiply with double-length product

The existing C(A) is multiplied by C(N). C(A) and the existing content of the A.R. (which is not used in the function) are deleted, and replaced by the double-length product.

Function 53                   Multiply with single-length rounded product.  
Clear A.R.

The existing C(A) is multiplied by C(N). C(A) is deleted and replaced by the left-hand 39 digits of the product, rounded off to be as nearly correct as possible. The A.R., whose original content is not used in the function, is cleared.

### Example 9

If  $9 \times 2^{-20}$  and  $3 \times 2^{-20}$  are multiplied by function 52, the result is  $27 \times 2^{-40}$ , held as a double-length number. 27 in binary is 11011, so the result appears thus:

Accumulator	A.R.
0.....0001 10	11 0000.....0000

But if function 53 be used, the process is, in detail as follows;

- (a) the double-length product is formed.
- (b) the quantity  $2^{-39}$  is artificially added.
- (c) the left-hand 39 digits of the result are placed in the accumulator, the other 38 digits are lost, and the A.R. is cleared.

The arithmetic may be represented thus:

	Accumulator	A.R.
(a)	0.....0001 10	11 0000.....0000
+		1
(b)	0.....0001 11	01 0000.....0000
(c)	0.....0001 11	00 0000.....0000

which is  $7 \times 2^{-38}$ , the correct approximation.

Function 54                      Double the double-length number N times

In this function, the digits of the double-length number are moved N places to the left, N zeros are inserted at the right-hand end, and the N left-hand digits are lost. Unless overflow takes place the effect is multiplication by  $2^N$ .

If  $N \geq 38$ , the A.R. is clear after this function.

Function 55                      Double C(A) N times. Clear A.R.

The digits in the accumulator are moved N places to the left, N zeros are inserted at the right-hand end, and the N left-hand digits are lost. Unless overflow takes place, the effect is multiplication by  $2^N$ .

The A.R. has no part in this process: it is cleared as a separate action.

Function 56                      Divide. Clear A.R.

Division is the inverse of multiplication, in that as  $a \times b = ab$ , so  $ab/b = a$ . Just as it is useful to have a double-length "ab" (product) in multiplication, so it is useful to have a double-length "ab" (numerator, or dividend) in division.

Function 56 therefore consists of the division of the double-length number by C(N). C(A) is deleted, and replaced by the single-length, unrounded quotient. The A.R. is cleared by the process.

When the numerator can be adequately expressed by a single-length number in the accumulator, care must be taken to ensure that the A.R. is clear before division takes place.

A note regarding the accuracy of the process used is given in Appendix 4B, and the arithmetic is discussed in Appendix 5.

#### Example 10

(a)  $C(151) = a \times 2^{-38}$ ,  $C(152) = b \times 2^{-38}$

To place  $a/b \times 2^{-38}$  in location 153

Address	Instructions				
	F1	N1	B	F2	N2
387	30	151		50	38
388	56	152		20	153

Form  $a \times 2^{-76}$   
Divide, forming  $a/b \times 2^{-38}$ , and write.

(b)  $C(251) = a \times 2^{-19}$ , known to be positive

$C(252) = b \times 2^{-19}$ , To place  $a/b \times 2^{-19}$  in location 253

487	30	251		51	19
488	56	252		20	253

Form  $a \times 2^{-38}$  and clear A.R.  
Divide, and write result.

(c)  $C(351) = a \times 2^{-19}$ , sign unknown,  $C(352) = b \times 2^{-19}$ .

State of A.R. not known. To place  $a/b \times 2^{-19}$  in location 353.

587	51	0		30	351
588	50	19		56	352
589	20	353			

Clear A.R., replace  $a \times 2^{-19}$ .  
Divide  $a \times 2^{-38}$  by  $b \times 2^{-19}$ .  
Write result.

The instructions 51 0 and 55 0 have the effect "Clear the A.R.", and no other.

#### Function 57

Read A.R.

The existing C(A) is deleted, and replaced by a word comprising a zero in the sign digit position and a copy of the 38 digits of the A.R. content in the remaining positions.

The A.R. is not affected.

This provides, in effect, a fast double-length left shift of 38 places, subject to the following provisos:

- (i) That the sign digit will be wrong if the number is negative.
- (ii) That the A.R. is not cleared.

The second of these points is usually of little consequence: in fact, it may well be found to be an advantage. How the former is overcome is shown below.

### Exercise 10

Verify that the following process is satisfactory, and that it is faster than would be the case if the instruction 54 38 were used:

a and b are integers of unknown sign, in locations 643 and 644. It is required to place the integer ab in location 645, it being known that ab is of smaller magnitude than  $2^{37}$ .

Address	Instructions					
	F1	N1	B	F2	N2	
412	30	643		52	644	ab x 2 <sup>-76</sup>
413	57	0		54	1	Double signless ab x 2 <sup>-38</sup>
414	50	1		16	645	Store signed ab x 2 <sup>-38</sup>

### Example 11

This emphasises the advantage of multiplying double-length and dividing, successively, whenever an expression of the form

$$\frac{a.c.e.....}{b.d.....}$$

is to be evaluated.

Suppose that a, b and c are stored to scale p (where p may be of the form  $2^{-r}$  or not), in locations 141 to 143. To evaluate ac/b and place it to scale p in location 144, we need:

Address	Instructions				
	F1	N1	B	F2	N2
614	30	141		52	143
615	56	142		20	144

a x p : ac x p<sup>2</sup>

(ac/b) x p :

This gives a clue to the method used to rescale after multiplication or before division when using scales not of the form  $x 2^{-r}$ . For if we make b = 1 the result is ac x p ; if c = 1, the result is (a/b) x p.

### Example 12

To extract the positive square root of a fraction, x.

If the formula  $y_{n+1} = \frac{1}{2}(y_n + \frac{x}{y_n})$

be used to generate a series of numbers  $y_i$ , it can be shown that successive values of  $y_i$  converge monotonically to the square root of  $x$  whatever initial value  $y_1$  be taken.

In adapting this process to the 803,  $y_1 = 1 - 2^{-38}$  is chosen, as this is the largest positive number which can be held, and must therefore be greater than or equal to the required root. Thus, until the best possible approximation to the root is found each new  $y_i$  is less than the previous, and  $y_{n+1} - y_n$  is negative.

Ideally,  $y_{n+1} - y_n$  should eventually reach zero, but rounding errors may bring about a situation in which it oscillates between  $\pm 2^{-38}$ . The process is therefore repeated until some  $y_{n+1} - y_n$  is zero or positive, at which stage the programme "exits with  $y_n$  in the accumulator".

#### Programme

It is assumed that  $x$  is in the accumulator, and that the A.R. is clear. The entry point is the first instruction in location 501.

Address	Instructions					
	F1	N1	B	F2	N2	
501	41	501		42	507	If negative stop. If zero exit.
502	20	509		30	508	Write x.
503	20	510		30	509	Write $y_1$ . Replace x.
504	56	510		04	510	Form $2y_{n+1} = (y_n + \frac{x}{y_n})$
505	51	1		15	510	Write $y_{n+1}$ , form $y_{n+1} - y_n$
506	45	503		07	510	If negative, repeat: if positive, exit with $y_n$ .
507	40	511		00	0	EXIT
508		+274 877 906 943				$1 - 2^{-38}$
509						x
510						$y_n$
511						Next part of programme.

The reader is advised to work through this example to find the square root of  $1 - 2^{-38}$ . Attention is invited to 2.4.1 and to Appendix 4B.

#### 3.5 Group 6

Group 6 instructions are not allocated in the basic 803, the

group having been reserved for special use in installations having additional equipment. In the basic machine they are interpreted as "do nothing".

See Appendix 5 for the details of functions available when an Automatic Floating-Point Unit is fitted to the 803.

### 3.6 Group 7

Group 7 functions are mainly concerned with input and output, and are considered below individually.

#### Function 70                      Read the Word Generator

On the keyboard there is a set of buttons, called the word generator, on which a computer word can be set manually.

In function 70, the existing C(A) is deleted, and replaced by a copy of the word set on the word generator. The N digits in the instruction are not used.

This process can be used, in conjunction with other buttons on the keyboard, to place a few words in the computer by hand whenever the need arises.

It is also possible, by writing function 70 instructions in a programme, to enable the computer operator to control the way in which the computer obeys that programme.

#### Example 13

In some particular application of a computer, the calculation consists of two separate parts, A and B, of which the second part may take two different forms, B<sub>1</sub> and B<sub>2</sub>. Whether B<sub>1</sub> or B<sub>2</sub> is to be used is at the discretion of the operator.

The instructions comprising part A end in location 200. B<sub>1</sub> commences in location 301, and B<sub>2</sub> in 401. If the operator depresses the top left hand (sign digit) button of the word generator, B<sub>1</sub> is used. If he depresses the next button to it, B<sub>2</sub> is used. If he depresses neither, or both, the computer waits until one and one only is depressed.

