WILKES • WHEELER • GILL

PROGRAMS FOR AN ELECTRONIC DIGITAL COMPUTER

ADDISON-WESLEY PUBLISHING COMPANY, INC.

PROGRAMS FOR AN ELECTRONIC DIGITAL COMPUTER

Second Edition

By M. V. Wilkes, D. J. Wheeler,
and
Stanley Gill

Thoroughly revised and expanded to include machines other than the Electronic Delay Storage Automatic Computer (EDSAC), this Second Edition offers a general introduction to programming for any computer of the stored-program type. It is designed for those using electronic digital computers, for those putting new machines into operation, and for those wishing to assess the possible application of such computers to their own problems.

Programming cannot be taught in the abstract nor can it be learned without practice, and any book on programming must use the order code of some particular machine, real or hypothetical. Fortunately, the order code of the EDSAC lends itself well to the purposes of a book on programming, being fairly straightforward and relatively easily memorized. Moreover, the EDSAC is a single-address binary machine, and a recent survey shows that, of the types of machines currently in use, about 50% use single-address order codes and about 60% work in the binary scale. Another advantage of using the order code of an established machine is that it is possible to draw on experience of programming and of the teaching of programming.

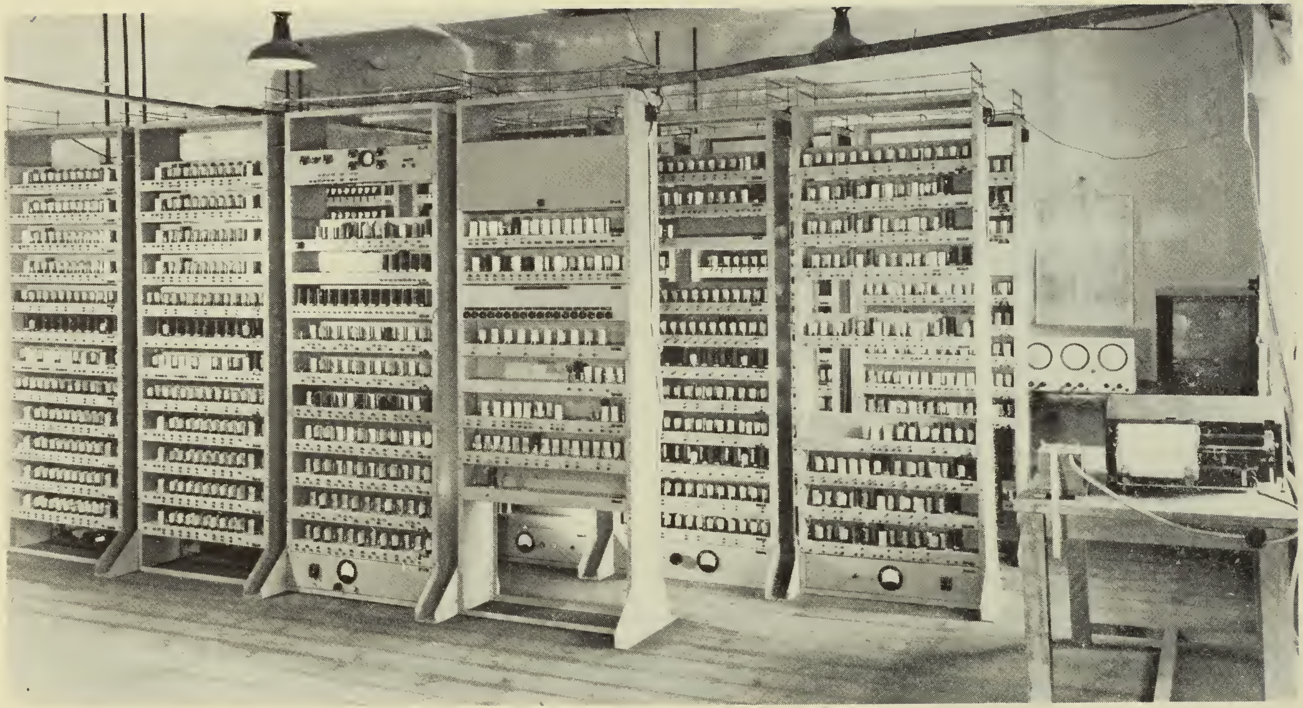
For these reasons, the authors chose to use the EDSAC as their model in writing the First Edition. Since the methods described for the EDSAC may readily be translated into order codes for other machines, the authors have retained in this revised edition the same general plan as that of the First Edition. However, the growing use of other types of machines has led them to expand the book to include other computers, in the hope of making the book useful to those working with any stored-program computer.

