1st Edition *August* 1958
2nd Edition *December* 1959
3rd Edition *September* 1962

PREFACE

Part 1 of this booklet contains a general introduction to the more important features of the order code of EDSAC 2 and to the programming facilities provided. It is designed to be read by the beginner. Those with experience of programming for other machines will find that they can read Part 1 rapidly.

For many problems a simpler system of programming known as *Autocode* will suffice. Details of this will be found in *A Beginner's Guide to the EDSAC 2 Autocode* by R. K. Livesley, and *EDSAC 2 Autocode Programming Manual* by D. F. Hartley. Both these documents are available from the Mathematical Laboratory.

Part 2 contains full details of the order code, the facilities provided by the program assembly routine, the error diagnosis facilities, and details of the peripheral equipment. It is intended for reference rather than for continuous reading.

EDSAC 2 was built with the aid of a generous grant from the Nuffield Foundation. I would be glad to hear of any errors or omissions that readers may notice in this booklet.

M. V. WILKES
Director

Part 1

INTRODUCTION TO PROGRAMMING FOR EDSAC 2

1 Introduction

From the point of view of the programmer the main units of a digital computer such as EDSAC 2 are the *store*, the *arithmetic unit*, and the *input* and *output* mechanisms. A calculation intended to be performed by the machine must be expressed as a sequence of operating *orders* or *instructions*, each calling for one of the elementary operations (such as addition or multiplication) that can be performed in the arithmetic unit. The complete schedule of orders is called the *program*.

The store is used to hold both the numbers needed during the course of the calculation, and also the orders comprising the program. This is made possible by expressing the orders in a coded numerical form. In EDSAC 2 there are in fact four stores, although only one of them—the *free* store—need concern the newcomer to programming.

2 The Free Store

The free store* consists of 1024 *registers* each capable of holding one number or two orders. A single order thus occupies a *half-register* or *storage location*. For purposes of identification each half-register is labelled by a number called its *address*; these addresses run from 0 to 2047 and the addresses of the two half-registers forming a whole register are always an even number and the odd number following it—that is, $2q$ and $2q + 1$ where q is an integer. The whole register is referred to by the even address $2q$. This leads to no confusion since it will be found that it is always clear whether we are talking about a half-register or a whole register. It is sometimes convenient to use the term *word* to describe the content of a register without specifying whether it is a number or a pair of orders. The notation $C(q)$ will be used to denote the content of half-register q, and $F(q)$ for the content of register q; in the latter case q must be even.

* Abbreviated to "the store" throughout the remainder of Part 1 of this booklet.

3 The Arithmetic Unit

The arithmetic unit contains equipment for performing additions, subtractions, multiplications and divisions, and is thus analogous to a desk calculating machine. It contains a register which plays the part of the result register in such a calculating machine. This register is known as the *accumulator* since the cumulative sum of a sequence of numbers can be formed in it. The content of the accumulator will be written $F(Acc)$. Ordinarily, the effect of any arithmetic operation is to change the value of $F(Acc)$.

4 Input and Output

The principal input and output medium of EDSAC 2 is punched paper tape. The machine is equipped with two photo-electric tape readers for taking information into the machine, and with two output punches for punching the results. Output tapes from these punches are passed into a mechanical tape reader controlling a teleprinter which gives a printed statement of the results obtained. Input tapes are prepared on a keyboard perforator and must have punched on them not only the program, but also any numbers which may be required by the machine during the course of the calculation.

5 Numbers

In all digital computers some restriction is imposed on the range of values which numbers may take. In EDSAC 2 we normally work with floating-point numbers which must lie (roughly) within the range $-10^{40} \leqslant x \leqslant 10^{40}$; each number is expressed with a precision approximately equivalent to 9 significant figures. It is also possible in EDSAC 2 to work with fixed-point numbers which lie in the range $-1 \leqslant x < 1$ and have a precision of approximately 12 decimals, but this facility will not be made use of in Part 1 of this booklet.

6 Order Code

In order to be able to write programs for a specified machine a programmer must have a knowledge of its *order code*; that is, he must know what are the various elementary operations that the machine can perform, and how they are called for. Orders in EDSAC 2 are of the *one-address* type; that is, each order which refers to the store (some do not) refers only to one register in the store. In the following sections the various available orders are introduced gradually so that the reader will become accustomed, by practice, to some of the more commonly used orders without having to master the whole order code at once. The complete order code is given for reference in Part 2, Chapter 2.

7 Storage of Orders

Orders are normally placed in the store in half-registers numbered in the sequence in which the orders are to be obeyed. The machine is so designed that, after the order in half-register q has been executed, the order in half-register $q + 1$ is taken next unless the order in half-register q has specified otherwise (see Section 10).

8 Written Form of Orders

When orders are written, the function and the address are written as two integers, normally separated by the letter f. The written forms of some of the more important arithmetic operations are as follows:

$10\,f\,q$	place in the accumulator the number in storage register q.
$11\,f\,q$	place in the accumulator *minus* the number in storage register q.
$12\,f\,q$	add to the number in the accumulator the number in storage register q.
$13\,f\,q$	subtract from the number in the accumulator the number in storage register q.
$14\,f\,q$	multiply the number in the accumulator by the number in storage register q.
$15\,f\,q$	divide the number in the accumulator by the number in storage register q.
$19\,f\,q$	copy the number in the accumulator into storage register q.

In all these, q stands for the address of a whole register in the store—that is, q is an even integer in the range 0 to 2046 inclusive; it is called the *address part* of the order. For orders with function numbers 10 to 15 the result of the operation is placed in the accumulator.

The contents of the store are not affected by any of these orders, except the order with function number 19, and this order does not change the content of the accumulator.

An order with function number x is usually called an *x order*.

The 10 order is said to *set* $F(q)$, the number in storage register q, in the accumulator; similarly, the 11 order is said to set $-F(q)$ in the accumulator. These orders are sometimes referred to as "clear and add" and "clear and subtract" orders. They delete the previous content of the accumulator, and this is lost unless a copy has previously been placed in some register in the store.

9 Some Simple Examples

We are now able to write down the sequences of orders required to perform some simple calculations. These orders are to be thought of as forming parts of larger programs; it is assumed that the

numbers they operate on have been calculated and placed where they are at an earlier stage of the work. For the reader's convenience we note, with each order, the operand (that is, the number in the store to which the order refers) and also the number in the accumulator after the order has been obeyed. This information is included purely for purposes of explanation and does not form part of the program as it would be read into the machine.

In writing our sequences of orders we draft them in such a way that the required calculation is carried out regardless of the content of the accumulator before the first order of the sequence is obeyed. This usually requires that the sequence starts with a 10 or 11 order.

Example 1. Given $x = F(200)$,
$y = F(202)$,

(that is, the numbers x and y are in registers 200 and 202 respectively) to form $x + y$ and place it in 204.

Order	*Operand*	*F(Acc)*
10 *f* 200	x	x
12 *f* 202	y	$x + y$
19 *f* 204	$x + y$	$x + y$.

Since the machine takes orders in sequence from the store these orders must be placed in successive half-registers; thus, if the first is placed in half-register 100, the second and the third will go into half-registers 101 and 102 respectively. We thus have:

Storage location	*Order*
100	10 *f* 200
101	12 *f* 202
102	19 *f* 204

Example 2. Given $a = F(300)$,
$b = F(302)$,
$x = F(350)$,
$y = F(352)$;

to place ax in 420 and $y(ax + by)$ in 430.

Location	*Order*	*Operand*	*F(Acc)*
100	10 *f* 350	x	x
101	14 *f* 300	a	ax
102	19 *f* 420	ax	ax
103	10 *f* 302	b	b
104	14 *f* 352	y	by
105	12 *f* 420	ax	$ax + by$
106	14 *f* 352	y	$y(ax + by)$
107	19 *f* 430	$y(ax + by)$	$y(ax + by)$.

10 Jump Orders

Any order which can make the machine depart from its normal procedure of obeying orders in the sequence in which they stand in the store is known as a *jump order*. A jump order can be *unconditional* or *conditional*. The effect of an *unconditional jump order* with address part q is to make the machine start to execute a new sequence of orders beginning at half-register q ("jump to q" or "transfer control to q"). A *conditional jump order* causes a jump only if a certain condition (depending on the particular jump order concerned) is satisfied; otherwise, the machine proceeds serially in the normal manner.

An unconditional jump order is:

50 *f q* jump to q; that is, execute next the order in q.

Two of the most common conditional jump orders, in which the criterion for a jump depends on the sign of the number in the accumulator, are as follows:

54 *f q* jump to q if $F(Acc) \geqslant 0$; otherwise proceed serially.
55 *f q* jump to q if $F(Acc) < 0$; otherwise proceed serially.

It will be seen that these two conditional jump orders are complementary. One would be sufficient, but it is much more convenient for the programmer to have both available.

Example 3. If $x = F(200)$ is negative, replace it by 0; otherwise leave it unchanged.

Location	*Order*
100	10 *f* 200
101	54 *f* 103
102	13 *f* 200
103	19 *f* 200

In this small program the sequence of orders performed by the machine is different according as $x \geqslant 0$ or $x < 0$. If $x \geqslant 0$ the number in the accumulator is positive or zero when the 54 order is encountered. This is the condition for a jump, and the machine therefore jumps to the address specified in the 54 order, that is, to 103. The effect is to omit the order in 102. If $x < 0$ the 54 order does not cause a jump and the order in 102 is executed.

11 Use of Parameters as Labels

In order to be able to write the correct address in a jump order the programmer must know the number of the half-register in the store which contains the order to which he wishes the machine to jump. When the first draft of a program is being written, this

information may not be available, since the programmer may not have decided where the various sequences of orders should go in the store. A very convenient facility is provided with EDSAC 2 whereby the order to which a jump is to be made may be given a label, and a reference to this label written in the address part of the jump order instead of the actual address. Labels are written after the orders to which they refer, and consist of a left-hand bracket followed by a number lying between 3 and 127 inclusive. In the address part of the jump order the number in the label appears preceded by the letter *p*, and is referred to as a *parameter*. If a label is used the sequence of orders in Example 3 may be written as follows:

Example 4.

```
10 f 200
54  p99
13 f 200
19 f 200 (99
```

Here the label 99 has been given to the last order, and the corresponding parameter *p*99 appears in the address part of the 54 order. Note that the letter *f* which normally follows the function digits may be omitted before a parameter. When parameters and labels are used, there is no need to write by the side of each order the number of the half-register in the store in which it is placed, and this will not be done in future.

When a program is read into the store of the machine from the program tape, parameters appearing in the address parts of orders are automatically replaced by the correct absolute addresses. The parameters and labels have then served their purpose and they do not correspond to anything remaining in the store of the machine after the reading process is complete. One consequence of using parameters and labels is that the programmer need have no knowledge of the actual registers in the store into which individual orders go; provided that the cross-referencing between the parameters and labels is correct, and provided that no part of the program is overwritten by succeeding parts, the program will operate correctly. Details of the way in which a program tape is made up will be given later, together with information about further uses which may be made of parameters.

12 Cycles of Orders

Many long calculations involve repeated application of the same group of orders to different sets of numbers. Such a repeated group of orders is called a *loop* or *cycle*. The number of times a cycle has to be repeated may be known in advance or may depend on the numbers produced in the course of the calculation. In either case

the cycle must include at least one jump order so that the machine can return from the end of the cycle to the beginning; moreover, it must include at least one conditional jump order since otherwise it will be impossible for the machine ever to leave the cycle.

An example of a short loop in which the number of repetitions is not known in advance, but is controlled by the results of the calculation, is the following:

Example 5. Register 104 contains the number $-x$, and register 102 contains y; x and y are both positive. Add y to $-x$ repeatedly until the result becomes positive (or zero); place the result in 104.

Order	*Operand*	*F(Acc)*
10 *f* 104	$-x$	$-x$
┌→12 *f* 102 (99	y	$-x+y,\ -x+2y,\ -x+3y,\ \ldots,$ successively
└─55 *p*99		$-x+y,\ -x+2y,\ -x+3y,\ \ldots,$ successively
19 *f* 104		final value of $-x+my$.

The 55 order causes a jump back to the previous order if the content of the accumulator is negative; when the cycle has been repeated a sufficient number of times the number standing in the accumulator when the 55 order is encountered will be positive and the machine will proceed serially to the following order which places the required result in 104.

A procedure of this kind could be used for reducing a large negative angle to the range 0 to 2π by successive addition of 2π. Note that the conditional jump for leaving the cycle is in this case also the jump which causes the cycle to be repeated.

The following is a less trivial example of a process in which the number of repetitions of a cycle depends on the results obtained as the calculation proceeds.

Example 6. Given $x = F(200)$,
$a = F(300)$,
where $0 < a < x < 1$, form the sum of the series $x + x^2 + x^3 + \ldots$, up to (but not including) the first term which is less than a, and place the result in 202.

We shall build up the sum term by term in 202, so that when the calculation is complete the result will already be in that register, as required. We shall also need to keep a record of x^n, the term last added to the sum, in order to form x^{n+1}, the next term; let x^n be kept in 204. If we start with x in both 202 and 204, we want the cycle to be such that, at the end of n repetitions, x^{n+1} is in 204 and $x + x^2 + \ldots + x^{n+1}$ is in 202. For brevity we shall write S_n for the sum of n terms, so that

$$S_{n+1} = S_n + x^{n+1}.$$

The orders required are as follows:

	Order	*Operand*	*F(Acc)*
	10 *f* 200	x	x
	19 *f* 202	x	x
	19 *f* 204	x	x
→	10 *f* 204 (99	x^n	x^n
	14 *f* 200	x	x^{n+1}
	19 *f* 204	x^{n+1}	x^{n+1}
	13 *f* 300	a	$x^{n+1} - a$
	55 *p*98		$x^{n+1} - a$
	10 *f* 204	x^{n+1}	x^{n+1}
	12 *f* 202	S_n	$S_n + x^{n+1} = S_{n+1}$
	19 *f* 202	S_{n+1}	S_{n+1}
	50 *p*99		S_{n+1}
	..... (98		

The 55 order is used to test whether the condition for leaving the cycle is fulfilled. This condition is $x^{n+1} - a < 0$ and a jump occurs if it is satisfied. If the condition is not satisfied, the machine proceeds serially and forms S_{n+1} by adding x^{n+1} to S_n. The 50 order then causes an unconditional jump back to the beginning of the cycle.

In this example, the jump order which returns from the end of the cycle to the beginning is not the same as the conditional jump order which tests whether the cycle is to be repeated. The reader will find that this is often the most convenient way to write a program although it may not appear to be the most economical in orders. The first three orders are not performed repeatedly and do not belong to the cycle proper. They are needed to set up the initial values of the quantities used in the cycle. Almost all cycles need a "prologue" of this nature; many of them also need an "epilogue." For instance, if we had required the sum to be in the accumulator at the end of the calculation, we could have placed the order 10 *f* 202 immediately after the 50 order in the above program. In writing a program, it is often convenient to write the orders of the cycle first and then to add any preliminary or final orders that may be required.