#### Programme

Address	Instructions				
	F1	N1	B	F2	N2
200	Last instructions of A.				
201	70	0		45	203    Read: To 203 (rh) if 1.....
202	55	1		41	401    Double: To B <sub>2</sub> if 01.....
203	40	201		55	1    Return if 00.....: Double
204	41	201		40	301    Return if 11.....: To B <sub>1</sub> if 10

### Function 71

### Read Input Channel 1 (normally Tape Reader)

In this function one character is read from the input tape, and the five-digit binary number obtained therefrom is "mixed" into the five right-hand positions of the accumulator. The instruction to read Input Channel 1 is usually preceded by a 06, 16 or 55 5 instruction, so that (at least) the five right-hand positions in the accumulator are clear when function 71 is performed. If such is the case, the effect of the function is to insert the character into the right-hand end of the accumulator.

If however, any or all of the five right-hand digits of the accumulator are not zeros when function 71 is performed, the effect obtained in each digit position is defined by:

0+0 = 0;    0+1 = 1;    1+0 = 1;    1+1 = 1, without carry

After the character has been read, the tape reader drive mechanism operates, and the tape is moved forward to bring the next character into the reading position. If the next function 71 instruction is so placed in the programme that the computer reaches it before the tape has been moved far enough, a circuit known as the "tape reader busy line" automatically holds the computer up until the tape reader is ready. However, the speed of the Elliott tape reader is such that this is not likely to happen under normal circumstances.

The time taken for the tape to be moved depends upon the type of tape reader fitted.

### Example 14

A two-digit decimal number has been punched on the input tape, and the tape has been placed in the reader in such a way that the first character is in the reading position.

The following programme reads this number, and stores it as an integer in location 98. The entry point is the first instruction in location 100:

Address	Instructions					
	F1	N1	B	F2	N2	
99					+15	Collating constant.
100	06	0		71	0	Read first character.
101	03	99		20	98	Delete parity digit, if any.
102	24	98		55	3	Multiply by 10, and store in 98.
103	24	98		06	0	
104	71	0		03	99	Read second character, delete parity digit, if any.
105	24	98				Add to number in 98.



### **Exercise 11**

How long would an 803 completed before December 1960 (See Appendix 3) take to obey the above if the tape reader's maximum speed is one character every

(a) 9.36 milliseconds

(b) 2.16 milliseconds?

(Note: These times are quoted for exercise only, and are not typical of the equipment provided)

### **Function 72**

This is described, with functions 75, 76 and 77, below.

### **Function 73**                      Write the address of this instruction

The N digits in the instruction specify the address of a location. The action is:

C(N) is deleted, and replaced by a word in which the right-hand nineteen digits correspond to the instruction "00 N", where N is the address of the location containing the 73 instruction. The state of the left-hand part of the new C(N) varies.

This instruction is generally used in conjunction with the B digit when utilising subroutines, and detailed discussion is therefore postponed until these topics are reached.

### **Function 74**                      Punch Specified Character on Output Channel 1

In this function, the character with telecode value N, where N in this case corresponds to the five right-hand N digits of the instruction, is punched on the output tape.

The other N digits are ignored: no numbers in the computer are affected.

When the necessary signals have been sent to the punch, the computer is then free to proceed with other activity. But if the next function 74 instruction is so placed in the programme that the computer is ready to obey it before the punch has finished the punching process, a circuit known as the "punch busy line" automatically holds the computer up until the punch is ready.

### **Example 15**

The instruction 74 19 causes the punching of the character which a teleprinter interprets as S or 3 according to the shift in use.

The instruction 74 115 has the same effect.

Functions 72, 75, 76 and 77

Channel 2 functions

The 803 has been designed in such a way that a variety of input and output devices or a magnetic film backing store may be attached to what is termed Channel 2.

The interpretation of functions 72, 75, 76 and 77 is described in the separate publications relating to the applications of such devices.

In computers in which use is not made of Channel 2, function 72 is interpreted as "clear the accumulator" and functions 75, 76 and 77 as "do nothing".

## CHAPTER 4 B-LINE MODIFICATION AND SUBROUTINES

Examples have been given, in the earlier chapters of this guide, in which a programme has been so written that the computer obeys one set of instructions several times. This technique saves storage space and is of value whenever one small part of a calculation needs to be carried out more than once.

This chapter describes methods whereby this can still be done when

- either (i) there must be some slight change in the instructions each time the set is used.
- (ii) the set is to be used on odd occasions during the course of a long calculation, rather than several times together.

### 4.1 B-Line Modification

Instances frequently occur in which the computer is required to carry out, several times, an action which is much the same at each repetition, but changes in some small detail each time it is performed. An elementary example is the addition of, say, 100 numbers stored in locations 100 to 199; after clearing the accumulator the functions to be performed are "add C(100)", "add C(101)", "add C(102)"..... "add C(199)". The action here is always "add", the change is an increase in the address of the addend.

Since instructions are stored in the computer in a numerical code, it is possible to cause a number to be added to an instruction, and thereby change or modify it, before it is obeyed. This could readily be done by using such a function as 24, specifying the location holding the instruction to be modified. Better still, in the addition problem considered above, would be function 22; for the addition of  $2^{-38}$  to the content of a location is tantamount to an increase by 1 of the N in the second instruction therein.

But an automatic facility termed B-line modification which is an improvement on either of the above is provided, and is as follows:

(i) If the B digit between a pair of instructions is a 0, then no modification takes place, and each instruction is obeyed as stored.

(ii) But if the B digit is a 1, then after the first instruction (F1 N1) has been obeyed normally, the second instruction (F2 N2) is modified by the addition of the right-hand part of the (new) content of location N1 before being obeyed.

The modification takes place in the control section, takes no extra time, and does not affect the version of the instruction (F2 N2) held in the store.

When a location (N1) is being used to hold a modifier, it is referred to as a B-line. This term is a legacy from one of the earlier

Manchester University computers, in which numbers were represented by lines of dots on a cathode ray tube. The A-line corresponded to the accumulator, and the B-line held the modification, if any, to be made to the next instruction.

There are restrictions on the combined use of B-line modification and transfer instructions, and function 77; these are stated in Appendix 3.

It is conventional to indicate B = 1 by placing a / in the B column rather than by writing a 1.

It does not matter what use is made of the N1 digits in the first instruction.

#### Example 16

If it is desired to have an instruction (F2 N2) modified by the content of location 173, we may write any of the following:

F1	N1	B	F2	N2
00	173	/	F2	N2
22	173	/	F2	N2
50	173	/	F2	N2
57	173	/	F2	N2
74	173	/	F2	N2

In the second instance, where C(173) is altered by function 22, the modification is by the new content.

#### Example 17

A situation is frequently reached, near the end of a calculation, in which the 5 right-hand digits of the accumulator content correspond to a telecode character which must be punched on the output tape. The following instruction pair causes the computer to do this:

20	4	/	74	0
----	---	---	----	---

The relevant digits are placed in location 4, and the instruction 74 0 is then modified into the required output instruction before it is obeyed.

But observe that there must be no 1's in the accumulator in the F2 positions: for these would also be placed in location 4, and, when added to the existing F2, would produce an instruction specifying some function other than 74.

#### Exercise 12

Verify that, in Example 17, there may be 1's elsewhere in the accumulator (e.g. in the F1 positions, or in the 8 left-hand N2 positions), without spoiling the effect.

### Exercise 13

Satisfy yourself by theoretical consideration, or by writing the whole thing out in binary, that

If  $C(A)$  is  $-80 \times 2^{-38}$  and  $C(238)$  is S, before the computer obeys

10      238      /      04      200

then afterwards  $C(A) = S + C(120)$

and                       $C(238) = -80 \times 2^{-38}$

This is employed below.

### Example 18

The following programme adds together 100 items, namely the contents of locations 100 to 199 inclusive, and exits with the sum in location 238. It is entered at the first instruction of location 240. (It is assumed that the sum is within capacity).

Address	Instructions				
	F1	N1	B	F2	N2
238					Count location and total store.
239				-100	Count constant.
240	26	238		30	239
241	10	238	/	04	200
242	12	238		41	241
243	Next part of programme.				

Note how the powerful "exchange" functions 10 and 12 are used. They enable the total-to-date and the count to be switched between location 238 and the accumulator. Without such functions it would be necessary to allocate separate locations for the count and the total, and there would be more instructions in the loop.

### 4.2 Subroutines

Suppose that, during a lengthy calculation, it is necessary to find the square roots of several numbers. The need would then arise for the computer to carry out the functions of Example 12 several times, and a considerable saving in storage requirements could be achieved by arranging that the same set of instructions is used each time, rather than by having several similar sets, each to be used once.

This technique can be applied to any small section of programme which is designed to fulfil a standard requirement, and such sections of programme are termed subroutines. Several are available in the Elliott 803 Library of Programmes: these include, in addition to subroutines

which form algebraic and trigonometrical functions of numbers, others which read numbers from the input tape or punch numbers on the output tape, carrying out the necessary decimal-to-binary or binary-to-decimal conversions.

As far as is practicable, each subroutine is written to conform to the following standard:

- (i) The first location, location p say, is spare.
- (ii) The entry point is location p + 1.
- (iii) The exit location contains the instruction pair.  

$$00 \quad p \quad / \quad 40 \quad 1$$
- (iv) If the purpose of the subroutine is to calculate the function of some number, that number must be in the accumulator at the time at which the subroutine is brought in to use, and the function is in the accumulator when the subroutine exits.