In the Addison-Wesley
Mathematics Series
Eric Reissner
Consulting Editor

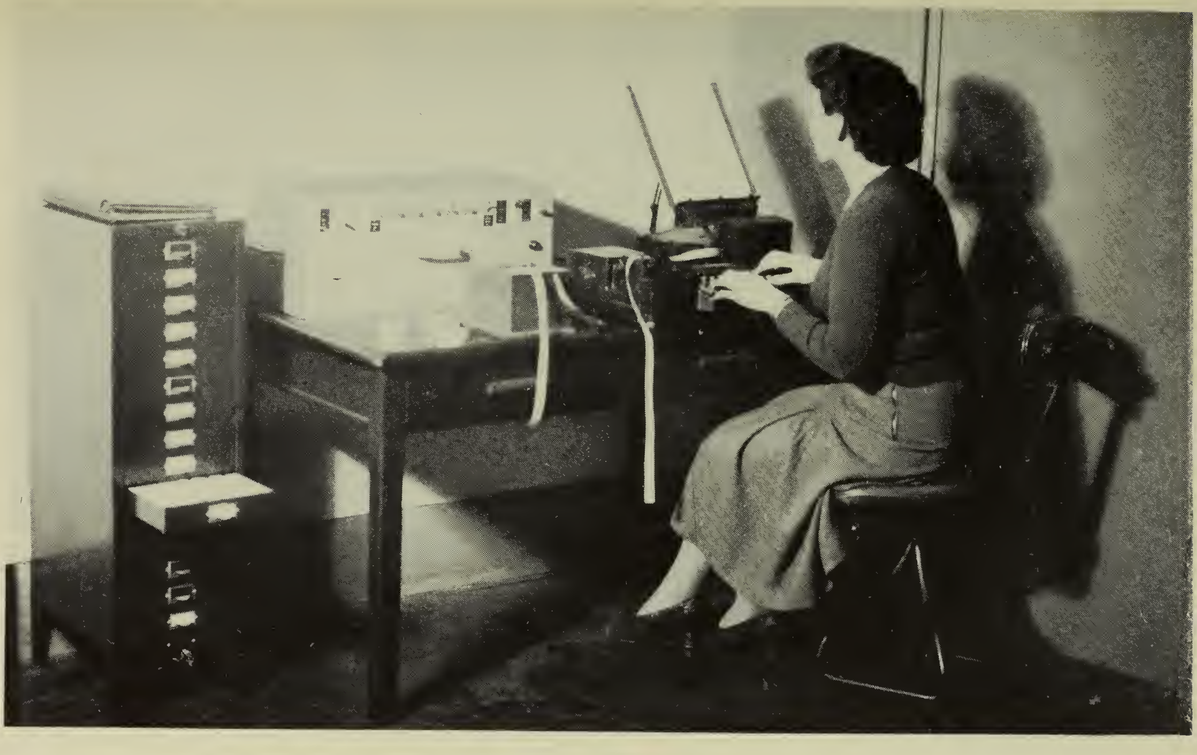
W–6

1 50

THE PREPARATION OF PROGRAMS FOR AN ELECTRONIC DIGITAL COMPUTER



A general view of the EDSAC. The racks in the front row contain (from left to right): part of the store (two racks), pulse generator, and input-output units. Behind are three racks containing the control, and, in the rear, the remainder of the store (two racks) and the arithmetical unit (three racks). On the extreme right of the photograph may be seen the tapereader for the input tape, and the teleprinter on which results are printed.



The library of tapes on which subroutines are punched is contained in the steel cabinet shown on the left. The operator is punching a program tape on keyboard perforator. By placing library tapes on the tapereader shown in the center of the photograph, the operator can copy them mechanically onto the tape she is preparing.

ADDISON-WESLEY MATHEMATICS SERIES

Eric Reissner, *Consulting Editor*

Apostol—Mathematical Analysis, A Modern Approach to Advanced Calculus

Bardell and Spitzbart—College Algebra

Dadourian—Plane Trigonometry

Davis—Modern College Geometry

Davis—The Teaching of Mathematics

Fuller—Analytic Geometry

Gnedenko and Kolmogorov—Limit Distributions for Sums of Independent Random Variables

Kaplan—Advanced Calculus

Kaplan—A First Course in Functions of a Complex Variable

LeVeque—Topics in Number Theory, Vols. I and II

Martin and Reissner—Elementary Differential Equations

Meserve—Fundamental Concepts of Algebra

Meserve—Fundamental Concepts of Geometry

Munroe—Introduction to Measure and Integration

Perlis—Theory of Matrices

Spitzbart and Bardell—College Algebra and Plane Trigonometry

Spitzbart and Bardell—Plane Trigonometry

Springer—Introduction to Riemann Surfaces

Stabler—An Introduction to Mathematical Thought

Struik—Differential Geometry

Struik—Elementary Analytic and Projective Geometry

Thomas—Calculus

Thomas—Calculus and Analytic Geometry

Vance—Trigonometry

Vance—Unified Algebra and Trigonometry

Wade—The Algebra of Vectors and Matrices

Wilkes, Wheeler, and Gill—The Preparation of Programs for an Electronic Digital Computer

THE PREPARATION OF PROGRAMS FOR AN ELECTRONIC DIGITAL COMPUTER

by

MAURICE V. WILKES, F.R.S.