13 Cycles of Orders with a Count

In the last example the number of terms of the series we had to take could not be predicted in advance. We might instead wish to take a definite number, say 50, terms of the series. One way of doing this would be to place somewhere in the store a number equal to the number of cycles to be performed, and to subtract one from this number in the course of each cycle. We would arrange to leave

the cycle when the number became zero. While counting in this way, using one of the ordinary registers of the store, would be perfectly possible, counts of this type are so frequently required that the machine is provided with two special registers which can be used for the purpose. They are known as *modifier registers*, for reasons which will appear in the next section, and are distinguished by being referred to as the *s register* and the *t register*. Their contents are integers which are denoted by s and t respectively. The operations involving these registers will be explained in terms of the s register; similar statements with t substituted for s apply to the t register.

Orders which set the value of s are as follows:

70 *s q* place the integer q in the s register.
71 *s q* place the integer $-q$ in the s register.

Note that q here stands for a number and does not refer to the content of storage register q as it does in the orders described earlier.

It is usually convenient (for reasons which will appear in the next section) to count in multiples of 2; for this purpose we require means of increasing or decreasing s by 2, and we also require a conditional jump order which tests the value of s. Both the increasing or decreasing of s and the testing can, in fact, be done by one order:

74 *s q* increase s by 2; if the new value of s is 0 proceed serially, if not jump to q.
75 *s q* decrease s by 2; if the new value of s is 0 proceed serially, if not jump to q.

The definitions here given of these two orders assume that the register s originally contained an even number; more complete definitions are given in Part 2. Counting, with the aid of these orders, is extremely simple. If we have a sequence of orders which we wish to be repeated, say, fifty times, all that we need to do is to place in front of the sequence a 71 order which sets -100 in the s register, and at the end of the sequence a 74 order which causes a conditional jump back to the first order of the sequence; thus:

```
71 s 100
..... (99 ┐
.....     │  orders to be repeated
.....     │  50 times.
          │
.....     ┘
74 s p99
```

Alternatively, we may make use of a decreasing count instead of an increasing count:

```
70 s 100
..... (99 ┐
.....     │  orders to be repeated
.....     │  50 times.
          │
.....     ┘
75 s p99
```

Example 7. Given $x = F(200)$, form the sum

$$x + x^2 + \ldots + x^{50};$$

place the sum in 202.

In this example, it will be convenient to use the following order, which has the effect of clearing a given storage location:

9 *f q* clear *q*; that is to say, set $F(q) = 0$.

Note that this order does not affect the content of the accumulator.

A sequence of orders for performing the required calculation is given below.

Order	*Notes*
71 *s* 100	set count (counting upwards)
9 *f* 202	clear 202
10 *f* 200	put x in accumulator (as initial value of x^n)
50 *p*98	jump into cycle
10 *f* 204 (99 ┐	
14 *f* 200	
19 *f* 204 (98	cycle of orders which adds x^n to partial sum
12 *f* 202	in 202 on nth repetition.
19 *f* 202 ┘	
74 *s p*99	count.

14 Automatic Modification of Orders

It often happens that a group of orders has to be repeated a number of times with slight and systematic changes on each repetition. These changes will normally be in the address parts of some of the orders. For example, suppose we have $x_1, \ldots, x_{40}$ stored in 302, . . ., 380 and $y_1, \ldots, y_{40}$ in 402, . . ., 480, and we wish to form $x_1y_1 + \ldots + x_{40}y_{40}$ in 200. This can be done by repeated application of the cycle of orders given below provided that the address parts of the first two orders can be increased by 2 on each repetition so that, on the nth repetition, they have the values shown. S_n is

written for the sum up to and including x_ny_n, and it is assumed that storage register 200 is clear at the beginning.

Order	*Operand*	*F(Acc)*	
10 *f* 300 + 2n	x_n	x_n	┐
14 *f* 400 + 2n	y_n	x_ny_n	│
12 *f* 200	S_{n-1}	$x_ny_n + S_{n-1} = S_n$	│ (1)
19 *f* 200	S_n	S_n	┘

It will be observed that the increment in the address parts of the first two orders on each repetition is 2; this is the same as the increment in the number in the *s* register if we use that register for counting and count up in steps of two.

EDSAC 2 is provided with a facility whereby an order may be modified (before being obeyed) by having the content of the *s* (or *t*) register added to its address part. In the written form of the order this requirement is indicated by using the letter *s* (or *t*), instead of *f*, to separate the function and address parts of the order. It must be understood that the order in the store is not altered in any way; the modification takes place in the control unit of the machine on each occasion before the order is executed. The following example shows how the order modification facility may be used in the calculation described above:

Example 8. Given $x_1, \ldots, x_{40}$ stored in 302, . . ., 380,
$y_1, \ldots, y_{40}$ stored in 402, . . ., 480,
to form $x_1y_1 + \ldots + x_{40}y_{40}$ in 200.

We already know what the arithmetic orders in the cycle should look like in the control unit before they are executed; these orders are given in (1) above and the cycle can be built up around them as follows:

```
 9 f 200          clear 200 ready to form sum
71 s  80          set count initially at −80
10 s 382 (99 ┐
14 s 482     │    form S_n in 200
12 f 200     │
19 f 200     ┘
74 s p99          count.
```

On the first repetition, when $s = -80$, the addresses in the 10 and 14 orders (as executed) are, respectively, 302 and 402; these addresses are greater by 2 on each repetition, in consequence of *s* being increased by 2 by the 74 order. When constructing a repetitive cycle of this type, it will usually be found that the simplest way to determine the correct addresses to write in *s*- or *t*-modified orders

is to consider what happens when the cycle is gone through the first time.

Note that there are two kinds of order in which s may appear, namely, those which include a reference to s in their specification, and those which do not. In orders of the second kind—of which those with function numbers 9, 10, 11, 12, 13, 14, 15 and 19 are examples—the letter s in the order indicates that the number s is to be added to the address part of the order before it is executed. In those of the first kind—of which orders with function numbers 70, 71, 74 and 75 are examples—the letter s does not denote that the number s is to be added to the address part of the order, but indicates that the modifier register s is concerned in some way in the operation specified by the order. Similar remarks apply in the case of the t register. Orders of the first kind include all arithmetic orders and some jump orders; orders of the second kind are called *modifier orders* and have function numbers from 66 to 90 inclusive.

When s or t is added to the address part of an order the sum is taken modulo 2048; there is no overflow from the address part of the order into other parts, and a meaningful order always results.

Example 9. Given $a_0, \ldots, a_{40}$ stored in 300, . . ., 380
and $x = F(200)$,
to form $S = a_0 + a_1x + \ldots + a_{40}x^{40}$
in the accumulator.

Here, for the first time, we meet an important point of tactics; it is not enough to know what one wants to calculate but one must also decide the best way in which to calculate it. One's first idea might be to build up the series term by term, as we did earlier in Example 6. There is, however, a more economical way which is indicated by writing the quantity to be calculated in the form

$$S = (\{[(a_{40}x + a_{39})x + a_{38}]x + \ldots\}x + a_0).$$

S can be evaluated by a cyclic process in which the arithmetic steps in each cycle consist simply of the multiplication by x of the partial result already obtained and the addition of a coefficient a_j to the result. When we go through the cycle the nth time, the arithmetic orders to be executed must be as follows:

14 *f* 200	multiply by x
12 *f* 380 − 2*n*	add a_{40-n}.

The address of the second order must be decreased by 2 on each repetition and it follows that, if we wish to use a modifier register for modifying the addresses, as well as for counting, we must count

down and not up. We are thus led to the following sequence of orders:

70 *s* 80	set count initially at 80
10 *f* 380	put a_{40} in the accumulator
14 *f* 200 (99	multiply by x
12 *s* 298	and add a_{40-n}
75 *s p*99	count.

15 Special Orders

In addition to orders of the normal type, there are provided in EDSAC 2 certain *special orders* or *permanent subroutines*. From the point of view of the programmer these are to be regarded as single orders, but for engineering reasons they all have the function number 59, and are distinguished by their address parts. The operations they perform are more complicated than those of normal orders. Five of them replace the number x in the accumulator by some function of x, and are as follows:

59 *f* 11	place $x^{\frac{1}{2}}$ in the accumulator
59 *f* 12	place e^x in the accumulator
59 *f* 13	place $\log_e x$ in the accumulator
59 *f* 14	place $\sin x$ in the accumulator
59 *f* 15	place $\cos x$ in the accumulator.

One special order is concerned with the input of numbers:

59 *f* 10	read a number from the input tape and place it in the accumulator.

The form in which numbers have to be punched on the input tape is discussed in Section 20.

Several special orders are concerned with the output of numbers in various forms and with various precisions. A typical one is:

59 *f* 27	punch on the output tape the number in the accumulator, in floating-decimal form with 7 significant figures.

This order leaves the content of the accumulator unchanged.

Quite a wide range of calculations can be programmed very simply with the aid of these special orders.

Example 10. Read a number x from the input tape, calculate $y = e^{-x \sin x}$ and punch x and y on the output tape.

A program for performing this calculation is as follows:

59 *f* 10	read x from input tape into accumulator
19 *f* 2	put x into 2 for temporary storage
59 *f* 27	punch x on output tape

59 *f* 14	form sin *x* in accumulator
14 *f* 2	form *x* sin *x* in accumulator
19 *f* 2 11 *f* 2	change sign of *x* sin *x*
59 *f* 12	form $y = e^{-x \sin x}$ in accumulator
59 *f* 27	punch *y* on output tape.

What comes next in the program depends on what we want to do. We may wish to read a new value of *x* from the tape and compute *y* as before; in this case, the orders given above will be followed by a jump order transferring control back to the beginning. If, however, the one calculation is all that we want to do, we shall terminate the sequence with an order which stops the machine and indicates to the operator that the calculation is ended. This order is as follows:

101 *f* 0 stop the machine and light the stop warning lamp.

16 Punched Form of Orders

So far, we have been concerned solely with the design of a program, that is, with the writing down of the orders on paper. EDSAC 2, however, does not respond to written symbols and, in order to get a program into the machine, the program must first be punched on to paper tape. The tape is of the same type as that used in teleprinter operation and is punched on a *keyboard perforator*, a device with a keyboard resembling that of a typewriter. When a program tape has been punched, it may be passed through a tape reader connected to a teleprinter, and the result will be a printed copy of the program as written.

Depressing one of the keys on the keyboard causes a single row of holes to be punched on the tape. In this row, there are five positions in which holes can be punched, giving 32 possible combinations of holes and blanks. In addition, there is a small sprocket-hole which helps to guide the tape through the tape reader but which has no programming significance.

Most of the keys on the keyboard perforator have two symbols inscribed on them, one belonging to *figure shift* and the other belonging to *letter shift*. One key is marked LETTERS and another key is marked FIGURES (the latter key is also marked ERASE for a reason which will be explained shortly). The character punched on the tape by the depression of a particular key normally has the significance associated with figure shift, but if the LETTERS key is depressed, succeeding symbols have the significance associated with letter shift up to the point, if any, at which FIGURES is punched. CARRIAGE RETURN, LINE FEED and SPACE are common to both shifts. A complete list of the symbols appearing on the keys, together with the corresponding code groups punched on the tape, is given in Appendix A.

Figure shift, in addition to containing the figures from 0 to 9, includes all the small letters and symbols which are used in writing programs (for example, *f*, *p*, *s*, *t*, brackets and the decimal point). It is not therefore necessary to make use of letter shift when punching programs unless it is desired to include titles in the output, in the manner described in Section 18. The figure shift symbol consists of a row of five holes and can be used to cancel or "erase" a symbol punched in error. All that is necessary is to backspace the tape and to press the ERASE (or FIGURES) key. Erase symbols on the tape have no effect, either when a program is being printed on a teleprinter, or when a program tape is being read into the machine (except when printing a title—see Section 18).

A program is punched as it is written, each order being followed by carriage return and line feed symbols. The carriage return symbol is not there merely to help produce the required layout when the program is printed on a teleprinter; for input into EDSAC 2 it is necessary to have some terminating symbol at the end of each order, and the carriage return symbol has been adopted for this purpose. An alternative termination is provided by two consecutive space symbols. Care must therefore be taken not to punch two spaces together except when they are required to indicate the end of an order, but single spaces may be included, if desired, to improve the appearance of the printed page. Similarly, additional line feeds may be included for the same purpose.

17 Tape Stop

It is sometimes desirable to cause the reading of a tape to come to a halt while the operator takes some action. For example, a program tape may be in two parts, and it may be necessary to stop reading while the second tape is being put in the tape reader. The programmer may cause this to happen by punching an asterisk on the input tape at the point at which he wishes reading to stop. The asterisk should follow the carriage return and line feed symbols (or pair of spaces) terminating the previous order. The machine may be restarted after stopping on an asterisk by raising and lowering the RUN key.

18 Titles

It is sometimes helpful, and reduces confusion, if identifying titles are printed on output sheets. This is very easy to do on EDSAC 2 since facilities are provided by means of which titles may be copied directly from the input tape on to the output tape.

A title must be preceded on the input tape by the letter *t*; the *t* will not be copied on to the output tape, but succeeding symbols will be copied until a line feed symbol is reached. If the title happens

to be so long that it takes more than one line of print, a *t* must be punched before each line. Letter shift and figure shift symbols may be included in titles as required. If the last symbol in the title is on letter shift, a figure shift symbol must follow it before the terminating line feed, in order that the succeeding program may be printed correctly on a teleprinter.

19 Directives

A program punched on an input tape must be preceded by a group of symbols which specifies where the first order is to be placed in the store; it must be followed by another group of symbols, which indicates that the program has been completely read in, and which causes the machine to execute the orders, beginning with that in a specified location. Groups of symbols punched on the tape for these and similar purposes are called *directives*.

A directive which causes an order punched after it to go into storage location q is written

$$p1 = q$$

where q is an integer. Succeeding orders will go into storage locations $q + 1$, $q + 2$, etc., unless another directive is encountered.

A directive which causes the machine to stop reading tape and start executing the program, beginning with the order in storage location q, is written

$$s\,q$$

A directive must be terminated in the same way as an order, either by means of a carriage return and line feed or by two spaces.