The main programme and its subroutine(s) are placed in different parts of the store. Wherever it is required to employ the subroutine, an instruction pair of the following form is written in the main programme:

Address	Instructions				
	F1	N1	B	F2	N2
z	73	p		40	p+1
z+1	Next part of main programme.				

This causes the link, the "instruction" 00 z, to be placed in the spare location in the subroutine, and transfers to its entry point. When the subroutine has been obeyed, the exit instructions

$$00 \quad p \quad / \quad 40 \quad 1$$

are obeyed as "transfer to location z+1", i.e. to the next part of the main programme.

To appreciate the value of this in programming, consider again the case in which several square roots must be found during a calculation. Provided steps are taken to see that the necessary subroutine is in locations p upwards, the instruction pair

$$73 \quad p \quad \underline{40 \quad p+1}$$

may be written as many times as is necessary, and be regarded as the equivalent of one instruction: "form the square root of C(A)". It is customary to underline each instruction which specifies a transfer to a subroutine.

This technique is employed even when a subroutine is used only once during a calculation. For, by doing so, the need to have different

versions of each subroutine is avoided, and so is the clerical effort of copying the instructions into the middle of a new programme.

Detailed examples are given in the next chapter.

## CHAPTER 5 INPUT ROUTINES

This chapter discusses the methods by which programmes are placed in the store of the computer, and ends with an example of a complete programme.

Although it is possible, by manipulating the buttons on the keyboard, to place a few words in the store of the computer by hand, the process is slow and liable to error. The only practical means by which a new programme can be stored is to make the computer obey a set of instructions which cause it to read characters from the input tape, build the 5-digit groups obtained therefrom into words, and place these words in its own store. Such a set of instructions is permanently "built-in" in locations 0 to 3: it is referred to as "The Initial Instructions", and can be quickly brought into use whenever it is desired to re-programme the computer.

This set satisfies the main requirement of a fixed programme, namely that it should not take up much storage space. But its process of word-assembly is necessarily very rudimentary, and to prepare the type of programme tapes which can be read by it is not easy. For this reason, a second input routine, the Translation Input Routine (T.I.) has been written, by means of which the computer can be caused to read and translate programme tapes of a type which are easy to prepare.

### 5.1 The Initial Instructions

#### 5.1.1 Method of Storage

The Initial Instructions are permanently held in locations 0 to 3 and cannot be overwritten. If, for example, the instruction 20 2 is included in a programme, it will be interpreted as "do nothing".

Locations 0 to 3 differ from other locations in one other respect: namely that if any attempt is made to use them as sources of numbers, they appear to be sources of zero. Thus the instruction 30 3 is "clear the accumulator" whilst 02 0 is "replace C(A) by  $2^{-38}$ ." This last is of practical value at times.

The Initial Instructions use location 4 as a count location: programmes may therefore be placed in locations from 5 upwards.

#### 5.2.1 Binary-coded Programme Tapes

The Initial Instructions are designed to read what is termed a binary-coded programme tape. On such a tape each word is represented by a group of eight characters. Thus 24 18 / 31 21 is represented by the eight characters

10101 00000 00000 10010 10110 01000 00000 10101,

in which the first digit is a 1 and the remainder correspond to the binary form of the word itself.



The words of the programme are preceded by the eight-character group corresponding to the word

00      0      00      (N - 4)

where N is the address of the location in which the first word of the programme is to go.

The way in which the Initial Instructions read such a tape is described in the Elliott 803 Library of Programmes.

## 5.2 The Translation Input Routine

Programmes are usually written in the style shown in this guide, that is to say, in an octal/decimal code, and each programme is usually composed of a number of blocks. The T.I. has been designed to read programme tapes on which the instructions are punched in a form which is directly derived from the octal/decimal written form, the figures being interspersed with control symbols to indicate which number is what. The T.I. can also assemble one programme from several separate blocks of instructions: furthermore, it can decipher programmes which have been written in the relatively-addressed style described below.

The T.I. is not held permanently in the store. Whenever it is desired to re-programme the computer from a tape punched in the "T.I. Code", it is first necessary to place the T.I. in the store by means of the Initial Instructions. For this purpose, a binary-coded programme tape of the Translation Input Routine is kept by each 803 computer. The process may sound rather complex, but in practice it takes only a few seconds.

### 5.2.1 Relative Addresses

Until the blocks of a programme have been written, their lengths are not known, and it is therefore not practicable to allocate storage space to them. But it is not possible to complete the writing of each block unless there is some means of specifying the address of any word in that or any other block: one cannot complete the instruction 40 N until one knows what N is. This apparent deadlock is resolved by a technique known as relative addressing

Before any instructions are written, a list is made of the blocks which are needed. These blocks are numbered from 1 upwards. For example:

1. Main Programme.
2. Constants and working space used by 1.
3. Input subroutine. (To read in data).
4. Output subroutine. (To punch results).

There is no limit to the number of subroutines used, nor need the main programme be all in one block: complex calculations are usually programmed in several blocks.

Each block which does not comprise a subroutine chosen from the Elliott 803 Library of Programmes is then written separately, each address being written as though the block started in location "0,". A comma is placed after each such relative address, thus:

20 6, represents "Write in location 6, of this block".

Cross-references to locations in other blocks are written thus:

22 7,2 represents "Count in location 7, of block 2".

Absolute addresses and numerical details (N's) are indicated in the normal way, without commas, thus:

04 128 represents "Add C(128) to C(A)".

74 29 represents "Punch cr".

#### Example 19

The following is the square root programme of Example 12, re-written as a relatively-addressed subroutine, with standard entry arrangements.

Address	Instructions				
	F1	A1	B	F2	A2
0					Link.
1	41	1,		42	7, Enter with x in accumulator and A.R. clear.
2	20	9,		30	8,
3	20	10,		30	9,
4	56	10,		04	10,
5	51	1		15	10,
6	45	3,		07	10,
7	00	0,	/	40	1 Exit.
8		+274 877 906 943			1 - 2 <sup>-38</sup>
9					x
10					y <sub>n</sub>

#### 5.2.2 The Directory

When the programme has been completed, storage space is allocated to the blocks, and a directory is written at the head of the whole programme. This consists of a list of the block addresses, the absolute addresses of locations 0, of each block. A copy of this goes on the pro-

gramme tape, and indicates to the T.I. where it is to place the various blocks, and enables it to decipher the relative addresses.

The T.I. occupies and uses locations 5 to  $(176 + B)$  where B is the number of blocks in the programme which it is reading. The Elliott 803 Library of Programmes contains a number of checking routines by means of which any mistakes in a new programme can be more easily diagnosed. These are customarily placed in locations at the "top" end of the store. It is therefore best to place programmes between locations 192 and 3800 unless the need arises to use more space than this.

### 5.2.3 Numerical Data

The T.I. can read constants as well as instructions: these should be written on a programme sheet as shown below.

#### (i) Integers

+ 453	represents	$453 \times 2^{-38}$
- 9	represents	$-9 \times 2^{-38}$

#### (ii) Fractions

+ .5	represents	$1/2$
- .625	represents	$-5/8$

There must not be more than eleven decimal digits

#### (iii) Pseudo-Instructions

Best shown by example.

Since      00      3      /      00      0

is stored as      000000000000000011100000000000000000

this pseudo-instruction pair is a convenient way of writing  $7 \times 2^{-19}$ .

#### (iv) Constants to a preset scale

When many constants, all to some scale which is not provided for above, are needed in a programme, it is possible to "adjust" the T.I. in such a way that it divides any constant which contains a decimal point by a preset scaling factor before storing it. Thus to use scale  $\times 10^{-5}$  the scaling factor is preset to 100,000 (one hundred thousand). When this facility is in use, facility (ii) above is lost.

Such constants should be preceded by + or - and must contain a decimal point, thus:

-500.  
+17.23  
-.006

Further details of this, and of all other facilities offered by the T.I. are given in the Elliott 803 Library of Programmes.

### 5.3 Example 20 A Complete Programme

**Problem** To calculate the arithmetic mean  $\bar{X}$  and standard deviation SD for each of several sets of numbers  $X_i$ . Each set has a reference number M and contains N items. Each item is a positive decimal fraction:  $\sum X_i < 1$  for any set.

#### Formulae

$$\bar{X} = \frac{1}{N} \sum X_i ; SD = \left\{ \frac{1}{N} \sum X_i^2 - \bar{X}^2 \right\}^{\frac{1}{2}}$$

<u>Formats</u>	<u>Output</u>	The results to be printed in four columns without headings. The columns to represent:
		M      N      Mean      SD
	<u>Input</u>	The data tape to be punched with each set represented by the integer M, then the integer N, and then each of fractions $X_i$ of the set.

**Note:** These formats must be decided before programming can begin in earnest. The problem itself usually suggests a suitable output format: unheaded columns are being accepted here merely to simplify the example. The general arrangement of the input format can be determined by answering the question 'What numbers would I need, and in which sequence, if I had to do the calculation myself?' The detailed arrangement for each input number must be in accordance with the specification(s) of the input subroutine(s) to be used: it is assumed here that this is so.

Output is described in terms of a printed page as the computer's output tape is always interpreted by a teleprinter.