DAVID J. WHEELER

and

STANLEY GILL

SECOND EDITION

ADDISON-WESLEY PUBLISHING COMPANY, INC.
READING, MASSACHUSETTS, U.S.A.

ADDISON-WESLEY PUBLISHING COMPANY, INC.

Printed in the United States of America

Library of Congress Catalog Card No. 57-8713

SECOND EDITION — *December 1957*

PREFACE

When the first edition of this book was published in 1951, very few digital computers were in operation, and our own experience was confined exclusively to the EDSAC. However, we pointed out in the Preface that the methods described might be for the main part readily translated into the order codes of machines other than the EDSAC, and we went on to say: "It is hoped, therefore, that those who have charge of similar machines, or who are faced with the task of putting a new machine into operation, will find some of the ideas and methods presented here of assistance. It is hoped also that the book will be of use to those who wish to know something about the form in which problems are presented to an automatic digital calculating machine and who wish to assess the possibilities of the application of such machines to their own subjects." Comments which have reached us, and the continued demand for the book, have indicated that these hopes have been to some extent fulfilled, and we have therefore been encouraged to take advantage of the fact that reprinting would in any case be necessary, and to prepare a new edition. This follows the same plan as the first edition, but little of the original material has been taken over unchanged. Much more attention is now paid to machines other than the EDSAC, and the book is offered as a general introduction to programming for any machine of the stored-program type.

Programming cannot be taught in the abstract, or learned without practice, and any book on programming must use the order code of some particular machine, real or hypothetical. Fortunately, the order code of the EDSAC lends itself well to the purposes of a book on programming, being fairly straightforward and relatively easily memorized. Moreover, the EDSAC is a single-address binary machine, and a recent survey by M. H. Weik (for reference see Bibliography) shows that, of the types of machine currently in use, about 50% use single-address order codes and about 60% work in the binary scale. The order code of the EDSAC contains a few of the minor inconsistencies and complications invariably found in the order codes of real machines; the order codes of hypothetical machines especially designed for teaching may be free of these inconsistencies and complications, but are inevitably somewhat artificial in consequence. One advantage of using the order code of an established machine is that it is possible to draw on experience of programming and of the teaching of programming. The first two chapters of Part 1 of this edition are based on courses of programming given in this laboratory over a period of years; they include several sets of examples to be worked by the reader, and solutions are given to selected examples. Chapter 3 surveys the various types of order code to be found in digital computers, and is intended to give the reader some idea of what to expect when he meets a new machine. Later chapters deal with input and output, the contents of a library of subroutines, error diagnosis, and advanced methods. Parts 2 and 3 contain detailed information about the library of subroutines used with the EDSAC.

Some attention is given in Chapter 3 to "minimum access" coding as used in conjunction with many machines fitted with magnetic drums as their main stores. This subject cannot be taken very far in a general book on programming, however, since so much depends in the case of a particular machine on the precise timing of the various operations. For somewhat similar reasons, it is not possible to say very much about the use of storage on more than one level. We have, however, included in Chapter 8 an introduction to the problems encountered when an auxiliary store is used to supplement a higher speed main store.

We would like to repeat the acknowledgements we made in the preface to the first edition of the book to all the workers in the Mathematical Laboratory who helped in the development of the methods described, and to extend these acknowledgements to include those who have joined the laboratory since the first edition was published. We are especially grateful to Professor D. R. Hartree, F. R. S., for constant help and encouragement; Chapters 1 and 2 owe a good deal to some material prepared by him, on the basis of lecture notes by one of us, for a booklet issued in connection with a Summer School on Program Design. We would like to say, once again, how conscious we are of the heavy debt we owe to many colleagues working in other laboratories who have freely shared their ideas with us.

We would like again to thank Mr. E. N. Mutch for undertaking the heavy task of preparing Parts 2 and 3 for the press and for giving a great deal of editorial assistance with the preparation of the book as a whole. We are indebted to Mrs. M. O. Mutch and to Mr. J. Leech for reading the manuscript and for making a number of helpful comments.

We would also like to make acknowledgement to the Council of the Royal Society of Edinburgh for permission to reproduce a table, given in Section 5–7, from a paper which appeared in Volume 62A of their Proceedings, and to Dr. Miller for giving similar permission on behalf of the authors.