Example 11. Suppose that the orders given above for Example 10 are required to go into the store, starting in half-register 100; a complete specification for the program tape, including the directive which sends control to the beginning and starts the program, is as follows:

```
p1=100
59f10
19f2
59f27
59f14
14f2
19f2
11f2
59f12
59f27
101f0
s100
*
```


This program has been printed exactly as it would normally be punched. The layout could be made more elegant (and brought into line with that of programs printed elsewhere in this booklet) by punching single spaces before certain of the characters; there is, however, little point in doing this and it makes the program tapes longer. Note that the final directive must be terminated by having a carriage return, or two spaces, punched after it. In order that it may be clear from the printed version of the program that this requirement has not been overlooked, it is good practice to punch an asterisk at the end of the tape, in the manner shown in the above example.

The special directive 25*f*2 must be punched at the beginning of a program tape. This indicates that what follows is machine-code rather than Autocode; it can be preceded by a title, but must come before any directives.

20 Punching of Numbers

Numbers may be read into the machine either during the operation of the program (by a 59*f*10 order) or during the input of the program. Numbers may, in fact, be mingled with orders on the program tape.

Numbers must be punched on the tape in a prescribed style and this style is the same however the numbers are to be read, whether during the operation of the program or during its input. Each number must be terminated by a carriage return, or by two spaces, just like an order.

A number may be punched as a sequence of decimal digits (one to twelve in number), preceded by a sign, and with a decimal point in the appropriate place. If the number is positive the sign may be omitted, and if the number is an integer the decimal point may be omitted. It is also possible for the number to be followed by a power of 10 which is treated as a multiplying factor. In this case, the last digit of the number is followed immediately by the symbol $_{10}$ (which is a single symbol on the keyboard), and then by the exponent. The exponent must be an integer of one or two digits, preceded by a minus sign if it is negative. As with orders, single spaces may occur anywhere in numbers if their presence is thought to improve the layout, but a double space must not be punched except as a terminating character.

Examples of numbers, as they would be punched on the tape, are as follows:

```
12367
−217·42653
1₁₀−13
+17·21439₁₀9
−59432 67890
```

Note that in the third example it is absolutely necessary to punch the 1 before the power of 10.

A number occupies a whole storage register, composed of two half-registers of which the first has an even address. An order, on the other hand, occupies a half-register. This is allowed for automatically when numbers are read from the input tape. If a number follows a group of orders on the tape, the number will be placed in the first available whole register, a blank half-register being left if necessary. Thus, if one had punched on the tape:

$$\begin{aligned} p1 &= 100 \\ 10\,f\,2 & \\ 12\,f\,4 & \\ 50\,p\,3 & \\ 2{\cdot}78926 & \end{aligned}$$

the three orders would go into half-registers 100, 101 and 102, while the number would go into the whole register 104. The half-register 103 would be left unused.

A number intended to be read during the input of a program may be given a label and referred to by means of a parameter in exactly the same way as an order; for example, if the number in the above example were punched as 2·78926 (90, it could be referred to in the address part of an order by means of the parameter $p90$. In the case of numbers intended to be read by a $59\,f\,10$ order, labelling in this way is not appropriate, since a $59\,f\,10$ order places such numbers in the accumulator.

21 Output of Numbers

It has already been mentioned that a number may be punched on the output tape by means of the special order $59\,f\,27$. Numbers thus punched are in floating-decimal form with seven significant figures; that is, when the information on the output tape is printed on a teleprinter, the numbers will appear as in the following examples:

$$\begin{aligned} &2{\cdot}768291_{10}-15 \\ -&3{\cdot}087124_{10}28 \end{aligned}$$

If only five significant figures are required the order $59\,f\,25$ may be used.

If a number is less than 10^4 it may be rounded off to the nearest integer and the result printed by means of the order $59\,f\,24$. Similarly, a number less than 10^6 may be rounded off, and the result printed to the nearest integer, by the order $59\,f\,26$.

If more than a few numbers are to be printed, their layout on the printed page must be programmed. For this purpose there may be inserted, in the part of the program concerned with the

output of results, orders for punching on the output tape symbols for space, carriage return, and line feed. These orders are as follows:

$107\,f\,2$ punch on the output tape the symbol for carriage return.
$107\,f\,8$ punch on the output tape the symbol for line feed.
$107\,f\,30$ punch on the output tape the symbol for space.

Note that when carriage return and line feed are both required, the carriage return *must* precede the line feed, and not vice versa.

Frequently, what is required is a regular layout in which numbers are printed in, say, c columns and divided into blocks each containing, say, b numbers. Facilities are provided whereby such a layout may be preset in advance. Full particulars are given in Part 2 (Section 3.6), but for the present the reader may like to know that such a layout may be preset by the pair of orders

$$\begin{aligned} &46\,f\,q \\ &59\,f\,3 \end{aligned}$$

where $q = 100c + b$. Thereafter the programmer need pay no further attention to layout; he simply writes the output orders necessary to punch the numbers, and the necessary spaces, carriage returns, and line feeds will be inserted automatically.

22 Listing of Constants

When a program is being written it often happens that the occasion arises when it is necessary to use some specific number as the operand of an order. For example, we may wish to multiply the number in the accumulator by 360 to convert revolutions into degrees. This can be done in a very simple manner in EDSAC 2. The programmer merely writes the number itself as the address part of the instruction, using an asterisk (instead of the letter f) to separate it from the function number. Thus, to multiply the number in the accumulator by 360 the programmer would write:

```
14 * 360
```

Similarly, to add 1·414 he would write:

```
12 * 1·414
```

When the program is being read into the machine, the various constants which appear in orders in this way are automatically listed in a part of the store specially set aside, and the addresses of the registers into which they go are inserted into the orders concerned. If the programmer takes no special action, the machine will form the list of constants starting in storage register 2000, but the

programmer may, if he wishes, specify where the list is to start by means of a directive of the form

$$p2 = 1800$$

which would cause the first number in the list to go into storage register 1800.

Note that only constants may be treated in this way. If a number is to be changed during the course of the calculation, then the programmer must arrange for it to go into some known (or labelled) register.

23 Further Orders in the Order Code

This section introduces some further orders in the order code of EDSAC 2.

The following are orders, similar to those with function numbers 10 to 13, but involving the modulus of the content of a specified storage register:

20 *f q* set $|F(q)|$ in the accumulator.
21 *f q* set $-|F(q)|$ in the accumulator.
22 *f q* add $|F(q)|$ to the content of the accumulator.
23 *f q* subtract $|F(q)|$ from the content of the accumulator.

The following are additional jump orders:

51 *f q* jump to *q* and clear the accumulator.
52 *f q* jump to *q* if the content of the accumulator is zero; otherwise proceed serially.
53 *f q* jump to *q* if the content of the accumulator is *not* zero; otherwise proceed serially.

Since the orders with function numbers 10, 11, 20 and 21 provide means of setting the content of any storage register in the accumulator, it is seldom necessary to clear the accumulator; if this has to be done, a 51 order provides one way of doing it.

The following are additional modifier orders:

72 *s q* add the *number q* to *s*.
73 *s q* subtract the *number q* from *s*.
79 *s q* set the address part of the order in location *q* equal to *s*.
80 *s q* set *s* equal to the address part of the order in location *q*.

There are similar orders, containing *t* instead of *s*, which concern the *t* register. Orders with function numbers 79 and 80 are useful for temporary storage and replacement of the content of a modifier register if this register is required for another purpose in the course of a calculation.

It is sometimes necessary to halt the machine temporarily, for

instance to give time for a new tape to be put in the tape reader. For this purpose we have the order:

102 *f* 0 wait until the machine is manually restarted.

24 Explicit Setting of Parameters

We have already seen how the labelling of an order or a number sets the corresponding parameter to a definite value. An alternative way of setting a parameter is by means of a directive of the form

$$p10 = 1000$$

Any parameter up to *p*127 may be set in this way. It will be noted that the above directive is of the same form as those used to set *p*1 and *p*2 and, in fact, *p*1 and *p*2 may be regarded simply as parameters which have special functions in relation to input. The value of *p*1 is increased by unity each time an order is read from the tape, and by two each time a number is read from the tape. During the reading of the tape, therefore, *p*1 continually records the address of the next half-register or register to be filled. The value of *p*2 is increased by two each time a fresh constant is added to the list.

25 Adding of Parameters to Addresses

If the address part of an order consists of an integer followed by a parameter, the value of the parameter will be added to the integer during input to form the address part of the order as it is stored in the machine. For example, the address part of an order punched as

$$19\ f\ 2p21$$

would go into the store with its address part equal to

$$2 + \text{(value of parameter } p21\text{)}.$$

This facility is quite useful, since it enables the programmer to refer to individual members of sequences of numbers without labelling each one, it being sufficient if the first of the sequence is labelled. It also enables the amount of labelling of orders in a program to be reduced, but here the programmer is advised to make use of the facility sparingly or he will find that, when he comes to make corrections or modifications to his program, he has to pay careful attention to the cross-referencing.

A parameter may be used any number of times in the address parts of orders which are read into the machine before it is set.

26 Use of Parameters in Directives

Parameters may be used in directives in the same way as they are used in the address parts of orders. One may, for example, write

$$p21 = p10$$

which would cause the parameter $p21$ to be set to the same value as $p10$. It is necessary that $p10$ shall have been set to a definite value before this directive is reached on the program tape. One can also write

$$p20 = p1$$

which causes $p20$ to be set to the current value of $p1$, that is, to the address of the storage register into which an order immediately following on the tape would go. Conversely, it is permissible to write

$$p1 = p20$$

which would have the effect of causing the following orders punched on the tape to go into the store, starting at the half-register whose address is equal to $p20$.

Modified parameters may be used in directives in the same way that they may be used in the address parts of orders; for example, one may write

$$p21 = 2p20$$

which would set $p21$ to the value (2 + value of parameter $p20$). Similarly,

$$s\ 2p10$$

would cause the reading of the tape to be halted, and control sent to location (2 + value of parameter $p10$).

Parameters may only be used to set other parameters, or in s directives, if they are set at an earlier point on the input tape.

27 Subroutines

A *subroutine* is a self-contained group of orders for performing some calculation, more or less complete in itself, which forms part of a larger calculation. The group of orders required to evaluate a power series (Example 9) could be regarded as a subroutine, although it is perhaps too short for this to be worth while. A more realistic example would be a group of orders for inverting a matrix, or for evaluating some special function required in a particular problem.

A typical program consists of a collection of subroutines together with a *master routine*. At the beginning of the calculation, control is sent to the master routine, which contains the orders required for the general organization of the calculation and for calling into action the various subroutines as they are required. The master routine is thus responsible for organizing the calculation on the highest level, while the subroutines look after the details.

A programmer obtains two advantages by making use of subroutines. In the first place, the breaking up of a long program into

a number of self-contained functional units makes it easier to write, easier to check, and easier to modify if required. Often, subroutines can be tested by themselves before being incorporated into the program. The second advantage arises from the fact that quite a number of calculations of the type that can be performed by a subroutine turn up repeatedly in different problems. It would obviously be a waste of time if each programmer had to devise a set of orders for performing them; this is avoided by having a *library* containing subroutines for the more common operations. Library subroutines are carefully tested before being placed in the library, and the programmer can normally rely on their not containing any mistakes. They are stored on short lengths of punched paper tape which can be copied mechanically on to a program tape.

The subroutines in the EDSAC 2 library are designed in such a way that the programmer may place them wherever he wishes in the store without regard to the location of the master routine. A subroutine is entered, when required, by means of a jump order in the master routine and, when the subroutine has done its work, control is returned to the master routine by another jump order (sometimes called a *link*) in the subroutine. In order to standardize and make automatic the process of entering and leaving a subroutine, two special orders are provided in the order code of EDSAC 2; their function numbers are 58 and 60. Subroutines using these orders for entering and leaving are of the type called *closed*. For many purposes the programmer need not know the exact specifications of the 58 and 60 orders. It is sufficient for him to have the following partial descriptions:

58 f q enter the closed subroutine, of which the first order is in location q.

60 f 0 return from a closed subroutine to the location in the master routine immediately following the 58 order which called in the subroutine. Restore s to the value it had before the 58 order was executed.

Note that, in contrast to everything which has been described earlier, s and t are NOT treated symmetrically by these orders. The content of the s register is always restored to its original value on exit from a subroutine, whereas the content of the t register is not so restored unless steps to that end have been taken by the designer of the subroutine. A more circumstantial account of the action of the 58 and 60 orders is given in the next section.

Although he is not compelled to do so, the programmer will usually find it convenient to write his program in the form of a relatively brief master routine, together with a number of closed subroutines. Some of these subroutines he will have to make himself; for others he will be able to draw on the library.

28 Entering and Leaving a Subroutine

The actions performed by the 58 and 60 orders are as follows:

58 *f q* if *r* is the location of this order, place 0 *f s* in 0 and 0 *f* (*r* + 1) in 1; jump to *q*.

60 *f* 0 set the value of *s* equal to the address part of the order in 0, and jump to the address specified by the address part of the order in 1.

Since register 0 is used by the 58 and 60 orders, it should not normally be used for any other purpose inside a subroutine. Special steps must, however, be taken if it is necessary for one closed subroutine to call in another closed subroutine. One simple procedure is to copy the contents of register 0 into some other register in the store immediately after the first subroutine is entered. Register 0 is then available for use by the second subroutine and can have its contents restored to their original value immediately before the 60 *f* 0 order at the end of the first subroutine is encountered. The first subroutine would then be constructed as follows:

```
10 f 0 ]
19 f 2 ] copy contents of 0 into 2
 . . .
 . . .
58 f q   jump to second subroutine starting at q
 . . .
 . . .
10 f 2 ]
19 f 0 ] reinstate original contents of 0
60 f 0   return to master routine.
```


Part 2

A PROGRAMMERS' GUIDE TO

EDSAC 2

CHAPTER I

GENERAL INFORMATION

1.1 The Free Store

The *free store* of EDSAC 2 consists of 2048 *half-registers* or *storage locations* each with capacity for 20 binary digits. A half-register is capable of holding one order. The half-registers may be combined in pairs to form *registers*; each register has capacity for 40 binary digits and is capable of holding one number.

Each half-register has an *address* in the range 0 to 2047. Two half-registers which are combined to form a register must have consecutive addresses, the lower one being even (for example, 2 and 3, but not 3 and 4). The register may be referred to by either of these addresses, but it is customary to use the lower (even) one.

The digits of register m, starting at the more significant end, will be denoted by $m_0, m_1, \ldots, m_{39}$. The digits $m_0, \ldots, m_{19}$ form the half-register with the lower (even) address; the digits $m_{20}, \ldots, m_{39}$ form the half-register with the higher (odd) address.

The free store is so named because its contents can be altered freely by the programmer. It was the original high-speed store of the machine, and is still the principal working store, used by all programs.

1.2 The Reserved Store

In addition to the free store there is a store of similar size and speed whose contents are available to the programmer but which cannot be directly altered by him. This is called the *reserved store*. Its primary purpose is to contain certain permanent subroutines (see Chapter 3) and some error diagnosis routines (see Chapter 5).