<u>Blocks</u>	1	Main programme.
	2	Working space for 1.
	3	Input subroutine.
	4	Print subroutine.
	5	Square root subroutine.

#### Subroutines

<u>For Block 3:</u>	General Number Input Subroutine, 803 T 5
Entry:	Standard.
Store:	55 Locations.
Function:	Reads one integer or fraction.
A.R.:	Clear on Exit

For Block 4    General Number Print Subroutine    803 T 6

Entry:            Standard entry with a parameter word in the location after that which holds the entry instructions 73... 40... . Exit from the subroutine is to the location after that holding the parameter word.

Parameters:	<u>To Print</u>	<u>On New Line</u>	<u>On Same Line</u>
	Fraction	40 0 / 00 0	40 0    30 0
	Integer	20 0 / 00 0	20 0    30 0

Store:            131 locations.

Function:        Prints C(A), regarded as a fraction or as an integer, on the same line as the last number printed or on a new line, according to the parameter used.

A.R.:            Clear on exit.

For Block 5    Square Root Subroutine shown as Example 19 of this guide.

Entry:            Standard.

Store:            11 locations.

Function:        Forms square root of C(A).

A.R.:            Must be clear on entry:    clear on exit.

Main Programme and Block 2

When the above details have been attended to, the main programme can be written. This is quite straightforward, and is given over-leaf. Locations in Block 2 are allocated as the need arises, care being taken to see that no new location is taken up unless all those so far allocated are still in use. Use is made of the exchange functions whenever possible.

Block 2

- 0,        Holds  $\sum X_1$ .
- 1,        Holds  $\sum X_1^2$ .
- 2,        Holds  $N \times 2^{-38}$  and later  $\frac{1}{N}$
- 3,        Count location, initially holds  $-N \times 2^{-38}$ .  
          Later used to hold  $\bar{X}$  and then  $\bar{X}^2$ .
- 4,        Holds  $X_1$ .

# Block 1

Address	Instructions					Notes
	F1	N1	B	F2	N2	
0	73	0,3		<u>40</u>	<u>1,3</u>	Read M.
1	73	0,4		<u>40</u>	<u>1,4</u>	Print M
2	20	0	/	00	0	Parameter for integer on new line
3	73	0,3		<u>40</u>	<u>1,3</u>	Read N.
4	26	0,2		26	1,2	Clear $\sum X_i$ and $\sum X_i^2$ .
5	20	2,2		21	3,2	N to 2,2: -N to 3,2.
6	73	0,3		<u>40</u>	<u>1,3</u>	Read $X_i$ and clear A.R.
7	24	0,2		20	4,2	Add to $\sum X_i$ .
8	53	4,2		24	1,2	$X_i^2$ added to $\sum X_i^2$ .
9	02	3,2		42	11,	Count and test for end of set.
10	20	3,2		40	6,	Set not ended, read next item.
11	02	0		56	2,2	$2^{-38} \div N \times 2^{-38} = \frac{1}{N}$ , as fraction.
12	10	2,2				$\frac{1}{N}$ to 2,2 and N to Acc.
13	73	0,4		<u>40</u>	<u>1,4</u>	Print N
14	20	0		30	0	Parameter for integer on same line
15	30	2,2		53	0,2	Form $\bar{X}$ .
16	20	3,2		53	3,2	Form $\bar{X}^2$ .
17	10	3,2		00	0	$\bar{X}^2$ to 3,2 and $\bar{X}$ to Acc.
18	73	0,4		<u>40</u>	<u>1,4</u>	Print $\bar{X}$
19	40	0		30	0	Parameter for fraction on same line
20	30	1,2		53	2,2	Form $\frac{1}{N} \sum X_i^2$ and clear A.R.
21	05	3,2		00	0	Form $(SD)^2$ .
22	73	0,5		<u>40</u>	<u>1,5</u>	Form SD.
23	73	0,4		<u>40</u>	<u>1,4</u>	Print SD.
24	40	0		30	0	Parameter for fraction on same line
25	40	0,		00	0	Return to deal with next set.

## The Complete Programme

The following indicates one way in which the above programme could be written out. The meanings of the various symbols are explained in full in the 803 Library of Programmes. In brief they are:

- = Beginning of label (i.e. title of programme).
  - bl End of label.
  - @ Beginning of directory (see 5.2.2).
  - + Beginning of a number (here, in some cases, a block address).
  - \* End of directory, or of a block.
  - ) End of block and end of tape.
- 

cr lf fs = ls MEANS AND STANDARD DEVIATIONS cr lf fs bl

@ + 192

+ 220

+ 230

+ 290

+ 430

\*

-----  
----- BLOCK 1 -----  
----- IN FULL -----  
-----

\*

+ 0

+ 0

+ 0

See note (a)

+ 0

+ 0

\*

Copy 803 T 5 with \* instead of )

See note (b)

Copy 803 T 6 with \* instead of )

-----  
-- SQUARE ROOT ROUTINE --  
-----  
)

---

The above is passed to a teleprinter operator, who prepares and checks the programme tape.

- Notes:
- (a) These +0's ensure that the locations of Block 2 are set to zero initially. It is not essential that this should be done, but, for reasons which are outside the scope of this book, it is good practice always to arrange that the working space block is clear at the start.
  - (b) The standard versions of library subroutine tapes end with ) which indicates 'end of block and end of tape'. This remark is an instruction to the teleprinter operator to copy the standard subroutine tape into our programme tape, but to change the ) to \* ( 'end of block' ).
  - (c) The information given above about Library Programmes consists of extracts from the 803 Library of Programmes relevant to our present purposes, and is not complete. Before attempting any serious programming it is important to study the complete descriptions given in the library.



## **CHAPTER 6 THE KEYBOARD**

### **6.1 Description of the Keyboard**

The keyboard of the 803 carries the word generator, the indicator lamps, the loudspeaker, the 'clear store' and other control buttons, and the 'operate' bar.

#### **6.1.1 Word Generator**

On the word generator section of the keyboard there are four rows of buttons. These contain 6, 14, 6 and 13 buttons respectively, and correspond to the F1, N1 and B (together), F2 and N2 digits of a computer word. A depressed button represents a 1, and a released button a 0. When any button is depressed it locks down: the buttons in any row may be released by depressing the 'clear' button at the end of the row.

A decimal word generator has been fitted to some early machines. This has thirteen columns of 10 buttons each, labelled 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The buttons in each column are mutually exclusive: that is to say, depressing any button in a column releases the one already depressed. This set of 130 buttons is read in function 70 in various modes, depending upon the positioning of four mutually exclusive buttons provided for the purpose. However, for the operating steps described in 6.2 below, the 'mode 1' button should be depressed, so that the word generator is in "instruction" mode; that is to say, numbering the columns from left to right:

Columns	1 and 2	represent F1 in octal
Columns	3 to 6	represent N1 in decimal
Column	7	represents B
Columns	8 and 9	represent F2 in octal
Columns	10 to 13	represent N2 in decimal

#### **6.1.2 Indicator Lamps**

The 'display' lamp is illuminated while the computer is switched on.

The 'overflow' lamp is illuminated while the overflow indicator is set.

The 'parity' lamp is illuminated and the computer stops when store parity fails. The lamp is extinguished as soon as the computer is restarted.

#### **6.1.3 Loudspeaker**

This is pulsed every time an instruction from Group 4, 5, 6 or 7 is to be obeyed, and the pitch of the note emitted therefore varies with the frequency of occurrence of such instructions in the programme

in use. There is a volume control knob adjacent to the loudspeaker.

When the computer is in a dynamic stop, the maximum pitch is reached and maintained. The same pitch is heard for a short period when the computer is waiting for a busy line to be cleared.

#### 6.1.4 Clear Store Button

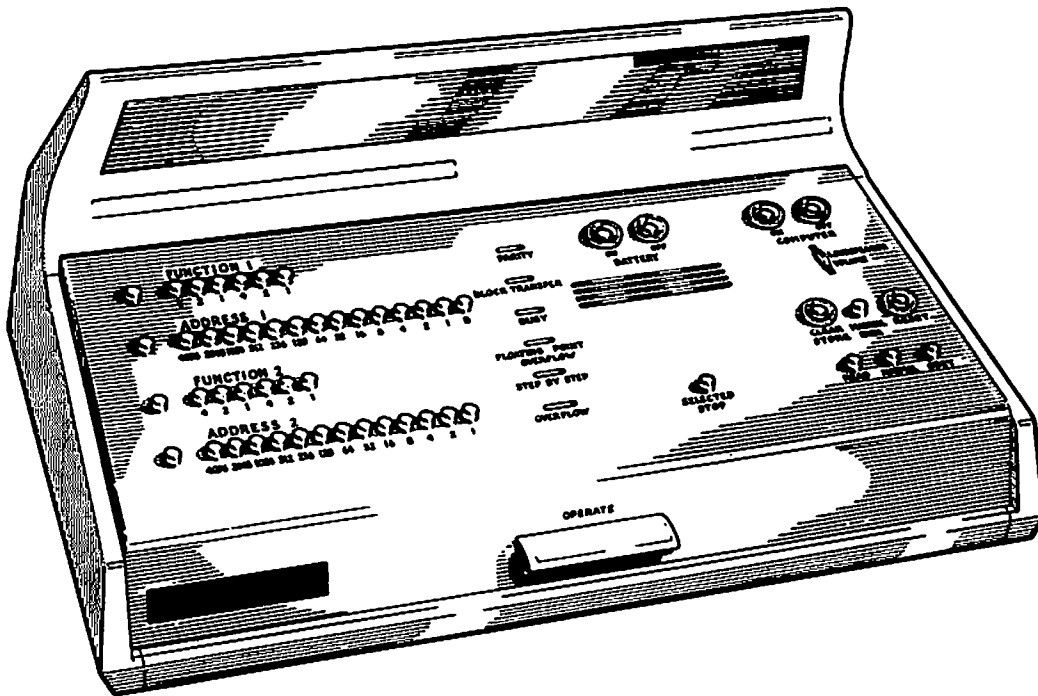
This is used to clear the store of its content, and to ensure that the parity setting in each location is correct. The effect of using it is to make all the parity digits in locations 4 to 8191 of the store 1's and all the other digits 0's. It is a press-press button: that is to say, if pressed, it locks down; if pressed again, it springs up. The method of use is given in 6.2 below.