1.3 The Main Store

In January 1962 a further high-speed store was added to EDSAC 2. This is called the *main store*, and consists of 16,384 registers numbered 0–16,383 (counting in ones, since half-registers in this store are not considered as separate entities).

The main store cannot be used without the free store, but the free store (and reserved store) may still be used without the main store. Programming conventions for the main store differ in some details from those for the free store. Full details will be found in the document, *Programming for EDSAC 2 with Main Store*, by D. W. Barron, which was issued in February 1962. Copies of this document are obtainable from the Mathematical Laboratory.

Part 2 of this booklet refers to the programming system and conventions adopted for the free store unless otherwise specified. For completeness, however, orders relevant only to the main store are included in the full order code on pp. 35–42.

1.4 Representation of Numbers

In EDSAC 2 numbers may be represented either in fixed-point form or in floating-point form. In fixed-point form the number in register m, which is denoted by $N(m)$, is defined as

$$N(m) = -m_0 + \sum_{1}^{39} 2^{-i} m_i.$$

It follows that the first digit plays the role of a sign digit and that the binary point comes immediately after it. A fixed-point number x must always lie in the range $-1 \leqslant x < 1$.

In floating-point representation the number in register m, denoted by $F(m)$, is of the form $F(m) = x.2^p$; here p is an integer satisfying $-128 \leqslant p < 128$ and x lies in the range $-1 \leqslant x < 1$. The first 32 digits in register m, interpreted as a fixed-point number, give the value of x; thus

$$x = -m_0 + \sum_{1}^{31} 2^{-i} m_i.$$

The last eight digits in register m, interpreted as a binary integer, give the value of $p + 128$; that is

$$p = \sum_{32}^{39} 2^{39-i} m_i - 128.$$

The floating-point representation of a number is not unique, and one representation is taken as the standard form. The standard form of a non-zero number is that in which $\frac{1}{2} \leqslant x < 1$ or $-1 \leqslant x < -\frac{1}{2}$. The standard form of zero is 0.2^{-128}; it will be observed that the fixed- and floating-point representations of zero

are identical. When arithmetic operations are performed on floating-point numbers in EDSAC 2 the results are always left in standard form.

A non-standard positive number is treated as zero by orders 11–18, 20–23 and 25. A non-standard negative number causes a report (in which the accumulator content is 0.2^{-128}) if used as operand by these orders. Note that this means that an attempt to use the content of a cleared-to-ones register as a floating-point number will lead to a report. See Section 5.2 for a full description of the report facility.

1.5 Representation of Orders

An order consists of 20 binary digits. Of these the first two give the *modifier letters* (see Section 2.1) according to the following table:

Modifier letters (as punched on tape)	*Binary digits*
f or *sr*	00
r or *tr*	01
s	10
t	11

The next seven digits give the function number represented as a positive integer in binary form. These digits, together with the modifier digits, constitute the *function part* of the order. The last eleven digits constitute the *address part*. The notations $f(m)$ and $a(m)$ will be used for the integers given respectively by the function part and the address part of an order; thus

$$0 \leqslant f(m) < 512, \quad 0 \leqslant a(m) < 2048.$$

1.6 The Arithmetic Unit

The arithmetic unit contains three registers, K, L, and M, each of 40 binary digits and each capable of holding a number in the same way as is described above for storage registers. For many fixed-point calculations, registers M and L are combined to form a single 79-digit register which is called the (double-length) *accumulator*; it is often referred to as register A. In combining the registers M and L the sign digit of L is ignored and the remaining 39 digits of L are attached to the less significant end of M. Thus

$$N(A) = -M_0 + \sum_{1}^{39} 2^{-i} M_i + \sum_{1}^{39} 2^{-39-i} L_i.$$

It is convenient also to have a notation for the number that would result if $N(A)$ were rounded off to a single-length number; we therefore write

$$N(A)_R = N(M) + 2^{-39} L_1.$$

If we have $N(M) = 1 - 2^{-39}$ and $L_1 = 1$, this would give $N(A)_R = 1$, which cannot be stored as a fixed-point number; to avoid this inconvenience we write in this anomalous case $N(A)_R = 1 - 2^{-39}$.

Apart from its use as part of the double-length accumulator, L is used to hold the quotient after division with remainder (order 45). Register K is used for accumulative multiplication (orders 16, 17, 24, 36, 37 and 44), and during transfers from magnetic tape.

1.7 The Overflow Flip-Flop

Overflow is said to occur when the correct execution of an order would give rise to a number lying outside the permitted range. In such cases the result actually formed will be incorrect and it is desirable that an alarm should be given. EDSAC 2 is provided with a flip-flop called the *overflow flip-flop* which records whether an otherwise undetected overflow has taken place. The overflow flip-flop is capable of storing a single binary digit denoted by α; $\alpha = 1$ if overflow has taken place and $\alpha = 0$ otherwise. α is set to 1 by overflow resulting from any of the following orders:

6, 7, 11, 12, 13, 14, 15, 16, 17, 18, 20, 22, 23, 25, 31, 32, 33, 34, 35, 36, 37, 38, 40, 42, 43, 45, 96;

α is also set to 1 by a 15 order if the operand is not in standard form. α is set to 0 by any order (except 62) which obliterates the previous content of M, that is by any of the following orders:

10, 11, 20, 21, 26, 30, 31, 40, 41, 46, 51, 100, 110, 120.

(Note that orders 11, 20, 31, and 40 appear in both lists; these orders initially set $\alpha = 0$ but may then cause an overflow and set $\alpha = 1$.) Orders 56 and 57, which enable the programmer to test the value of α, also set $\alpha = 0$. If $\alpha = 1$ an attempt to place a number in the store by one of the orders,

8, 18, 19, 38, 39, 49, 68,

or to use the number as an operand for one of the permanent subroutines

1–4, 11–15, 20–29, 31–33, 41, 42,

will lead to a report stop (see Section 5.2); this ensures that a number cannot be placed in the store, or made use of, if an undetected overflow has occurred in its calculation. The report stop occurs after the order initiating it has been executed, but (in the case of a 59 order) before the permanent subroutine has been entered. If it is desired to place in the store a number in whose calculation an overflow may have taken place the 109 order should be used, unless it is required also to ensure that $\alpha = 0$, in which case the order 56 *r* 1 followed by a 19 order should be used.

1.8 The Modifier Registers

EDSAC 2 contains two *modifier registers* (or B registers). Each modifier register has capacity for 11 binary digits and can hold an integer in the range 0 to 2047. The registers and also the numbers in them are denoted by s and t. Arithmetic operations in the modifier registers take place modulo 2048; there is no indication of, or restriction against, overflow. Similarly, when the content of a modifier register is added to the address part of an order, the addition takes place modulo 2048 and there is no carry into the function part of the order.

Register r, which contains the address of the order currently being executed, or about to be executed, is similar to a modifier register, and it is possible to cause the address part of an order to have r added to it before the order is executed. This facility was provided primarily to enable general purpose library subroutines to be stored on magnetic tape; it is recommended that the ordinary programmer should use it only for local references. Note especially that orders with functions 0–47 or 96–127 which are r-modified must have addresses in the range −256 to +255.

CHAPTER 2

THE ORDER CODE

2.1 Form of Orders

In principle, the written form of an order is

(function number) (modifier letter) (address part);

the function number is an integer between 0 and 127, and the address part an integer between 0 and 2047. It is necessary to distinguish between $a(r)$, the address part of the order in the store (in the half-register whose address is r), and m, the address in the order as actually obeyed. To obtain m we may have to add to $a(r)$ the content of a modifier register, and also a further contribution if the previous order was a 2, 3, 4 or 5 order. For the moment we neglect this last possibility.

Orders are of two kinds. Orders with function numbers between 64 and 95 refer in their definition to register s or t, and are called *modifier orders*. In such orders it is necessary to specify which of s and t is referred to. The significance of the possible modifier letters in a modifier order is as follows:

Modifier letters	*Refer to*	*Obeyed address*
s	s	$a(r)$
t	t	$a(r)$
sr	s	$a(r) + r$
tr	t	$a(r) + r$.

The significance of the modifier letters in other orders is as follows:

Modifier letter	*Obeyed address*
f	$a(r)$
r	$a(r) + r$
s	$a(r) + s$
t	$a(r) + t$.

2.2 Structure of the Order Code

The function numbers have, as far as possible, been arranged in decades for easy memorizing, and functions with the same last digit (in the scale of ten) tend to be similar. In particular:

functions with last digit 0 usually have a 'set positively' effect;
functions with last digit 1 usually have a 'set negatively' effect;

functions with last digit 2 usually involve addition;
functions with last digit 3 usually involve subtraction;
functions which affect both arithmetic or modifier registers and the store have last digit 8 (except 48);
functions which affect only the store have last digit 9.

The function numbers not discussed below will give rise to a report stop (see Section 5.2). Functions 98 and 114 to 119 are used to control the magnetic-tape equipment; the programmer does not need detailed knowledge of their specification.

Unless otherwise stated, orders are obeyed in the sequence in which they are stored; that is, the new value of r will be $r + 1$. The effect of orders on the overflow flip-flop has already been stated in Section 1.7. Register L is cleared by all floating-point orders (except 19 and 24) and by all fixed-point orders which by their nature clear the less significant half of A; that is, by orders

8, 10–18, 20–23, 25, 26, 30, 31, 35, 40, 41, 46, 51, 62, 68, 96, 100, 110, 120.

In the list of orders given below specifications are given opposite the order numbers; on occasion some further comments are given beneath.

2.3 Description of Orders

2 Increase the obeyed address of the next order by m.
3 Decrease the obeyed address of the next order by m.
4 Increase the obeyed address of the next order by $a(m)$.
5 Decrease the obeyed address of the next order by $a(m)$.

These four orders make it possible to use any half-register in the store as a modifier register.

6 Multiply $N(A)$ by 2^m if $m < 1024$, or by 2^{m-2048} if $m \geqslant 1024$.
7 Multiply $N(A)$ by 2^{-m} if $m < 1024$, or by 2^{2048-m} if $m \geqslant 1024$.

If $0 \leqslant m < 1024$, the 6 order causes the content of the accumulator to be shifted left m places, and the 7 order causes the content of the accumulator to be shifted right m places. Such orders would be punched as $6\,f\,m$ and $7\,f\,m$. An order punched as $6\,f\,-m$ will cause a left shift of $-m$ places, i.e. a right shift of m places; a negative address similarly reverses the direction of the shift caused by a 7 order.

8 Exchange: set $N(m)$ in A and $N(A)_R$ in register m. Clear L.

The 8 order can be used for floating-point as well as for fixed-point numbers, since if M contains a floating-point number, L will normally be clear and the fixed-point rounding off will have no effect.

9 Store zero in register m.

This order also works both in floating point and in fixed point, since the zeros are the same in both cases.

10 Set $F(m)$ in M, without change of representation, and clear L.

This order is identical with the 30 order.

11 Set $-F(m)$ in M; clear L.
12 Add $F(m)$ to $F(M)$; clear L.
13 Subtract $F(m)$ from $F(M)$; clear L.
14 Multiply $F(M)$ by $F(m)$; clear L.
15 Divide $F(M)$ by $F(m)$; clear L.

This sets the overflow flip-flop if $F(m)$ is not in standard form.

16 Add $F(K).F(m)$ to $F(M)$; put the old value of $F(M)$ in K and clear L.
17 Subtract $F(K).F(m)$ from $F(M)$; put the old value of $F(M)$ in K and clear L.

$F(K)$ is initially set by a 24 order, but note that it is changed by a 16 or 17 order.

18 Add $F(m)$ to $F(M)$, putting the result in register m as well as in M; clear L.
19 Store $F(M)$ in register m, without change of representation.
20 Set $|F(m)|$ in M; clear L.
21 Set $-|F(m)|$ in M; clear L.
22 Add $|F(m)|$ to $F(M)$; clear L.
23 Subtract $|F(m)|$ from $F(M)$; clear L.
24 Set $F(m)$ in K, without change of representation.

This order is identical with the 44 order.

25 Multiply $F(M)$ by $-F(m)$; clear L.
26 Set the integer m in M as a floating-point number, and clear L ($0 \leqslant m \leqslant 2047$).

29 Store "or": form in m the *logical sum* of the contents of M and m; that is, set $m_i = 0$ if both m_i and M_i were previously 0 and set $m_i = 1$ otherwise.

Information can easily be packed by means of the 29 order. The most convenient way to unpack it again is by means of the 62 order.

30 Set $N(m)$ in A; clear L_0.
31 Set $-N(m)$ in A; clear L_0.
32 Add $N(m)$ to $N(A)$.
33 Subtract $N(m)$ from $N(A)$.
34 Multiply: set $N(A)_R.N(m)$ in A; clear L_0.

If $N(A) \geqslant 1-2^{-40}$, $N(m)$ is placed in A and L_0 is cleared.

35 Divide: set $N(A)/N(m)$, correctly rounded, in M; clear L.

Dividing $+a$ by $-a$ produces the correct result but sets $\alpha = 1$.

36 Add $N(K).N(m)$ to $N(A)$; put the old value of $N(M)$ in K and clear L_0.

37 Subtract $N(K).N(m)$ from $N(A)$; put the old value of $N(M)$ in K and clear L_0.

$N(K)$ is initially set by a 44 order, but note that it is changed by a 36 or 37 order. The 36 and 37 orders accumulate a true double-length product, since the product $N(m).N(K)$ is added to (or subtracted from) $N(M) + 2^{-39}N(L)$, and L_0 is set to zero in the result.

38 Set $N(A) + N(m)$ in A and its rounded value in register m.
39 Store $N(A)_R$ in register m.

It is possible to store $N(M)$ in register m without rounding, irrespective of what may be in L, by a 19 order, since this copies the content of M without change of representation.

40 Set $|N(m)|$ in A; clear L_0.
41 Set $-|N(m)|$ in A; clear L_0.
42 Add $|N(m)|$ to $N(A)$.
43 Subtract $|N(m)|$ from $N(A)$.
44 Set $N(m)$ in K.
45 Exact division: this order is best described if the accumulator is thought of as containing the integer $2^{78}N(A)$ and register m as containing the integer $2^{39}N(m)$. Order 45 divides the integer in the accumulator by that in register m, placing the quotient, as an integer, in L, and the remainder, as an integer, in M. The remainder has the same sign as $N(m)$, zero, however, being taken as having either sign.

Overflow (which will set $\alpha = 1$) will normally cause the value of the remainder to be wrong, as well as that of the quotient, except when dividing $+a$ by $-a$, when the correct result is obtained but with overflow set.

46 Set the value of m in M as a fixed-point integer; clear L.

Here $0 \leqslant m \leqslant 2047$, as with the 26 order.

47 Set $N(m)$ in L (or, what is the same thing, set $F(m)$ in L).
48 Set $N(L)$ in register m.

This will not cause a report stop even if $\alpha = 1$.

49 Store $N(M)$ in register m and $N(L)$ in register $m + 2$.

50 Jump to m; that is, take m as the next value of r.
51 Jump to m and clear M and L.
52 Jump to m if $N(M) = 0$; otherwise continue serially.
53 Jump to m if $N(M) \neq 0$; otherwise continue serially.
54 Jump to m if $N(M) \geqslant 0$; otherwise continue serially.
55 Jump to m if $N(M) < 0$; otherwise continue serially.

Note that these last four orders refer to M, not to A, and that they work equally well for floating-point numbers.

56 Jump to m if $\alpha = 1$; otherwise continue serially.

57 Jump to m if $\alpha = 0$; otherwise continue serially.

In both cases α is reset to 0.

58 Enter closed subroutine: jump to m, set the function parts of the orders in half-registers 0 and 1 to zero, and their address parts to s and $r + 1$ respectively.

This order is used to enter a closed subroutine; in effect it plants in 1 a link which is used by a 60 order. The value of s is preserved, and the s register is therefore available for use within the subroutine. Note that a similar precaution is *not* taken with t. If it is desired to use one closed subroutine within another, the contents of register 0 (i.e. of half-registers 0 and 1) must be removed and stored in safety before the inner subroutine is entered, and restored after it has been left; otherwise the link from the outer subroutine to the master routine will have been lost. Register 0 should not be used as working space in a closed subroutine.

59 Enter permanent subroutine m.

For details see Chapter 3.

60 Leave closed subroutine: set $a(0)$ in s and jump to $a(1) + m$.

This order provides the exit from a closed subroutine, and is complementary to the 58 order. It restores the value of s and returns control to the master routine, skipping m half-registers. (Here m will usually be 0, but it is sometimes convenient to use the m half-registers immediately following the 58 order to contain subsidiary data (*program parameters*) for the closed subroutine.)

62 Place in M the logical product of $N(M)$ and $N(m)$. That is, replace M_i by $M_i.m_i$, $0 \leqslant i \leqslant 39$. Clear L.

This operation is also known as *collate* or *logical "and."*

In the following descriptions of orders 66–90 (*modifier orders*) it is assumed that the order refers to the s register; there are similar orders for the t register. All calculations in the modifier registers, or in the address parts of half-registers, are done modulo 2048; they do not set the overflow flip-flop.

66 Clear L; then cyclicly shift the word in A m places left and set the new value of $2^{39}N(L)$ in s modulo 2048.

A 66 order leaves the overflow flip-flop unchanged.

68 Split: if $F(M) = x.2^p$, set p in s (modulo 2048), set x as a fixed-point number in M and in register m, and clear L.

$F(M)$ is not standardized before this order is obeyed. (It is, in a way, the complement of order 96, though the roles of m in the two orders are quite different.)

69 Set s and $a(m) = d$, where d is the numerical equivalent of the row of holes under the reading head of the selected tape reader, and advance the tape one row.

The tape reader is previously selected by a 106 order. The table of numerical equivalents for 5-track tape is given in Appendix A.

70 Set m in register s.

71 Set $-m$ (i.e., $2048-m$) in register s.

72 Add m to s.

73 Subtract m from s.

74 Repeat, adding: increase s by 2; jump to m if the new value of s is not 0 or 1; otherwise continue serially.

75 Repeat, subtracting: decrease s by 2; jump to m if the new value of s is not 0 or 2047; otherwise continue serially.

76 If $s \geqslant m$ take $r + 2$ as the next value of r (i.e. skip one location); otherwise proceed serially.

77 If $s < m$ take $r + 2$ as the next value of r; otherwise proceed serially.

In the last two orders s is treated as a non-negative integer. The order skipped is often a jump order, or a 2, 3, 4 or 5 order.

78 Add $a(m)$ to s, putting the result in the address part of half-register m as well as in s.

79 Store s as the address part of half-register m.

80 Set $a(m)$ in s.

81 Set $-a(m)$, i.e., $2048-a(m)$, in s.

82 Add $a(m)$ to s.

83 Subtract $a(m)$ from s.

86 If $s \geqslant a(m)$, skip the next order; otherwise proceed serially.

87 If $s < a(m)$, skip the next order; otherwise proceed serially.

88 Exchange $a(m)$ and s.

89 Store s (modulo 512) as the function part of half-register m.

90 Scale: if $N(A) \neq 0$, multiply $N(A)$ by 2^n, where n is as large as possible without overflow, and set $s = -n$. If $N(A) = 0$, jump to m and set $s = -129$.

96 Convert to floating: if $m < 1024$, put $2^m N(A)$ in M as a floating-point number; if $m \geqslant 1024$, put $2^{m-2048}N(A)$ in M as a floating-point number. Clear L.

98 This order is used in magnetic-tape control routines.

99 Store $M_0M_1 \ldots M_{19}$ or $M_{20}M_{21} \ldots M_{39}$ in half-register m, according as m is even or odd.

A 99 order stores the content of the corresponding half of register M in half-register m. The 99 order will not cause a report stop if $\alpha = 1$.

100 Set in the more significant half of M the order in half-register m, and in the less significant half of M the order in the less significant half of register m; clear L.

This enables one to use tables (e.g. for multiple switches) in which each entry occupies a half-register (rather than a complete register) without elaborate unpacking facilities.

101 Stop the machine and light the STOP light on the control panel. Display m on the neons on the control panel.

This order should be used at the end of a program.

102 Light the WAIT light on the control panel and wait until the machine is manually restarted. Display m on the neons on the control panel.

This allows a halt for tape-handling, etc.

105 Set b.†

106 If in the binary representation of m the "1" digit is present, connect tape reader 1 and disconnect tape reader 2; if the "2" digit is present, connect tape reader 2 and disconnect tape reader 1; if the "4" digit is present, connect output channel 1; if the "8" digit is present, disconnect output channel 1; if the "16" digit is present, connect output channel 2; if the "32" digit is present, disconnect output channel 2.

Both output channels can be connected at the same time. A value of m which gives rise to contradictory instructions, such as 3 or 12, will have a random effect; it will not, however, lead to a report stop.

107 Punch character corresponding to m on the output tape.

The teleprinter code for 5-track tape is given in Appendix A.

108 Store b.†

109 Store $N(M)$ in register m, ignoring overflow flip-flop; i.e. independently of α.

110 As order 100, except that m is to be taken as an address in the reserved store.

This makes it possible to read the tables available in the reserved store, and also the data relating to the program stored there. Some of these are listed in Section 4.12. If the address m is even, the effect of the 110 order is to read into M from the whole register m, thus making it possible to read numbers, as well as orders, from the reserved store.

112 Set or increment b.†

114–119 These orders are used in magnetic-tape control routines.

120f1 If OPTIONAL STOP key 1 is depressed, light the OPTIONAL STOP light on the control panel and wait until the key is raised or the RESET button pressed. Set M from the manual register and clear L.

Note that the manual register is not read until the key is raised or the RESET button pressed.

† Full specifications of these orders will be found in the document, *Programming for EDSAC 2 with Main Store.*

120f2
120f4 } If OPTIONAL STOP key 2 or 4 is depressed, light the OPTIONAL STOP light on the control panel and wait until the key is raised or the RESET button pressed. Otherwise proceed immediately to the next order. In either case clear M and L, i.e. set $N(A) = 0$.

A 120 order with address between 1 and 7 will halt the machine if the keys corresponding to the binary ones in the address are depressed, and will read the manual register if the address is odd.

120f16 Set digits in M to give information about peripheral equipments as described below, and clear L.

(a) Line printer

$M_0 = 1$ if buffer can accept characters.

(b) Output channels

$M_{13} = 1$ if channel 1 selected
$M_{14} = 1$ if channel 2 selected
$M_{15} = 1$ if output is busy, i.e. if the output device on either channel, or the camera, is moving.

(c) Input channels

$M_{12} = 1$ if channel 1 selected
$= 0$ if channel 2 selected.

(d) Output devices on channel 1

Punch A	M_2	=1 if selected
Punch B	M_3	=0 if unselected.
Punch C	M_4	
Line printer	M_5	

(e) Output devices on channel 2

Punch A	M_6	=1 if selected
Punch B	M_7	=0 if unselected.
Punch C	M_8	
Line printer	M_9	

The remaining digits of M are used to sense various flip-flops in the machine and are of no interest to the ordinary programmer. They are likely to take different values each time a program is run.

See Chapter 6 for further information on peripheral devices.

121 f 1 Brighten the spot on the cathode-ray tube at a point with x and y co-ordinates given by the digits $M_{10} - M_{19}$ and $M_{30} - M_{39}$, respectively, where M_{10} and M_{30} act as sign digits.

121 *f* 2 Advance the film in the camera of the cathode-ray tube output device by one frame.

2.4 Operation Times

The following table gives the approximate times taken by operations other than those controlling peripheral equipment.

Type of order	*Time in μsec*
Floating point + or −	100–170
Floating point ×	210–340
Floating point ÷	480
Fixed-point + or −	30
Fixed point ×	290–330
Fixed point ÷	350
Shifts	$25 + 5n$ (n = no. of places shifted).
Others	17–42

Permanent subroutines	*Time in msec (approx.)*
11	3·0
12	$3{\cdot}4 + 0{\cdot}4p$ (where $p = \log_2 F(M)$ if this is positive, and 0 otherwise).
13	3·8
14	4·0
15	4·0
31	3·0
32	3·4
33	3·8
34	4·0
35	4·0.

CHAPTER 3

THE PERMANENT SUBROUTINES

3.1 The 59 Order

The actual effect of a 59 order is to bring about a jump into the reserved store in order to perform one of the subroutines permanently stored there. The programmer may, however, regard a 59 order merely as an order whose effects are rather more complicated than those of other orders.

It is convenient to divide the 59 orders into the following groups:

arithmetic operations (m = 11–15, 31–35),
input orders (m = 8, 10, 30),
print orders (m = 20–29, 41, 42),
digit layout control (m = 1, 2),
page layout control (m = 3, 4, 5),
operations connected with differential equations (m = 7, 17, 37),
operations connected with magnetic tape (m = 18, 19, 38, 39),†
matrix division (m = 9),
planting of orders in the reserved store (m = 6).

A 59 order with any value of m not listed here will cause a report stop (except 40, see Section 4.11).

The value of t may be altered by subroutines 10, 17, 30 and 37. The content of the accumulator is altered by subroutines 3, 10–15, 17, 30–35, and 37. If there has been a recent undetected overflow (so that $\alpha = 1$) there will be a report stop on subroutines 1–4, 11–15, 20–29, 31–33, 37, 41, 42, that is, on all those subroutines except 34 and 35 which use the initial content of the accumulator.

3.2 Arithmetic Operations (Subroutines 11–15, 31–35)

The following subroutines are used for calculating certain elementary functions. In the descriptions given, symbols carrying a dash denote values on exit from the subroutine, those without a dash denote values on entry. For floating-point subroutines (11–15) relative errors are given, and for fixed-point subroutines (31–35) absolute errors are given.

† These facilities are fully described in the document *User's Guide to the EDSAC Magnetic-Tape System*, by D. W. Barron. Copies of this document are available from the Mathematical Laboratory.

m	*Description*	*Max. error*
11	$F'(M) = \sqrt{F(M)}$	2^{-30}
12	$F'(M) = \exp F(M)$	$2^{-30}(p+1)$*
13	$F'(M) = \log_e F(M)$	2^{-30}
14	$F'(M) = \sin F(M)$	2^{-30}
15	$F'(M) = \cos F(M)$	2^{-30}
31	$N'(A) = \sqrt{N(A)}$	2^{-38}
32	$N'(M) = \frac{1}{4} \exp N(M)$	2^{-38}
33	$N'(M) = \frac{1}{128} \log_e N(A)$	2^{-41}
34	$N'(M) = \frac{1}{2} \sin \pi N(M)$	2^{-37}
35	$N'(M) = \frac{1}{2} \cos \pi N(M)$	2^{-37}

* Where $p = \log_2 F(M)$ if this is positive, and 0 otherwise.

All these, except 31, clear L; 13 and 33 disturb K. 31 operates on A but the result it produces has only single-length accuracy. However, the *relative* error (error/$\sqrt{A}$) is always $\leqslant 2^{-39}$.

A report stop will occur if the function called for is undefined; this will be the case for subroutine 11 if $F(M) < 0$, for subroutine 13 if $F(M) \leqslant 0$, for subroutine 31 if $N(A) < 0$, and for subroutine 33 if $N(A) \leqslant 0$. An overflow may occur during the operation of subroutine 12 and this will either cause a report stop or set $\alpha = 1$.

3.3 Input Operations (Subroutines 8, 10, 30)

Subroutines 10 or 30 may be used to read a single number from the input tape and to place it in the accumulator. Normally, subroutine 10 is used to read a floating-point number and subroutine 30 to read a fixed-point number; however, if the number on the tape has the prefix *f* or *n* (see Section 4.6) this will determine whether it is to be treated as a floating- or fixed-point number, overriding, if necessary, the choice of subroutine. Certain other prefixes may be used in conjunction with subroutines 10 and 30, and will have the same effect as they have when read by the assembly routine. These prefixes are *t*, *, (, and), and their effects are listed in Section 4.6. In addition, / may be used as a prefix and will cause a jump to the half-register whose address is †*t*; this makes it possible for the machine to read a sequence of numbers of indeterminate length and to pass on to some other action when the last number has been read. If, when subroutine 10 or 30 is called in, the next item (in the sense of Section 4.1) on the tape is not a number, or one of the above prefixes, a report stop will usually occur.

A number read in floating-point form is placed in M and t is left unaltered; thus $F'(M) = x$. If the number is read in fixed-point form and if $|x| < 1$, x is placed in A and t is set equal to 0; on the other hand, if $|x| \geqslant 1$, then $x.2^{-39}$ is placed in A and t is set equal to 39. An overflow will either cause a report stop or set $\alpha = 1$.

† Note that t is disturbed.

Order 59 *f* 8 will read a 5-track tape punched in base-32 form, and check the reading by carrying out a sum check. The tape must be punched in a special form; this is best carried out by using library routine *P*5* or *P*8*. Input of a base-32 tape may also be initiated by pressing the SET BINARY key, in which case tape reader 1 is used and output channel 1 is selected (see Section 6.3). When the base-32 input routine is in use (whether entered by the key or by order 59 *f* 8) and the sum check fails, the REPORT light and the whole column of neons on the operator's control desk will be lit, but no output will take place.