While the 'clear store' button is in use, the computer does not stop if wrong parity is found in any location of the store.

#### 6.1.5 Control Buttons and Operate Bar

The 'read', 'normal' and 'obey' buttons, the 'selected stop' button and the 'operate' bar are the principal means by which the operator controls the computer. Their use is described in 6.2 below.

The 'read', 'normal' and 'obey' buttons are together termed the 'step-by-step' buttons: they are mutually exclusive. The 'selected stop' button is a press-press button. The 'operate' bar is spring-loaded.



## 6.2 Manual Control of the Computer

To understand the use of the control buttons and the 'operate' bar it is necessary to have a little knowledge of the way in which the computer works.

In the cycle of actions in which the computer carries out one function there are three phases, which may be summarised thus:

- (a) Determine which instruction to obey next.
- (b) Take a copy of this instruction from the store, and place it in the control unit.
- (c) Obey it.

Whenever the computer is stopped, whether by means of the control buttons or as a result of store parity failure, it is always in the position of having just completed phase (b). There is therefore an instruction in the control unit, waiting to be obeyed: this is termed the present instruction.

### Important Notes

- (a) The 'step-by-step' buttons must be pressed firmly, and only one should be pressed at any one time: otherwise the mechanical interlock between the buttons may fail, causing the computer to operate wrongly.
  - (b) Unless otherwise specified, the 'selected stop' and 'clear store' buttons must be kept in the released position.
- (i) To stop the computer immediately  
Depress either the 'read' or the 'obey' button.
  - (ii) To cause the computer, having stopped, to obey the present instruction, and stop again:  
Depress, or leave depressed, the 'obey' button, and depress the 'operate' bar once.
  - (iii) To cause the computer, having stopped, to read one instruction from the word generator, and replace the present instruction thereby.  
Depress, or leave depressed, the 'read' button, set the instruction on the F1 and N1 buttons of the word generator, and depress the 'operate' bar once.  
  
(This instruction is obeyed when either (ii) or (iv) is performed).
  - (iv) To cause the computer, having stopped, to obey the present instruction, and continue then to obey the instructions in its store at full speed:

Depress, or leave depressed, the 'normal' button, and press the 'operate' bar once.

(v) To clear the store:

Depress, in this sequence, the 'clear store' and 'normal' buttons, and the 'operate' bar. Wait at least 10 seconds. Then press the 'read' or 'obey' button and release the 'clear store' button by pressing it again.

(vi) To cause the computer, while running, to stop before obeying either instruction in a specified location:

Set the address of the specified location on the N2 buttons of the word generator and depress the 'selected stop' button. (On earlier machines it is necessary also to depress the 'obey' button.)

(vii) To cause the computer, having stopped, to start and run at full speed (as in (iv)), but to stop again before obeying either instruction in a specified location:

As in (vi), then depress the 'operate' bar.

If the computer has stopped on a selected stop, and is then restarted with the same selected stop setting still on the buttons, it will, in general, obey one instruction only, and stop again. For if the first stop was before obeying the first instruction of a pair, and if this is not a transfer instruction, the next stop is made before obeying the second instruction of the same pair.

The following practice is recommended:

If the computer is stopped while it is obeying a programme, and one or more instructions are inserted from the word generator, as in (iii) above, the return to the programme should be by the method of Example 21, below.

**Example 21**

To enter a programme at the first instruction in location X.

- (i) Depress the 'read' button.
- (ii) Set 40 X on the F1 and N1 buttons of the word generator.
- (iii) Depress the 'operate' bar.
- (iv) Depress the 'normal' button.
- (v) Depress the 'operate' bar

### **6.3 Recent Additions to the Keyboard (April 1961)**

In addition to the items described above, the latest models also have:

#### **6.3.1 Manual Data Button**

This is a push-push button. While it is depressed the word generator is "busy". That is to say, if the computer's programme contains a Function 70 instruction, the computer will stop on reaching that instruction, and will not proceed until the button is cleared. This enables the operator to take his time about putting a setting on the word generator, without needing to stop the computer altogether.

#### **6.3.2 Step-by-Step Lamp**

This is lit whenever the 803 is in the step-by-step state.

#### **6.3.3 Busy Lamp**

This is lit whenever the 803 is in the busy state: e.g. is waiting for the manual data button to be released before obeying a 70 instruction, or is held up waiting for a magnetic film backing store function to be completed.

#### **6.3.4 Other Buttons and Lamps**

The other buttons and lamps are used in conjunction with Channel 2 devices and the Automatic Floating-Point Unit. See separate publications, and also Appendix 5.



# APPENDIX 1

## The ELLIOTT Telecode

<u>Tape</u>	<u>Binary Value</u>	<u>Decimal Value</u>	<u>Character Indicated</u>	
			<u>Figure Shift</u>	<u>Letter Shift</u>
.	00000	0		Blank
. o	00001	1	1	A
. o	00010	2	2	B
. oo	00011	3	*	C
.o	00100	4	4	D
.o o	00101	5	\$	E
.oo	00110	6	=	F
.ooo	00111	7	7	G
o.	01000	8	8	H
o. o	01001	9	'	I
o. o	01010	10	'	J
o. oo	01011	11	+	K
o.o	01100	12	:	L
o.o o	01101	13	-	M
o.oo	01110	14	.	N
o.ooo	01111	15	%	O
o .	10000	16	0	P
o . o	10001	17	(	Q
o . o	10010	18	)	R
o . oo	10011	19	3	S
o .o	10100	20	?	T
o .o o	10101	21	5	U
o .oo	10110	22	6	V
o .ooo	10111	23	/	W
oo.	11000	24	@	X
oo. o	11001	25	9	Y
oo. o	11010	26	£	Z
oo. oo	11011	27	Figure Shift	
oo.o	11100	28	Space	
oo.o o	11101	29	Carriage Return	
oo.oo	11110	30	Line Feed	
oo.ooo	11111	31	Letter Shift	

# APPENDIX 2

## Powers of 2 in Decimal

$2^n$	n	$2^{-n}$
2	1	.5
4	2	.25
8	3	.125
16	4	.062 5
32	5	.031 25
64	6	.015 625
128	7	.007 812 5
256	8	.003 906 25
512	9	.001 953 125
1 024	10	.000 976 562 5
2 048	11	.000 488 281 25
4 096	12	.000 244 140 625
8 192	13	.000 122 070 312 5
16 384	14	.000 061 035 156 25
32 768	15	.000 030 517 578 125
65 536	16	.000 015 258 789 062 5
131 072	17	.000 007 629 394 531 25
262 144	18	.000 003 814 697 265 625
524 288	19	.000 001 907 348 632 812 5
1 048 576	20	.000 000 953 674 316 406 25
2 097 152	21	.000 000 476 837 158 203 125
4 194 304	22	.000 000 238 418 579 101 562 5
8 388 608	23	.000 000 119 209 289 550 781 25
16 777 216	24	.000 000 059 604 644 775 390 625
33 554 432	25	.000 000 029 802 322 387 695 313
67 108 864	26	.000 000 014 901 161 193 847 656
134 217 728	27	.000 000 007 450 580 596 923 828
268 435 456	28	.000 000 003 725 290 298 461 914
536 870 912	29	.000 000 001 862 645 149 230 957
1 073 741 824	30	.000 000 000 931 322 574 615 479
2 147 483 648	31	.000 000 000 465 661 287 307 739
4 294 967 296	32	.000 000 000 232 830 643 653 870
8 589 934 592	33	.000 000 000 116 415 321 826 935
17 179 869 184	34	.000 000 000 058 207 660 913 467
34 359 738 368	35	.000 000 000 029 103 830 456 734
68 719 476 736	36	.000 000 000 014 551 915 228 367
137 438 953 472	37	.000 000 000 007 275 957 614 183
274 877 906 944	38	.000 000 000 003 637 978 807 092
549 755 813 888	39	.000 000 000 001 818 989 403 546



### APPENDIX 3

#### The 803 Instruction Code

These tables give the complete instruction code of the 803. Some special notes follow.