3.4 Output Operations (Subroutines 20–29, 41, 42)

All the subroutines in this category punch the number in M on one of the output tapes.† Before control is returned to the free store the contents of M and L are restored to their original value. The digit layout of numbers punched by subroutines 21 and 22 can be preset to the programmer's requirements with the aid of subroutines 1 and 2 in the manner described in Section 3.5; the digit layout of the numbers printed by the other subroutines is given below, commas representing spaces. In all cases if the number is negative a minus sign is punched; if the number is positive a space is punched instead. Non-significant zeros are suppressed as far as possible; a completely suppressed exponent represents 0, not 1.

m	*Style*	*Width*	*Examples*
20	$M_0M_1 \ldots M_{19}$ as order	9	127*t*2047, ,,0*f*0,,,,
23	Floating binary	21	–·123456, 123456_2–127, ,·000345, $123456_2$7,,,
24	$F(M)$ as integer (4 figs.)	7	,,,–4,, ,1234,,
25	$F(M)$ to 5 figures	12	$-1 \cdot 2345_{10}$–29, ,$1 \cdot 2345_{10}$,,,
26	$F(M)$ as integer (6 figs.)	9	–123456,, ,,,,,,0,,
27	$F(M)$ to 7 figures	14	$-1 \cdot 234567_{10}$–30, ,$0 \cdot 000000_{10}$,,,
28	$F(M)$ as integer (8 figs.)	12	–123, 45678,, ,,,,,,,–47,,
29	$F(M)$ to 9 figures	18	,$1 \cdot 23456, 789_{10}$–12,, $-1 \cdot 98765, 432_{10}$,,,,
41	$N(M)$ to 11 figures	16	–·12345, 123456,, ,·00000, 000000,,
42	$2^{39}N(M)$ as integer	16	–123456, 123456,, ,,,,,,,,,,,,0,,

† Or, if the line printer has been selected, print the number in M directly, within the limitations of the characters available on the printer (see Chapter 6).

Subroutine 23 is not really intended for use by programmers; it was provided primarily for use in report stops. It prints $N(M)$ to 12-figure accuracy, followed by the apparent exponent of $F(M)$; there is no standardization before printing. Subroutines 24, 26, and 28 print the integer nearest to $F(M)$; if this is outside their range, the number is printed in the style of subroutine 25. Note that for fixed-point printing styles it is $N(M)$, not $N(A)_R$ that is printed.

3.5 Digit Layout Control (Subroutines 1, 2)

The digit layout of numbers to be punched by subroutine 21 can be set by entering subroutine 1 with a *digit layout parameter* in M. The digit layout parameter is a fixed-point fraction and is arrived at in the manner described below. The digit layout for subroutine 22 can similarly be set by entering subroutine 2 with the appropriate digit layout parameter in M. It is thus possible for the programmer to preset two different digit layouts for use later in the program when required.

As an alternative, the digit layouts for subroutines 21 and 22 may be set during the reading of the input tape by means of the directives $1/x$ or $2/x$ respectively, where x is the digit layout parameter.

The digit layout constants for subroutines 21 and 22 are kept in the reserved store in registers 118 and 120, respectively.

The method of determining the digit layout parameter will now be described. It has already been explained that the parameter has the form of a fixed-point fraction; this fraction must not have more than 12 digits. If the digit layout parameter is negative, a number punched on the output tape will contain a sign indication; that is, a positive item will be preceded by an extra space and a negative item by a minus sign punched immediately before the first unsuppressed digit. If the parameter is positive no sign indication will be punched.

It is frequently required to suppress non-significant zeros in a number, that is, zeros which have no non-zero digits to the left of them. There are four ways in which non-significant zeros may be treated:

A. Non-significant zeros printed.
B. Non-significant zeros replaced by spaces.
C. Non-significant zeros wholly omitted.
D. Non-significant zeros omitted, but count kept of them with a view to inserting an extra space at the end of the number for each zero omitted.

The digits of the digit layout parameter are interpreted in pairs (except the third digit which is taken by itself). A pair of digits may either set the print subroutine to one of the states A, B, C, or D,

or may leave its state unaltered. When any printing takes place, treatment of non-significant zeros is determined by the state of the print subroutine at the time. If a pair of digits alters the state of the print subroutine, this alteration takes place before any printing which may be called for by that pair of digits.

The digit layout parameter must not be such as to change the style of zero suppression after any digits of the number have been printed, since this may cause a suppression of significant zeros.

The first two digits of the parameter specify how the number is to be expressed in preparation for printing. In the following list of possibilities, d stands for an integer in the range $0 \leqslant d \leqslant 12$. It is understood that if $d = 10$, 11, or 12, there is a carry-over into the first digit of the pair; for example, if $d = 11$, $2d$ will appear as 31. The letters in the second column indicate the state to which the print subroutine is set by the pair of digits concerned.

$0d$	A	$F(M)$ converted to standard decimal form, with the numerical part rounded off to d digits and having one digit before the decimal point.
$2d$	B	$F(M)$ rounded off to the nearest integer and expressed as an integer having d digits, of which some may be non-significant zeros. A report stop will occur unless the integer is numerically less than both 10^d and 2^{39}.
$4d$	A	$N(M)$ to be expressed as a fixed-point fraction rounded off to d digits.
$6d$	B	$2^{39}N(M)$ to be expressed as an integer with d digits, some of which may be non-significant zeros. A report stop will occur unless $2^{39}N(M)$ is numerically less than 10^d.

Although the number is expressed at this stage in a certain way, it does not necessarily follow that it will, in fact, be so printed; later digits in the parameter may specify that more or fewer digits are to be printed, or that the decimal point is to be inserted in some place other than that implied by the form in which the number is originally expressed.

The digits of the number, beginning with the most significant, are printed in groups. The composition of the first group is determined by the third digit of the digit layout parameter; that of subsequent groups is determined by succeeding digits of the parameter, taken in pairs.

The third digit of the parameter, d, where $0 \leqslant d \leqslant 9$, causes the first d digits of the number to be printed, and leaves the state of the print subroutine unchanged.

Succeeding pairs of digits of the digit layout parameter are interpreted according to the following table. In each case a further d digits of the number are printed, where $0 \leqslant d \leqslant 9$. As before, the

letter in the second column gives the state to which the print subroutine is set by the pair of digits in question; if no letter appears, the state is left unaltered.

0*d*	A	print a decimal point followed by *d* digits.
1*d*	A	print *d* digits.
2*d*		print a space followed by *d* digits.
3*d*	B	print *d* digits.
4*d*	C	print *d* digits.
5*d*	D	print *d* digits.
6*d*	D	print subscript $_{10}$ followed by *d* digits of decimal exponent.

A pair of digits of the form 6*d* will only be used when the first two digits of the digit layout parameter are 0*d*. The exponent of the number is expressed as a three-decimal integer; the style in which it is printed may be varied, within limits, by choice of the digit layout parameter, but three digits should always be called for. For example, the pair of digits 63 will cause nothing to be printed if the exponent is zero, whereas the two pairs of digits 6211 will cause 0 to be printed.

The digit layout parameter must be terminated by the group of three digits 9*d*5 where $0 \leqslant d \leqslant 9$. This indicates that no further digits of the number are to be printed, and causes *d* spaces (plus those due to type D zero suppression) up to a maximum of 19 to be printed; if one of the page layout subroutines is being used to control page layout, and if it happens that the number is the last in a line, the final spaces are omitted. If the digit layout parameter contains any further digits these are ignored.

A common use of subroutine 21 or 22 is to enable a floating-point number to be printed without an exponent. Suppose, for example, we wish to print $F(M)$, with five figures before and three after the decimal point, suppressing non-significant zeros in the first four digits. This may be done by multiplying $F(M)$ by 10^3 and printing it with the digit layout parameter set equal to $-\cdot 2841103925$. It will be observed that the form in which the number is expressed when the first two digits of the parameter are interpreted (an 8-digit integer) is not that in which the number is actually printed.

In addition to those given above there are a few other pairs of digits which may be included in a digit layout parameter. These are not likely to be required by the ordinary programmer, but are given here for the sake of completeness.

The following pair of digits may form the first two digits of the parameter:

83 $M_0M_1 \ldots M_{19}$ to be expressed as an order, with the function digits expressed as a three-decimal integer and the address as a four-decimal integer.

The following pairs of digits may be used in the latter part of a parameter:

7*d*	D	print subscript $_2$ followed by *d* digits of the binary exponent.

When this pair of digits is used the parameter will have begun with 4*d*, and the binary exponent will be that explicit in the original $F(M)$. This exponent will, in effect, have been expressed for printing as a three-decimal integer, and remarks similar to those made in the case of the pair of digits 6*d* apply.

8*d*	D	print modifier letter of an order, followed by *d* digits of the address.

This pair of digits will be used when the parameter began with 83; they cause the address to be expressed as a four-decimal integer. As in the case of exponents, there is some choice in the style in which addresses are printed, but four digits must always be called for.

3.6 Page Layout Control (Subroutines 3, 4, 5)

These subroutines are designed to control the layout of numbers on the printed page, thus saving the programmer the trouble of including orders for carriage returns and line feeds.

Subroutine 3 causes the items to be printed in blocks of *b* items arranged in *c* columns. *b* and *c* are parameters whose values may be set by the programmer by entering subroutine 3 with $N(M) = (b + 100c)2^{-39}$. (This is most conveniently done by the order 46*f* *b*+100*c*). The conditions $b < 100$ and $c < 2048$ must be satisfied. When subroutine 3 has been set up in this way the programmer need pay no further attention to layout; items printed by any of the output subroutines (different items may be printed by different subroutines) will be arranged in blocks as described. The first item to be printed will be preceded by a carriage return and two line feeds, so that it begins a new block, and subsequently there will be punched at the end of each line carriage return, line feed, carriage return, or carriage return, line feed, line feed if the line is the last in a block.

An alternative way of setting the layout for subroutine 3 is to include on the input tape the directives 3/*b* and 4/*c* in that order. In this case $b < 2046$ and $c < 2048$. Both directives are needed.

Subroutine 4 provides a more complicated layout, in which a block can be started in the middle. Let $b' - 1$ be the number of items to be printed in the first block, and $c' - 1$ be the number of items to be printed on the first line, and let subsequent blocks be of *b* items in *c* columns. Then the layout is set up by entering subroutine 4 with

$$N(M) = 2^{-19}b' + 2^{-39}b$$
$$N(L) = 2^{-19}c' + 2^{-39}c.$$

The parameters b', b, c', c are represented by the digits of M and L as follows:

$$b' = M_9 \ldots M_{19} \quad c' = L_9 \ldots L_{19}$$
$$b = M_{29} \ldots M_{39} \quad c = L_{29} \ldots L_{39}.$$

Note that these correspond to the address fields in two consecutive registers.

Layout is suppressed when the store is cleared to ones, and can be suppressed by punching the directives $3/-1$, $4/-1$ on the program tape. It can be suppressed during the operation of a program by setting $b' = 2047$, which is most conveniently done by use of subroutine 5.

Subroutine 5 interchanges t and b', so that layout can be suppressed by the orders:

$$70\,t - 1$$
$$59\,f\,5$$

Provided that t is not disturbed in the interim, a subsequent $59\,f\,5$ will restore b' and therefore resume the page layout previously being used. This offers a convenient way of printing a single number on one output channel without disturbing the page layout of the numbers on the other output channel.

If more than one item is to be punched on the second channel, it may be convenient to preserve the layout counts (which are held in the reserved store) and set up a new layout for the second punch. The counts may be preserved in the free store by the orders:

$$110\,f\,122$$
$$19\,f\,m$$
$$110\,f\,124$$
$$19\,f\,m + 2$$

and reinstated, when the original output is to be resumed, by:

$$10\,f\,m$$
$$47\,f\,m + 2$$
$$59\,f\,4$$

If it is desired to punch sequences of numbers on the output tape in such a way that they may be read back into the machine by the assembly routine, or by subroutines 10 or 30, care must be taken that the numbers are terminated by a carriage return or two spaces. Some of the print routines will not do this: subroutines 20, 23, 25 and 27 may provide only a single space after an item, and subroutines 21 and 22 may not provide any spaces. These deficiencies may easily be rectified by use of the 107 order.

3.7 Differential Equations (Subroutines 7, 17, 37)

Subroutine 17 (floating point) enables the integration of a set of n first-order differential equations, $dy_i/dx = f_i(y_1, y_2, \ldots y_n)$, to be advanced one step, using the Runge–Kutta–Gill method†. $3n$ storage registers, beginning at address c, are required for use by the subroutine.

The programmer must write an auxiliary (closed) subroutine for evaluating the derivatives. Given $s = c$ and $y_1, y_2, \ldots$ in $c, c+6, \ldots$, the auxiliary places $hf_1, hf_2, \ldots$ in $c + 2, c + 8, \ldots$, where h is the interval in x, and sets $t = 2n$ on exit. A 58 order calling in the auxiliary must be punched *before* the order $59\,f\,17$ and must itself be preceded by $70\,s\,c$. The registers $c + 4, c + 10, \ldots$ must be cleared before the first step of the integration.

Subroutine 37 resembles subroutine 17 but works with fixed-point numbers. The auxiliary must form $2^p hf_i$, where p is a suitably chosen integer, and must set $N(M) = 2^{-39}\,p$ and $t = 2n$ on exit.

If the auxiliary has some exit in addition to the normal exit by a 60 order, a step of integration may be left incomplete. In these circumstances, subroutines 17 and 37 may be reset to normal by entering subroutine 7.

3.8 Matrix Division (Subroutine 9)

Subroutine 9 forms $\boldsymbol{A}^{-1}\boldsymbol{B}$ and $\boldsymbol{A}^{Q}$ where $\boldsymbol{A}$ is an n by n matrix, $\boldsymbol{B}$ is an n by p matrix, and $\boldsymbol{A}^{Q}$ is a quasi-inverse of $\boldsymbol{A}$. The quotient $\boldsymbol{A}^{-1}\boldsymbol{B}$ replaces $\boldsymbol{B}$, and $\boldsymbol{A}^{Q}$ replaces $\boldsymbol{A}$. It is possible to enter with $\boldsymbol{A}^{Q}$ already in the place of $\boldsymbol{A}$ to form $\boldsymbol{A}^{-1}\boldsymbol{B}$.

On entry:

s = (even) location of first element of $\boldsymbol{A}$ (or $\boldsymbol{A}^{Q}$)
t = (even) location of first element of $\boldsymbol{B}$
$N(M) = 2n$ (set by a 46 order).