Unless otherwise stated the N digits of an instruction specify the address of a location.

In the "Result" columns a' and n' indicate "new contents" of the accumulator and location N. a and n likewise indicate "old contents".

	<u>Function</u>	<u>Operation</u>	<u>Result</u>	
			a'	n'
<u>GROUP 0</u>	00	Do nothing	a	n
	01	Negate	-a	n
	02	Replace and count	n + 1	n
	03	Collate	a & n	n
	04	Add	a + n	n
	05	Subtract	a - n	n
	06	Clear	zero	n
	07	Negate and add	n - a	n
<u>GROUP 1</u>	10	Exchange	n	a
	11	Exchange and negate	-n	a
	12	Exchange and count	n + 1	a
	13	Write and collate	a & n	a
	14	Write and add	a + n	a
	15	Write and subtract	a - n	a
	16	Write and clear	zero	a
	17	Write, negate and add	n - a	a
<u>GROUP 2</u>	20	Write	a	a
	21	Write negatively	a	-a
	22	Count in store	a	n + 1
	23	Collate in store	a	a & n
	24	Add into store	a	a + n
	25	Negate store and add to store	a	a - n
	26	Clear store	a	zero
	27	Subtract from store	a	n - a

	<u>Function</u>	<u>Operation</u>	<u>Result</u>	
			<u>a'</u>	<u>n'</u>
<u>GROUP 3</u>	30	Replace	n	n
	31	Replace and negate store	n	-n
	32	Replace and count in store	n	n + 1
	33	Replace and collate in store	n	a & n
	34	Replace and add to store	n	a + n
	35	Replace, negate store and add	n	a - n
	36	Replace and clear store	n	zero
	37	Replace and subtract from store	n	n - a

N.B. The "1" added in count functions is to scale  $2^{-38}$ .

The time taken to obey any instruction of Group 0 to 3 is 720 microseconds in early machines, and 576 microseconds in machines completed after December 1960.

#### GROUP 4

<u>Function</u>	<u>Operation</u>
40	Transfer, unconditionally, to first instruction.
41	Transfer to first instruction if C(A) is negative.
42	Transfer to first instruction if C(A) is zero.
43	Transfer to first instruction if the overflow indicator is set. Clear the overflow indicator.
44	Transfer, unconditionally, to second instruction.
45	Transfer to second instruction if C(A) is negative.
46	Transfer to second instruction if C(A) is zero.
47	Transfer to second instruction if the overflow indicator is set. Clear the overflow indicator.

If a transfer is made to the second instruction in a location, that instruction is obeyed as stored, irrespective of the state of the B digit.

If a transfer instruction is placed in the first half of an instruction pair, the B digit in this pair must be zero.

The time taken to obey any instruction of Group 4 is 720 microseconds in early machines, and 288 microseconds in machines completed after December 1960.

## GROUP 5

In functions 52, 53 and 56 the  $N$  digits specify the address of a location in the store. In functions 50, 51, 54 and 55, the seven right-hand  $N$  digits specify a number  $N$ , which therefore lies between 0 and 127 inclusive. In function 57 the  $N$  digits are not used.

By "double-length number" is meant the combined content of the accumulator and the auxiliary register (A.R.), treated as one number of 1 sign and 76 fractional digits.

In the column "Time Taken", the upper time is for early machines, the lower time for machines completed after December 1960.

Function	Operation	Time Taken
50	Halve the double-length number $N$ times (with sign digit regeneration).	$\begin{cases} (N+2) \times 720 \text{ } \mu\text{sec.} \\ (N+2) \times 288 \text{ } \mu\text{sec.} \end{cases}$
51	Shift C(A) right $N$ places (without sign digit regeneration). Clear A.R.	$\begin{cases} (N+2) \times 720 \text{ } \mu\text{sec.} \\ (N+2) \times 288 \text{ } \mu\text{sec.} \end{cases}$
52	Multiply C(A) by C(N), with double-length product.	$\begin{cases} 41 \times 720 \text{ } \mu\text{sec.} \\ (42-Y) \times 288 \text{ } \mu\text{sec.}^* \end{cases}$
53	Multiply C(A) by C(N), with single-length rounded product. Clear A.R.	$\begin{cases} 41 \times 720 \text{ } \mu\text{sec.} \\ (43-Y) \times 288 \text{ } \mu\text{sec.}^* \end{cases}$
54	Double the double-length number $N$ times.	$\begin{cases} (N+2) \times 720 \text{ } \mu\text{sec.} \\ (N+2) \times 288 \text{ } \mu\text{sec.} \end{cases}$
55	Double C(A) $N$ times. Clear A.R.	$\begin{cases} (N+2) \times 720 \text{ } \mu\text{sec.} \\ (N+2) \times 288 \text{ } \mu\text{sec.} \end{cases}$
56	Divide the double-length number by C(N), with single-length quotient. Clear A.R.	$\begin{cases} 41 \times 720 \text{ } \mu\text{sec.} \\ 42 \times 288 \text{ } \mu\text{sec.} \end{cases}$
57	Read A.R. (Clear accumulator sign digit, and replace other digits of C(A) by a copy of the A.R. content).	$\begin{cases} 720 \text{ } \mu\text{sec.} \\ 576 \text{ } \mu\text{sec.} \end{cases}$

- \* Y is the number of consecutive 1's 0's at the left hand end of the content of store. e.g. if  $C(N) = 10 \times 2^{-38}$ , Y is 35.\*

## GROUP 6

Spare. In the basic 803, all instructions of Group 6 are interpreted as "do nothing", and take 720 microseconds to obey in early machines, and 576 microseconds in machines completed after December 1960. See Appendix 5 for Automatic Floating Point functions.

## GROUP 7

In function 73 the N digits specify the address of a location. In function 74 the five right-hand N digits specify a telecode character by its "value". In functions 70 and 71 the N digits are not used. The other functions of this group vary according to the type of input and output equipment fitted to the individual 803; this table gives outline details of a typical arrangement.

<u>Function</u>	<u>Operation</u>
70	Read the Number Generator.
71	Read the Tape Reader (normally attached to Channel 1) (Mix one character into the 5 right-hand positions of C(A).)
72	Channel 2 function (See Note below).
73	Write the address of this instruction (in the right-hand part of location N).
74	Punch specified character on Output Channel 1.
75	Channel 2 function (See Note below).
76	Channel 2 function (See Note below).
77	Channel 2 function (See Note below).

The time taken to perform function 70, 71, 73 or 74 is 720 microseconds provided the associated busy line (if any) is free, on early machines, and 576 microseconds on machines completed after December 1960.

Note on Channel 2. Separate publications relating to equipment which makes use of Channel 2 should be consulted for the interpretation of functions 72, 75, 76 and 77, but it should be noted that if a function 77 instruction is placed in the first half of an instruction pair, the B digit in that pair must be zero.

### Special Notes

#### (a) The B Digit

If the B digit is a 1, the right-hand instruction is modified by the addition of the corresponding part of the content of the location specified by N1, the N digits of the left-hand instruction, before being obeyed.

(b) Addresses 0 to 3

Locations 0 to 3 contain the Initial Instructions; if a transfer is made to location 0, 1, 2 or 3, the computer endeavours to obey this routine. If location 0, 1, 2 or 3 is used as B-line, the modification is by addition of the appropriate initial instruction.

If any instruction specifies the content of location 0, 1, 2 or 3 as operand, the operand used is zero: further it is not possible to change the content of location 0, 1, 2 or 3. Thus the instruction 12 2 has no effect other than to change  $C(A)$  to  $2^{-38}$ .

(c) Altering the content of a location by an instruction contained therein

This note is included for completeness: the situation described arises very infrequently.

If the left-hand instruction in a location calls for the alteration of the content of that same location in such a way that the right-hand instruction is changed, then, if the B digit is an 0, the right-hand instruction is obeyed in its new form. But if the B digit is a 1, the instruction actually carried out is the old form modified by the new.

Example: If  $C(250)$  is 22 250:14 10, the computer obeys it as 22 250 and then 14 11. But if  $C(250)$  is 22 250 / 14 10, the computer obeys 22 250 and then 30 21 (i.e.  $14\ 10 + 14\ 11$ ).

## APPENDIX 4

This appendix is in two parts:

- A. A list of all functions which can cause overflow.\*
- B. A note on the accuracy of the division process.

### A. Functions which can cause Overflow \*

<u>Title</u>	<u>Function</u>	<u>Circumstances causing Overflow</u>
Negate	01, 11, 21, 31	In the case where the number to be negated is -1.
Count	02, 12, 22, 32	When C(N) is $1 - 2^{-38}$ .
Add	04, 14, 24, 34	Whenever the sum is outside the range -1 to $1 - 2^{-38}$ inclusive.
Subtract	05, 15, 27, 37	Whenever the difference is outside the range -1 to $1 - 2^{-38}$ inclusive.
Negate and add	07, 17, 25, 35	
Multiply	52, 53	In the case of $-1 \times -1$ only.
Double	54, 55	Whenever the correct result lies outside the range -1 to $1 - 2^{-38}$ inclusive.
Divide	56	Whenever the modulus of the numerator (dividend) is greater than that of the denominator (divisor) and in the three cases shown in B(c) below.