The order $59\,f\,9$ is followed immediately by a program parameter $f\,k\,a$ where

$a = 2p$
$k = f$ if $\boldsymbol{A}$ is given
or s if $\boldsymbol{A}^{Q}$ is given

and f indicates the mode of storage of $\boldsymbol{A}$ and $\boldsymbol{B}$ according to the following scheme:

f	$\boldsymbol{A}$ stored by	$\boldsymbol{B}$ stored by
0	rows	columns
1	rows	rows
4	columns	columns
5	columns	rows.

† For details of this method reference may be made to Technical Memorandum No. 62/8: *Integration of Ordinary Differential Equations on EDSAC 2*, by D. W. Barron. This document was issued on 12 April 1962, and copies may be obtained from the Mathematical Laboratory.

If an overflow occurs, control is returned to the order next after the program parameter, with the elements of A and B disturbed; otherwise control is returned to the order next but one after the program parameter. The action of the subroutine changes M, L, K and t.

The division is effected by pivotal operations on rows, with interchange of rows to ensure use of the largest pivot. The time required is $0{\cdot}25n(n+8)(n+3p)$ msec, or $0{\cdot}75np(n+8)$ msec if the quasi-inverse is being used.

The determinant of matrix A may be found by calculating

$$\prod_{r=0}^{n-1}\left(\max_{s=r}^{n-1}(A^{Q}_{r,s})\right)$$

Note that p may be zero, but n must be greater than zero, and that the quasi-inverse should only be used with subroutine 9 or for evaluating the determinant.

3.9 Planting of Orders in the Reserved Store (Subroutine 6)

Order $59f6$ stores $M_0 \ldots M_{19}$ or $M_{20} \ldots M_{39}$ in half-register s of the reserved store, in the manner of a 99 order. A report occurs if $s \geqslant 126$. This facility is provided primarily for use in library subroutines (e.g. $Z8$).

CHAPTER 4

THE PROGRAM ASSEMBLY ROUTINE

The facilities described here are those of the *assembly routine* introduced in February 1962. They differ in some respects from the facilities of the program input routine described in earlier editions of this booklet. This chapter describes the facilities available for free-store programs: the extensions for main-store programs are given in the document *Programming for EDSAC 2 with Main Store*.

4.1 Introduction

The normal method of reading orders and numbers into the store is by means of the assembly routine, which is brought from magnetic tape by the directive $25/s2$, which must be punched at the head of the program tape. Subroutines 10 and 30 use the same conventions, and may therefore be considered with the assembly routine in the following description.

The way in which a program must be punched on an input tape has been designed so that it can be printed out in readable form on a teleprinter; certain layout symbols must therefore be included on the input tape. All symbols used in punching a program are available on figure shift; the letter shift symbol should only be used within a title (see Section 4.6).

Apart from the layout symbols, a program tape consists of four components: *orders*, *numbers*, *directives*, and *prefixes*. The first three, each of which contains several rows of tape, will be referred to as *items*; a prefix obeys rather different rules. Orders and numbers are intended to be read into the store; directives and prefixes control the way in which orders and numbers are so read. Except as controlled by directives, orders and numbers will be stored in a consecutive list in the sequence in which they are read; this may cause a half-register with an odd address to be left unaffected, since a number can only be read into a complete register.

4.2 Parameters

A parametric address facility is available, with 125 parameters numbered from 3 to 127 (inclusive). An address as written and punched consists of a numerical part with any number of parameters added to it or subtracted from it. The numerical part of the address

must be written first and may be preceded by a minus sign; each parameter must be written as the letter p or n (according as it is to be added or subtracted) followed by the number of the parameter; if the numerical part is zero it may be omitted, provided that at least one parameter is present. Examples of addresses containing parameters are:

$$-5p12n17 \qquad p3p3n6 \qquad 57n1n4p96.$$

The address will always be calculated modulo 2048, overflow being ignored.

The parameters $p1$ and $p2$ serve a special purpose. $p1$, which is automatically varied during input, defines the address in the store into which the order or number currently being read (or about to be read) is to be put. The use of $p2$, which is analogous to $p1$, will be explained in Section 4.4. Initially, $p1$ is set equal to 2 and $p2$ to 2000 by operation of the directive $25/s2$.

The remaining parameters are freely available for use by the programmer. They may be set in two ways, either *explicitly* or *implicitly*. A parameter is set explicitly by a directive (see Section 4.5). It can be set implicitly by attaching a *label* to an order or number; when this is read by the assembly routine, the address of the half-register or register into which it is put will be set as the value of the relevant parameter. A label consists of an opening bracket, followed by the number of the parameter; it must precede the carriage return or two spaces which terminate the item.

The value of a parameter may be set explicitly whether or not it previously had a value; it can be set implicitly only if it previously did not have a value. The value of a parameter can be cancelled, without another value being set immediately, by a suitable directive. A parameter whose value, at the current stage of reading in the program, has been set and not subsequently cancelled is called *set*; a parameter whose value has not yet been set, or has not yet been set for a second time after being cancelled, is called *unset*. Initially all parameters except $p1$ and $p2$ are unset; this occurs automatically. Parameters may be unset at a later stage of reading a program by a directive (see Section 4.5).

An unset parameter can be set either explicitly or implicitly. It can also be used as a *forward reference* (that is, it can be used in an order which occurs on the input tape before the value of that parameter has been set).

Set parameters can be used in directives as well as in orders.

4.3 Representation of Numbers

A number consists of a string of not more than 12 digits, possibly preceded by a sign and possibly containing a decimal point. It may be multiplied by a power of 10 and/or of 2; there must not be more

than one of each, with the power of 10 first if both are present. A power is specified by the symbol $_{10}$ or $_2$ followed by an exponent. This is an integer with no decimal point, preceded by a minus sign if negative. An exponent of 10 may not exceed two digits; an exponent of 2 may not exceed three digits.

Examples of numbers are

$$1234 \qquad +\cdot 1234_{10}-8 \qquad -1\cdot 4_{10}6_2-20$$

Regardless of how a number is written, or punched on the tape, it may be stored in the machine in either floating- or fixed-point form. This will be determined by the prefix f, n, or nn if present (see Section 4.6); otherwise the representation will be the same as that of the last number read in, whether by the assembly routine or by subroutines 10 or 30.† Initially, the machine is set to read numbers in floating-point form (by operation of the directive $25/s2$).

If a number is read in fixed-point form, it will be converted to binary form as it stands if it is less than 1 in absolute value; otherwise it will be multiplied by 2^{-39} during conversion. Note that -1 must therefore be punched as -1_239.

A number is normally placed in the next available register, leaving a half-register unused if necessary. A fixed-point integer preceded by $p+$ or $p-$ is placed in the next available half-register, being negated in the case of $p-$. It is necessary for such an integer to be less than $2^{19}-1$ in modulus.

4.4 Representation of Orders

An order is punched in the same way as it is written. It starts with the function number, which is a decimal integer in the range 0 to 127, with no sign; the function number must not be omitted, even if it is zero. It is followed by the modifier letter or letters (f, r, s, t, sr, tr), and then the address, written according to the conventions of Section 4.2. If the address contains no numerical part but at least one parameter, and the modifier letter is f, this letter may be omitted.

If an order refers to a number whose value is known, there is another form in which it can be written. Instead of giving the address of the number in the usual way, it is possible to write the number itself instead of the address; the assembly routine will then put the number in a list in another part of the store, and put the correct address into the order. For this purpose the order must be punched as function number, followed by an asterisk, followed by the number punched according to the usual rules (and preceded

† The prefix nn is only meaningful when read by the assembly routine, not by subroutines 10 and 30.

by *f*, *n* or *nn* if necessary). *p*2 is used to specify the register into which the next constant is to be put, and is automatically stepped on by 2 every time this facility is used.

4.5 Directives

The directives available are as follows:

pm = *a* Set parameter *m* to the value *a*, where *a* is an address, written according to the rules of Section 4.2 but with no forward reference allowed.

In particular, the directive *p*1 = ... determines where the orders or numbers next read are to be put in the store, and the directive *p*2 = ... determines where the constants written in place of addresses in orders (see Section 4.4) are to be put.

=*m* Unset the parameters from *m* to 127 inclusive, to enable them to be used for forward references or to be set implicitly. *m* is an integer from 3 to 127.

=*m*/*n* Unset the parameters from *m* to *n* inclusive.

In general the higher-numbered parameters should be used for cross-reference within subroutines (and re-used when necessary) while the lower-numbered ones should be used for references from one part of the program to another (and so kept fixed).

s m Start program at location *m*, having first checked that all parameters used have had their values set. Here *m* is an address containing no forward reference.

sr m Start program at *m* without making the above check. This directive is useful for interludes. (See Section 4.11)

1/*x* to 4/*x* Set digit or page layout parameters. For details, see Sections 3.5 and 3.6.

The digit or page layout constant is treated as a fixed-point number whatever the type of number conversion currently selected.

4.6 Prefixes

A prefix has an effect similar to that of a directive. It may be punched on its own or before an item, but not within an item or between an item and its terminating symbols. The possible prefixes are as follows:

* Wait until the machine is manually restarted.

This is useful to give time for tape-handling, etc.

f Read numbers in floating-point form.

n Read numbers in fixed-point form.

nn Read numbers in fixed-point form, scaled by 2^{20}.

This is useful for setting main-store addresses.

(Change to tape reader 1.

) Change to tape reader 2.

r If *p*1 is odd, place the dummy order 2*f*0 in the next half-register.

This enables one to be sure of reading the next order into an even half-register. The dummy order will have no effect unless the last order was a 76, 77, 86 or 87 order.

t Punch all succeeding symbols, up to and including the next line feed, on to the output tape, keeping no record of them inside the machine.

This facility is used to put *titles* on the output tape; if a title of several lines is wanted, each line must have its own prefix. If the title is in letter shift, care must be taken to include a figure shift at the end, before the line feed.

pt Begin optional section.

pn End optional section.

If key 0 on the control panel is up, everything between *pt* and *pn* is ignored. If key 0 is down, characters following *pt* up to and including the first line feed are copied to the output tape, then the program following is read in the normal way. The *pn* must follow immediately after the last carriage return, line feed of the optional section. A *pt* not paired with a *pn*, or vice versa, causes a report (see Section 5.3).

pp See Section 4.9.

These prefixes may all be used with the assembly routine. All except *r*, *pt*, *pn* and *pp* may be used with subroutines 10 and 30. A prefix which may be used with subroutines 10 and 30 but not with the assembly routine is:

/ Cease reading numbers and jump to the free-store register whose address was in *t* when the 59*f*10 or 59*f*30 order was obeyed. (See Section 3.3)

4.7 General Punching Rules

Within titles, any symbol may be punched and will be copied on to the output tape. Outside a title a figure shift (erase symbol)

will always be ignored, and a letter shift symbol must never be punched.

Each item must be terminated by two spaces or a carriage return; these are to be regarded as part of the item. Single spaces in an item will be ignored; all other symbols will lead to an input report (see Section 5.3). Outside an item, line feeds, carriage returns, spaces and blank tape will all be ignored.

Prefixes must always be outside an item.

It will be seen that an erase can be punched over any symbol, and that an asterisk or bracket can be punched over a line feed; this simplifies the use of correction tapes.

4.8 Transfer of Subroutines from Magnetic Tape

The directive *q/* causes library subroutine number *q* to be transferred from magnetic tape to the free store, starting at the address given by the current value of *p*1. This must be even: if there is any doubt the directive *rq/* should be used; this will ensure that *p*1 is even before the transfer starts. *p*1 is automatically incremented when the transfer is successfully completed. A list of library subroutines stored on magnetic tape, together with their code numbers, is given in the document *Users' Guide to the EDSAC Magnetic Tape System.*

4.9 Masks and Tables

It is sometimes required to read in an item of the form

$$a.2^x + b.2^y + c.2^z + \ldots$$

for example when setting up masks or logic tables. This is facilitated by the prefix *pp*. The effect of this is to add the number following the *pp* to the previous item. The addition is done in fixed point, ignoring overflow; a report occurs if *p*1 is odd, or if the *pp* is followed by an order. Thus to set the quantity

$$15_2{}^{20} + 128_2{}^{11} + 7_2{}^5 + 1$$

in register 100 we would write

$$p1 = 100$$
$$15_2 20 \quad pp128_2 11 \quad pp7_2 5 \quad pp1$$

The number following *pp* is restricted to be an integer, possibly with a binary exponent, and is normally terminated by a double space (though carriage return, line feed is allowed). Fixed-point conversion must be selected by an *n* elsewhere on the tape. Note that the item preceding the first *pp* must be correctly terminated by a double space.

4.10 Use of Store by the Assembly Routine

The assembly routine uses registers 0, 1, 2 and 15,104 to 16,383 in the main store. It disturbs the contents of registers 0 to 38 and 2046 in the free store whenever it is called from magnetic tape. (New information can be placed in these registers, but any information previously there is destroyed.) Information cannot be placed in registers 0 or 2046 in the free store.

If the directive 25/*s*2 is read by use of the SET START AND CLEAR key, the free store, and registers 3 to 15,103 of the main store, are cleared to ones, and page layout is unset.

Parameter *m*, if set, is stored as a fixed-point integer in register 16,128 + *m* of the main store.

4.11 Interludes

An interlude is a small section of program which is obeyed during the input of the main program. The interlude should be started by the directive *sr m* (see Section 4.5), and at the end of the interlude control should be returned to the assembly routine by the orders:

```
70 s 2
59 f 18
 1 r 25
50 f 2
```

These orders may be anywhere in the free store from location 40 upwards. They cause the assembly routine to be re-entered without clearing the store, with parameters 1 and 2 set to the values they had when the interlude was started. Registers 0 to 38 and 2046 of the free store are disturbed. The interlude must not disturb registers 16,128 to 16,383 of the main store. (The above sequence of orders takes the place of 59*f*40, which was used to terminate interludes in the program input routine described in earlier editions of this booklet.)

4.12 Constants in the Reserved Store

The following constants are available in the reserved store and can be read by a 110 order.

N(1000) to *N*(1022):	
	N(1000 + 2t) = $10^{t+1} \cdot 2^{-40}$ $t = 0, 1, \ldots 11$
N(1422)	26.2^{-8}: approximately 1/10
N(1424)	1000/1024
N(1426)	10/16
N(1428)	$2^{-19} - 2^{-39}$ i.e. 0*f*0, 127 *t* 2047
N(1430)	−1
N(1432)	$100 \cdot 2^{-39}$

$N(1434)$	$1/2$
$N(1436)$	$1/\sqrt{2}$
$N(1438)$	$1/2\sqrt{2}$
$F(1440)$	-3
$N(1442)$	$1/10$
$F(1488)$	$1/6$
$F(1490)$	$1/2$
$N(1496)$	$1/12$
$N(1498)$	$1/4$
$N(1532)$	$\log_e 2$
$F(1534)$	$\pi/2$.