\* This does not include "Floating-Point Overflow": See Appendix 5

B. The Division Process. This note concerns the accuracy of the result produced by the computer when the true quotient lies within the range -1 to +1 inclusive, in function 56.

(a) Since the result is not rounded, when the true quotient is not exactly expressible as a 39-digit computer number, the result may be  $2^{-38}$  less than that which would be obtained by using a process containing a round-off stage.

(b) If a and b are both positive, and  $b > a$ , and if the quotient is exactly expressible as a 39-digit computer word,

then for

$a/b, -a/b, 0/b$  : the computer results are correct,

but for

$a/-b$ ,  $-a/-b$ ,  $0/-b$  : the computer results are  $2^{-38}$  less than the arithmetically correct results.

(c) For the division  $\pm a/\pm a$ ,  $a > 0$ , the effects of (b), combined with the fact that there is no computer representation of +1, cause the results shown below to be produced. Note also the result obtained for  $0/0$ , and that the overflow indicator is set for this case, and for both cases in which the result of  $\pm a/\pm a$  errs by more than  $2^{-38}$ .

<u>Division</u>	<u>Computer Result</u>	<u>Overflow Indicator</u>
$a/a$	-1	Set
$-a/a$	-1	Not Set
$a/-a$	$1-2^{-38}$	Set
$-a/-a$	$1-2^{-38}$	Not Set
$0/0$	$-2^{-38}$	Set

## APPENDIX 5

### THE AUTOMATIC FLOATING-POINT UNIT

#### 1. INTRODUCTION

*Floating-point representation* is the term used to describe a certain method of representing numbers within a computer. This method is akin to the notation by which such a number as .000000000135 is represented by writing  $1.35 \times 10^{-10}$ .

The purpose of such a notation is to enable very small or very large numbers to be represented without the need to write long strings of figures. In the same way, the purpose of floating-point representation within the computer is to enable very small and very large numbers to be represented by a limited numbers of digits - by one computer word, in fact.

This representation can be used in any 803, but in the basic machine it is necessary to use subroutines to perform even such simple operations as addition, subtraction etc. However, if an automatic floating-point unit is fitted, it is possible to carry out arithmetic on floating-point numbers by means of special functions provided for the purpose.

#### 2. FLOATING-POINT REPRESENTATION

##### 2.1 General

Any given number  $A$  can be represented in many ways by a pair of numbers  $(a, b)$  which satisfies the equality

$$A = a \times 2^b$$

in which  $a$  is called the mantissa or argument and  $b$  the (binary) exponent. Such a representation is termed (binary) *floating-point*, by contrast with the forms discussed in the main part of this manual, e.g. integers, fractions, etc, all of which are *fixed-point*.

For example, the number 6 can be represented by  $(6, 0)$  or by  $(.75, 3)$  or by  $(12, -1)$ .

The usefulness of the notation lies, however, in the fact that any given non-zero number can be represented by a pair of numbers in which the magnitude of the mantissa lies between  $\frac{1}{2}$  and 1 and the exponent is integral. In practice when using this notation in the computer there are upper and lower limits on the value of the exponent,



and so there are upper and lower limits on the magnitude of the non-zero numbers which can be represented.

Zero can, of course, be represented by any pair in which the mantissa is 0.

## 2.2 Standard Floating-Point Representation within 803

In 803 computers *fitted with automatic floating-point units* the details of the representation used are as follows:

- (i)  $b$  is an integer,  $-256 \leq b \leq 255$
- (ii) if  $A > 0$ ,  $\frac{1}{2} \leq a < 1$   
           if  $A = 0$ ,  $a = 0$ ,  $b = -256$ , always  
           if  $A < 0$ ,  $-1 \leq a < -\frac{1}{2}$

(iii) 30 digits representing the mantissa,  $a$ , and 9 digits representing the exponent,  $b$ , are "packed" together in one word.  $a$  is on the left, and is represented in the same way as a normal fixed point fraction: that is to say, the left hand digit is the sign digit, the second is the  $2^{-1}$  digit, the third is the  $2^{-2}$  digit, and so on down to the thirtieth, which is the  $2^{-29}$  digit; while if  $a$  is negative, it is represented by  $(2 - |a|)$ . The remaining 9 digits represent the integer  $(b+256)$  directly; this must satisfy the relation  $0 \leq (b+256) \leq 511$ .

## 2.3 Examples of floating-point numbers

For ease of reading, the numbers in the examples below have been printed with the sign digit, the fractional digits of the mantissa and the digits of the exponent in separate groups.

Zero

0 00000000000000000000000000000000 000000000

-1

1 00000000000000000000000000000000 100000000

+1

0 10000000000000000000000000000000 100000001

+ 240, i.e.  $\frac{15}{16} \times 2^8$

0 11110000000000000000000000000000 100001000

-.078125, i.e.  $(-\frac{5}{8}) \times 2^{-3}$

1 01100000000000000000000000000000 011111101

$+(1 - 2^{-29}) \times 2^{255}$ , the largest possible positive number, about  $5.8 \times 10^{76}$

0 11111111111111111111111111111111 11111111

$-(\frac{1}{2} + 2^{-29}) \times 2^{-256}$ , the smallest possible negative number, about  $-4.3 \times 10^{-78}$

1 01111111111111111111111111111111 000000000

Observe that zero has the same form in floating-point representation as in fixed-point, and that the sign digit of any positive number or zero is 0, while that of any negative number is 1.

#### 2.4 Accuracy, Range and Round-Off

The accuracy of any representation is determined by the number of significant figures employed. Here this is 29 binary digits, so the accuracy is slightly less than that which would be expected from 9 significant decimal figures.

The last two examples given above show extreme values of numbers which can be represented. The actual range is defined by:

Zero	represented exactly and unambiguously		
Largest positive number:	$(1 - 2^{-29})$	$\times 2^{255}$	} representation accurate to 29 significant binary digits
Smallest positive number:	$\frac{1}{2}$	$\times 2^{-256}$	
Largest negative number:	-1	$\times 2^{255}$	
Smallest negative number:	$-(\frac{1}{2} + 2^{-29})$	$\times 2^{-256}$	

This may be summarised approximately by saying that zero is represented exactly and that any number  $A$  satisfying

$$4.3 \times 10^{-78} \leq |A| \leq 5.8 \times 10^{76}$$

can be represented to an accuracy approximately equal to that obtained by 9 significant decimal figures.

Certain rational numbers in this range can, of course, be represented exactly. These include, in particular, all integers in the range  $-536\ 870\ 912 \leq n \leq 536\ 870\ 911$ .

In each floating-point operation executed by the computer the result is rounded off *without bias* in such a way that its mantissa will not differ by more than  $2^{-29}$  from the correct result. If the true result of any floating-point operation upon two numbers, which are represented exactly, is itself capable of exact representation, then the result actually produced by the computer will be exactly correct.

It will be appreciated that the value of the exponent determines the absolute magnitude of the smallest increase or decrease in any number which can be represented within the computer. Thus, in general, the greater the magnitude of any particular number, the greater the step to the next representable number.

## 2.5 Floating-Point Underflow and Overflow

### The Floating-Point Overflow Lamp

(i) If the computer is called upon to execute any floating-point arithmetic operation, the correct result of which is of such *small* magnitude that it cannot be represented, the actual result generated will be zero. This effect is termed *floating-point underflow*, and does not affect the running of the computer, or any indicator lamp.

(ii) If the computer is called upon to execute any floating-point arithmetic operation, the correct result of which is of such *great* magnitude that it cannot be represented, it will stop, and the *floating-point overflow lamp* on the keyboard will be lit. The computer can be restarted by depressing the operate bar. *The result actually generated will be spurious.*

(iii) Floating point functions do not affect the fixed point overflow indicator on later machines\*.

## 2.6 Difference between automatic and programmed floating-point representation. Cautionary note

It should be observed that, before the advent of the 803 automatic floating-point unit, many subroutines and programmes were written by means of which floating-point operations could be performed, and that these remain in use on 803's not fitted with automatic floating-point units. In most of these programmes, the representation employed differs to some extent from the representation described above.

*This description applies to the automatic floating-point unit only.*

## 3. THE FLOATING-POINT INSTRUCTION CODE

### 3.1 Group 6 Functions : Automatic Floating-Point Arithmetic

In functions 60 to 64 inclusive the computer treats both *a* and *n* as standard floating-point numbers and produces a standard floating-point result, which replaces the old content of the accumulator.

The content of the auxiliary register is cleared by any of these operations, but the store is not affected.

<u>Instruction</u>	<u>Effect</u>	<u>a'</u>	<u>Time Taken</u>
60 N	Add n to a	$a + n$	3 x 288 $\mu$ sec.
61 N	Subtract n from a	$a - n$	3 x 288 $\mu$ sec.
62 N	Negate a and add n	$n - a$	3 x 288 $\mu$ sec.
63 N	Multiply a by n	$an$	17 x 288 $\mu$ sec.
64 N	Divide a by n	$a/n$	34 x 288 $\mu$ sec.
65 4096	Convert the fixed-point integer in the accumulator to floating-point from		2 x 288 $\mu$ sec.

## Conversion

In obeying the instruction 65 4096 the computer treats the existing content of the accumulator as a fixed-point integer to scale  $\times 2^{-38}$ , and converts this to standard floating-point form.

**For example:**

[illegible]

which is the fixed-point representation of the integer 15

is converted to

[illegible]

in which  $a = \frac{15}{16}$  and  $(b+256) = 260$

so that  $a \times 2^b = \frac{15}{16} \times 2^4 = 15$

It should be noted that function 65, with values of N  $\neq$  4096, is used for other purposes in certain special versions of 803, and that 65 4096 is the only permitted form of the instruction to convert an integer to floating-point representation.

## Negating

Observe that if C(Acc) is a floating-point number it may be negated by means of the instruction 62 0.

### 3.2 Other Functions which can be used straightforwardly with Floating-Point Numbers

<u>Functions</u>				<u>Remarks</u>
00	10	20	30	The effect is normal
03	13	23	33	Can be used to separate the exponent and mantissa: otherwise not of much practical value
06	16	26	36	The effect is normal
41	42	45	46	Will work equally well when C(A) is a floating-point number

### **3.3 Other Functions which will not, in general, produce sensible results in floating-point form**

These functions are:

01. 11, 21, 31,  
02. 12, 22, 32,  
04. 14, 24, 34,  
05. 15, 25, 35,  
07. 17, 27, 37,  
43, 47.

All Group 5

Where the effect of the above instructions is to perform arithmetic on one or more words which represent floating-point numbers, they will not, in general, produce sensible results. The functions 43 and 47 refer to the fixed-point overflow indicator, which is not usually affected by floating-point overflow: see 2.6 (iii).

It is, however, possible to carry out such actions as using 22 N to double a floating-point number in the store once, or using a fixed-point add or subtract instruction to double or halve several times by adding or subtracting an integer (thereby changing  $A = a.2^b$  to  $a.2^{b+n} = A.2^n$ ). It should be noted that these operations would not be subject to the normal rules of floating-point overflow and underflow.

## **4. CONVENTIONS USED IN AUTOMATIC FLOATING-POINT PROGRAMMES**

### **4.1 Floating-Point Decimal Notation**

Just as with fixed-point programming, constants and data which are to be operated on in binary in the computer, are expressed in decimal on programme sheets and when writing out data.

*While the actual details of the conventions used will vary from one subroutine or programme to another, the general written notation for a floating-point number is  $\pm a/b$ , representing  $\pm a \times 10^b$ .  $b$  must be an integer,  $a$  may be a fraction, integer or mixed number.*

Thus we could express the floating-point number

123.45

by any of these:

- (i)  $+.12345/3$
- (ii)  $+12345/-2$
- (iii)  $+123.45/$

Of these, the first is sometimes called the standard (decimal) floating-point form, which is roughly defined by saying that the mantissa,  $a$ , obeys the decimal inequality

$$.1 \leq |a| < 1$$

In the second example, where there is no decimal point, the mantissa is assumed to be an integer. In the third case, where there are no figures after the  $/$ , the exponent is assumed to be zero.

#### **4.2 Floating-point Translation Input Routine**

An augmented translation input routine is available, which will read and store floating-point constants written in any of the above notations. This routine occupies more storage space than the standard form, so programmes written for automatic floating-point computers can conveniently commence in location 256.

#### **4.3 Floating-point Subroutines**

Subroutines which will perform the functions of reading and printing floating-point numbers, including if desired the conversion from or to fixed-point form, are available. So also are subroutines to evaluate mathematical functions of floating-point numbers.

### **5. A COMPLETE FLOATING-POINT PROGRAMME**

#### **WARNING**

At the time (May 1961) of writing this text, the subroutines to which reference is made exist only in a provisional form. It should therefore be noted that the final details may differ from those assumed here.

#### **5.1 We assume that the following are available:**

- (i) An augmented translation input routine, of about 220 words length, which will read and store floating-point constants.

This may also be entered from any programme by the instruction pair 73 170 40 53 whereupon it will read one number (fixed or floating-point) from the input tape, and exit with this in the accumulator.

- (ii) A print routine which will, among other things, print a floating-point number in standard decimal floating-point form. The entry is standard, and the parameter 00 0 00  $n$  must be written in the location immediately following that holding the entry instructions, to specify that  $n$  decimal digits are required in the mantissa of the printed number.

Storage requirements: less than 200 locations.

- (iii) A floating-point square-root subroutine, which will extract the square root of the floating-point number in the accumulator. Entry is standard, and not more than 25 locations are required.

## 5.2 The problem to be Programmed

Given: The integer  $r$ , and the surface areas of  $r$  spheres.

Calculate and print, to five significant figures, the volume of each sphere and the mean volume per sphere.

## 5.3 Notation, Formulae, etc

The integer  $r$  is the number of spheres.

$r'$  is the standard floating-point form of  $r$ .

If the area of a sphere is  $A$ , then its volume  $V$  is determined by

$$V = A \sqrt{A} / 6 \sqrt{\pi}$$

and we note that  $6\sqrt{\pi} = 10.6347231$

If  $T$  is the total volume, then

$$T = \sum V = 0 + V_1 + V_2 + \dots + V_r$$

and if  $U$  is the mean volume, then

$$U = T / r'$$

## 5.4 Preliminary Work

<u>Blocks</u>	1 Main Programme
	2 Constants
	3 Workspace
	4 Print subroutine
	5 Square-root subroutine

(Input of data will be done by means of the augmented translation input routine.)

## Data Sequence

The following numbers will be punched on the data tape, in this sequence:

The integer  $r$

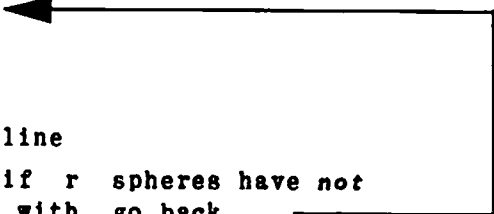
The floating-point numbers, representing the surface areas, one after the other.

### Output Format

The individual volumes will be printed in a column, and the mean volume will be printed to the right of the last individual volume.

### 5.6 Scheme, or Flow Diagram.

```
Read r
Set count  $-(r-1)$ 
Form  $r'$  and store it
Set T to zero
Read an area A
Form V
Add V to T
Print V on a new line
Count and test : if r spheres have not
                  yet been dealt with, go back
Otherwise: divide T by  $r'$  to form U
Print U
Stop
```



```
graph TD
    ReadR[Read r] --> SetCount[Set count -(r-1)]
    SetCount --> FormRPrime[Form r' and store it]
    FormRPrime --> SetT[Set T to zero]
    SetT --> ReadA[Read an area A]
    ReadA --> FormV[Form V]
    FormV --> AddV[Add V to T]
    AddV --> PrintV[Print V on a new line]
    PrintV --> CountTest[Count and test : if r spheres have not yet been dealt with, go back]
    CountTest --> ReadA
    CountTest --> DivideT[Otherwise: divide T by r' to form U]
    DivideT --> PrintU[Print U]
    PrintU --> Stop[Stop]
```



## 5.6 Programme

ADDRESS	INSTRUCTIONS	REMARKS
<u>BLOCK</u>	@	<u>DIRECTORY</u>
1	+256	Main Programme
2	+280	Constants
3	+285	Workspace
4	+290	Print subroutine
5	+520	Square root subroutine
	*	<u>BLOCK 1</u>
0,1	73 170 40 53	Read r
1,1	21 0,3 22 0,3	-(r-1) in 0,3
2,1	65 4096 20 1,3	r' in 1,3
3,1	26 2,3 74 27	T to zero, punch fs <sup>1</sup>
4,1	73 170 40 53	Read an area A
5,1	20 3,3 00 0	Copy of A in 3,3
6,1	73 0,5 40 1,5	Form $\sqrt{A}$
7,1	63 3,3 64 0,2	Form V
8,1	10 2,3 60 2,3	Add to T
9,1	10 2,3 00 0	(V back in accumulator)
10,1	74 29 74 30	Punch cr lf <sup>1</sup>
11,1	73 0,4 40 1,4	Punch V to
12,1	00 0 00 5	) 5 significant figures
13,1	32 0,3 41 4,1	Count spheres and test
14,1	30 2,3 64 1,3	Form U
15,1	74 28 74 28	Punch sp sp <sup>1</sup>
16,1	73 0,4 40 1,4	Punch U to
17,1	00 0 00 5	) 5 significant figures
18,1	40 18,1 00 0	Stop
	*	<u>BLOCK 2</u>
0,2	+10.6347231/	$6\sqrt{\pi}$

ADDRESS	INSTRUCTIONS	REMARKS
0,3 1,3 2,3 3,3	• +0 +0 +0 +0	<u>BLOCK 3</u>  Count of number of spheres r' T A
	•  Copy Floating-Point Print Subroutine without )	<u>BLOCK 4</u>
	•  Copy Floating-Point Square Root Subroutine with )	<u>BLOCK 5</u>

Note 1      The first character to be punched is fs,  
to ensure that the results will be printed as figures.  
Before each V, the characters cr lf are punched,  
to make the teleprinter "start a new line" before  
printing V. Before U we have sp sp: the student  
may work out why for himself.