CHAPTER 5

DETECTION OF ERRORS IN PROGRAMS

5.1 Introduction

EDSAC 2 provides two built-in facilities specially for the detection of errors in programs. These are the *report stop* and the *trace*. In addition, the assembly routine incorporates extensive checking, and there are available a number of service routines designed to facilitate the detection of program errors.

5.2 The Report Stop

Whenever an attempt is made to place in the store a number in whose calculation an undetected overflow has occurred, or to obey an order whose function number has no meaning, or to use a non-standard negative number as the operand in a floating-point order, or to read by subroutines 10 or 30 an improperly punched tape, a report stop will occur. The effect of this is to punch five or six items on an output tape, after which the machine stops on a 101 order in the reserved store.

The items printed are as follows:

(1) The content of M, in the style of subroutine 23; that is, $N(M)$ as a fraction to 12 digits, followed by the binary exponent of $F(M)$. If the report stop has been caused by overflow, the content of M is that after the order which caused the report stop has been obeyed, and is usually meaningless.
(2) An asterisk if $\alpha = 1$ at the order which caused the report stop.
(3) The order which caused the report stop.
(4) The address of this order in the store; that is, the current value of r.
(5) The current value of s.
(6) The current value of t.

If the order printed is a 59 order, it indicates that the conditions for use of the subroutine have not been satisfied; the value of s printed will be the value it had when the subroutine was entered (but the value of t is meaningless).

A report stop caused by an improperly punched input tape will give little useful information. Item (3) in the above list will be 59 *f* 10 or 59 *f* 30, and the cause of the report stop will usually be

obvious on looking at the last few rows of the tape that have been read into the machine.

A report can also occur during reading of numbers by subroutines 10 or 30 if the number exceeds capacity.

5.3 Program Input Reports

When the assembly routine encounters anything inadmissible on a program tape, the machine prints *i*/*j*, where *i* is a number which identifies the fault and *j* is the current value of *p*1, and then halts with the OPTIONAL STOP lamp on. If the RESET button is pressed the remainder of the offending item is ignored, and the reading of the program is resumed until the start directive is reached, when the machine stops with the REPORT lamp lit. If the fault is that of unsetting a parameter which has been used as a forward reference but not set, the machine prints *Pc*/*i*/*j* where *c* is the parameter number and *i*, *j* have the same meaning as before. If a report occurs in a start directive, a list of any parameters used but not set is printed when the RESET button is pressed; the machine then stops with the REPORT lamp on.

Since a report causes the remainder of the item to be ignored, an error may cause bogus input reports elsewhere. For example, an error in a parameter-setting directive will mean that the parameter is subsequently regarded as unset.

A list of possible errors, together with the corresponding identifying numbers, is given as an Appendix to the document, *Programming for EDSAC 2 with Main Store.*

5.4 Trace

During any period of time in which the TRACE key is held down, a list is kept of jump orders obeyed during the running of the program. This list is kept in the reserved store. A single repeated cycle occupies one entry in the list, and a double repeated cycle occupies two entries; at any time the list contains the 41 most recent entries. This list can be analysed, and printed as a record of jumps and cycles by service routine *PM*25*; alternatively, *PM*24* can be used to print details of the last two jump orders obeyed. The trace facility reduces the speed of the machine by a factor of about 2. If the TRACE key is down, the SET START AND CLEAR and SET START keys are inoperative.

5.5 Service Routines

Details of the service routines which are available to assist in the detection of errors in programs are given in a document called *EDSAC 2 Service Routines,* which is available from the Mathematical Laboratory.

5.6 Comparison Post Mortem

The original program input routine included a comparison post mortem facility, the purpose of which was to compare the program actually in the machine with the program originally read in, and to print out any discrepancies. A full description of this facility was given in previous editions of this booklet. It should be noted that the assembly routine now in general use does NOT include any comparison post mortem.

CHAPTER 6

PERIPHERAL EQUIPMENT

6.1 Equipment Available

Communication between the computer and the external world is via its *input* and *output channels*, to which may be connected various electro-mechanical and electronic devices collectively known as *peripheral equipment*.

This equipment comprises:

2 input channels, to each of which is connected one paper-tape reader capable of reading tape at speeds of up to 1000 characters (i.e. rows of holes) per second.

2 output channels, to which any of the following equipment may be switched:

Punch *A*—slow (30 ch/sec) paper-tape punch (5-track).
Punch *B*—fast (300 ch/sec) paper-tape punch (5-, 7- or 8-track).
Punch *C*—fast (300 ch/sec) paper-tape punch (5-track).
Line printer (75 lines per minute).

1 special output channel to which is permanently connected a cathode-ray tube output device, with camera.

In addition to the above input-output devices the term 'peripheral equipment' is also taken to include the magnetic-tape auxiliary store. This consists of 4 magnetic-tape units designated *A*, *B*, *C* and *D*. These communicate with the computer via 4 magnetic-tape channels known as 1, 2, 3 and 4. Any tape unit can be connected to any channel.

No further details of the magnetic-tape equipment will be given in this booklet since a full description is available in *Users' Guide to the EDSAC Magnetic Tape System*, by D. W. Barron. Copies of this document are available from the Mathematical Laboratory.

6.2 Input

Input of programs and of most data is from 5-track paper tape. The EDSAC teleprinter code for this is given in Appendix A. The assembly routine and permanent subroutines 10 and 30 (see p. 44) are designed to read information punched in this code. Subroutine 8 will read information punched in base-32 (see p. 45).

Special subroutines have to be provided to read information punched in any other 5-track form.

Some of the available tape readers (notably the green-coloured ones, made by Elliott Brothers) may be used to read 5-, 7-, or 8-track tape merely by pulling out the loading bar on the front of the reader to the appropriate position. There is no standard EDSAC 7-track code; one Flexowriter is available in the Mathematical Laboratory for preparing 7-track tape, but its use is normally restricted to special projects which cannot conveniently be handled on 5-track tape.

6.3 Output Equipment Selector

A set of four, 3-position keys is fitted to the top right-hand side of the control panel. These keys control the selection of slow punch *A*, high-speed punches *B* and *C*, and the line printer. If any key is in its centre position, its associated output device will not be connected to either of the output channels. If a key is placed in the L.H. position, its associated device will be connected to output channel 1 (as specified by a 106 order); if it is in the R.H. position, the device will be connected to output channel 2. Note that any number of the available output devices can be connected, if necessary, to the same channel.

A neon is mounted to the right of each key, and, when lit, indicates that the associated device is busy. When the OUTPUT BUSY light comes on, reference to these neons will enable the output device causing the stoppage to be identified. In particular, if an attempt is made to select an output device that is unplugged from the machine, the appropriate neon and the OUTPUT BUSY light will both come on, and the machine will be held in the "output busy" condition until the key is returned to its centre position.

Note that the machine will continue to operate even if one (or both) of its output channels is unconnected to a punch or printer; information put out on such a "dead" channel is, of course, lost.

6.4 Line Printer

The line printer is a modified Hollerith tabulator printer, capable of printing 80 characters on one line. The characters that can be printed are restricted to the decimal digits 0–9, decimal point, and minus sign; these are treated exactly the same in figure shift and in letter shift. All other characters normally printed by a teleprinter will appear as spaces, except asterisk, which is printed as a decimal point; the symbols for figure shift, letter shift and blank tape are ignored. A number such as $1{\cdot}23456_{10}-7$ is printed as 1·23456 –7.

The output of suffix $_{10}$ can be avoided by the use of library subroutine *P*4. It should be noted that, owing to the method of construction of the printer, artificial spaces always appear after the 20th and after the 40th columns (counting from the left).

The printer has associated with it a buffer store which can hold up to 80 characters. This buffer can be filled very rapidly by the machine by means of normal output orders (standard reserved-store print routines can be used for this purpose). When the buffer contains all the characters it is intended to print on one line, printing can be initiated by the output of a carriage return symbol: as with a teleprinter, all line feeds must be separately programmed. If the buffer is empty, the carriage return symbol is ignored, and nothing is printed. If an attempt is made to load 81 characters into the buffer without a carriage return, the printer will automatically carry out a line feed and then immediately print the 80 characters in the buffer, leaving the 81st character in the buffer for subsequent printing.

The print cycle takes approximately half a second, but the buffer becomes free, and may be reloaded, after about two-thirds of this period. Programs with a large amount of output can be arranged to overlap calculation and printing: this is most conveniently done by library subroutine *P*6, which uses the interrupt facility (see Section 6.5), or use can be made of order 120 *f* 16 (see page 41).

During the print cycle, before the buffer has become free for reloading, the printer can accept any of the following groups of characters without taking any further time:

(*a*) two line feeds,
(*b*) a single character to be printed,
(*c*) a line feed followed by a single character to be printed.

Any of these groups will cause the printer to become busy; this means that an attempt to send further characters to the printer, or to use any other output device (including the cathode-ray tube) will hold up the machine until the print cycle is finished. Note, however, that the machine can go on *computing* whilst the line printer is busy. The printer may also go busy due to certain fault conditions; these will usually cause the red NOT READY light on the printer to be lit.

There is a push button labelled PRINT on the front of the printer. When pressed this causes a line feed and then the printing of the contents of the buffer (if any). This button must not be pressed while any of the magnetic tapes are in use, since if it is pressed there is a danger that information on magnetic tape may be destroyed. The key labelled PAPER FEED on the machine control panel, when depressed, causes the paper in the printer to be advanced.

6.5 Interrupt Facility

This facility provides a means of breaking into a program to take special action when a button on the control panel is pressed, or when the line printer becomes free to accept more characters.

The programming involved in utilising this facility is elaborate, and the interrupt will generally be used in conjunction with special library subroutines (for example *P*6, *Z*10). However, the facility is described here for those users who wish to make direct use of it.

The source of an interrupt is selected by a 4-position switch on the control panel: the positions of this switch are OFF, MANUAL, PRINTER, PUNCH. A manual interrupt is signalled by pressing the button next to the interrupt switch. The neon above the button comes on, and will stay on until the RUN key is lifted. Whilst the neon is on the INTERRUPT switch should not be moved: the button, however, is inoperative. The printer interrupt is signalled whenever the buffer store becomes free to accept characters; if the switch is moved to PRINTER when the printer is not in use, there will be an immediate interrupt. The switch position labelled PUNCH is intended for use in conjunction with an automatic error check on high-speed punch B, which is in course of being provided.

The actual interruption takes place after the first jump order in the free store following the signalling of an interrupt condition; thus interrupt is ineffective in the reserved store. The effect is to transfer control to location 2041 in the free store, recording a link in the reserved store ($a(0) = s$, $a(1) = r$), and setting the most significant two digits of s to be:

01 if the interrupt was manual,
10 if the interrupt was from the printer,
11 if the interrupt was from the punch.

(The remaining digits of s are not all zeros, and may not be the same on different occasions that the program is run.)

Care must be taken to avoid the situation in which the program which deals with an interrupt is itself interrupted. If there is a possibility of this occurring, the first action of the interrupt program should be to preserve the link. Similarly, if the interrupt program uses a 59 order the link must first be preserved. The trace facility (see Section 5.4) is incompatible with the interrupt facility: if an interrupt occurs when the TRACE key is down the machine will enter a closed loop in the reserved store.

6.6 Cathode-Ray Tube Output

This device has two cathode-ray tubes mounted side-by-side. The left-hand tube has a camera associated with it, while the right-hand tube, which has a long-persistence screen (about 3 seconds), is more useful for visual observation.

The display on both tubes is controlled by the order 121 *f* 1 (see p. 41) which causes a spot on each tube to be brightened for approximately 30 μsec. The matrix of possible spot centres is 1024 by 1024, but each spot spreads over an area of about 4 by 4 matrix points, so that the line resolution is only about 1 in 256.

The camera on the left-hand tube works in association with order 121 *f* 2, which advances the film by one frame. To use the camera the on/off button on the top of the camera must be pressed down and turned clockwise to lock it in the "on" position. The L.H. neon above the camera tube will then light to indicate that the camera is on. Note that this button opens the camera shutter. It is preferable, therefore, only to turn the camera on at the beginning of a run in which it is to be used, and to turn it off again at the end of the run.

The approximate number of frames of film unused is indicated by an engraved disc on the camera. When the film is exhausted, the R.H. neon of the camera tube lights, as does the OUTPUT BUSY lamp on the control panel. To remove the "output busy" condition, and to allow the machine to be used whilst the camera is being unloaded, the camera must be switched off.

If an attempt is made to advance the film by means of order 121 *f* 2 and an "output busy" indication is given, the most likely cause is either that the camera is not switched on or that the film is exhausted.

Note that if the camera is switched on when the film is exhausted an immediate "output busy" indication will be given, even if an order 121 *f* 2 has not been used.

The order 121 *f* 2 takes about 30 μsec, but it inhibits any further output order, or positioning of magnetic tape, for about $\frac{1}{2}$ sec.

For normal operation it is recommended that the film should be advanced by 121 *f* 2 *before* displaying data by means of 121 *f* 1, and also advanced one frame at the end of a run.

APPENDIX A

EDSAC 2 TELEPRINTER CODE

Code	*Numerical Equivalent*	*Character* *Figure*	*Letter*
00·000	0	No effect	
00·001	1	*f*	F
00·010	2	Carriage return	
00·011	3	0	O
00·100	4	*r*	R
00·101	5	7	K
00·110	6	2	U
00·111	7	*s*	S
01·000	8	Line feed	
01·001	9	8	L
01·010	10	5	H
01·011	11	/	A
01·100	12	9	M
01·101	13	(	Z
01·110	14	=	E
01·111	15	$_{10}$	D
10·000	16	*p*	P
10·001	17	4	G
10·010	18	3	Y
10·011	19	$_{2}$	W
10·100	20	6	J
10·101	21	*t*	T
10·110	22	*n*	N
10·111	23	–	B
11·000	24	1	I
11·001	25	*	C
11·010	26	)	X
11·011	27	Letter shift	
11·100	28	·	V
11·101	29	+	Q
11·110	30	Space	
11·111	31	Figure shift	

Figure	*Numerical Equivalent*
0	3
1	24
2	6
3	18
4	17
5	10
6	20
7	5
8	9
9	12
+	29
–	23
·	28
$_{10}$	15
$_{2}$	19
f	1
n	22
p	16
r	4
s	7
t	21
=	14
*	25
(	13
)	26
/	11

Letter	*Numerical Equivalent*
A	11
B	23
C	25
D	15
E	14
F	1
G	17
H	10
I	24
J	20
K	5
L	9
M	12
N	22
O	3
P	16
Q	29
R	4
S	7
T	21
U	6
V	28
W	19
X	26
Y	18
Z	13

Action	*Numeral Equivalent*
Figure shift	31
Letter shift	27
Space	30
Carriage return	2
Line feed	8

INDEX