Cambridge, England
August 1957

M. V. W.
D. J. W.
S. G.

CONTENTS

Part One

PART TWO

Part Three

PART ONE

CHAPTER 1

THE ELEMENTS OF PROGRAM DESIGN

1–1 Introduction. A digital computing machine can perform only the basic operations of arithmetic, such as addition, subtraction, multiplication, and division. In order to be able to solve a mathematical problem such as the integration of a differential equation, it is first necessary to express the problem as a sequence of such operations. This may call merely for some expenditure of labor, or it may involve considerable mathematical manipulation; for example, where derivatives or integrals are involved it may be necessary to replace the continuous variables by variables which change in discrete steps.

If the computation were to be performed by a human computer it would be possible to communicate the problem to him in a series of instructions or orders, each specifying an elementary arithmetical operation. It is convenient to use the same nomenclature when speaking of a machine, but here the "instructions," or "orders," are groups of symbols punched on a paper tape, or prepared in some other form which can be fed into a machine. A sequence of orders for performing some particular calculation is called a *program.* It must contain everything necessary to cause the machine to perform the required calculations, and every contingency must be foreseen. A human computer is capable of reasonable extension of his instructions when faced with a situation which has not been fully envisaged in advance, and he will have past experience to guide him. This is not the case with a machine.

1–2 Types of automatic computing machine. In early automatic computing machines which were mechanical or electromechanical in action, the orders were usually punched in coded form on paper tape, one group of holes corresponding to each order. These holes were read by a sensing device, which caused the machine to perform the operation called for; the tape was then advanced so that the next group of holes came under the reading head, and the next order was similarly executed. In addition to a sensing mechanism for the main program tape, several other sensing mechanisms were usually provided. These could be used to read endless loops of tape which contained orders for performing parts of the program which had to be repeated a number of times. Control of the machine was passed from one tape to another as required. Examples of machines which worked in this manner are the Automatic Sequence Controlled Calculator at Harvard University, and relay calculators built by the Bell Telephone Laboratories, and by Harvard University. The system just described, while admirable for controlling a relay machine, would not be fast enough for a machine in which the computation is performed by electronic means, and in which it is desired to realize the very high speed which this makes possible. The ENIAC, which was the first purely electronic machine to be built, therefore used a system in which the various steps of the program were initiated by "program pulses" passed from one unit of the machine to another. For example, to cause a number standing in one register, or "accumulator," to be added to the number standing in another accumulator, both accumulators needed to be stimulated by a program pulse, one to transmit and one to receive. When the operation was finished, both accumulators emitted a pulse, and one of these (it did not matter which, since they both occurred at the same time) was used to stimulate the next action. Putting a problem on the machine consisted, therefore, of making a large number of connections by means of plugs and sockets, and setting a number of switches. The main objection to this system is that it takes some time to change over from one problem to another. In all later machines the orders are expressed in a coded form, and placed in advance in a quick-access store, or memory, from which they are subsequently taken and executed one by one. The orders are usually passed into the machine by means of a punched tape or some similar medium, but this is used simply as an intermediary in the process of transferring the program to the store; it does not control the computing action of the machine directly.

A store, or memory, is also needed in automatic computing machines for the purpose of holding numbers, and in most machines the same store is used to hold the orders; this is made possible by the device of expressing the orders in a numerical code. Many machines working on these principles are now in operation. The principles derive from a report, drafted by J. Von Neumann in 1946, in which the design was outlined of a new machine (the EDVAC) then projected by the Moore School of Electrical Engineering (University of Pennsylvania) where the ENIAC had been built. The designers of the ENIAC, Dr. J. Presper Eckert and Dr. John W. Mauchly, were closely associated with the origin of this new project. It was found that machines designed along the lines laid down in this report were much smaller and simpler than the ENIAC, and at the same time more powerful.

The methods by which programs are prepared for all machines of the EDVAC type are, as might be expected, similar, although the details vary according to the different order codes used. Anyone familiar with the use of one machine will have no difficulty in adapting himself to another. In this book we shall first describe in detail how programs are prepared for the EDSAC, a machine used in the University Mathematical Laboratory, Cambridge. Later, we shall turn our attention to other machines.

The EDSAC, like the EDVAC, uses the binary system for internal calculation, but this is not an essential feature, and some other machines use the decimal system. Even if the binary system is used inside the machine it is only rarely that the programmer needs to take notice of this fact, since input and output are ordinarily performed in the decimal system, the necessary conversion being done by the machine itself as part of the program.

The following comparisons between the decimal and binary systems will serve to explain the binary system to those unfamiliar with it.

Decimal			Binary		
0.1	represents	$\frac{1}{10}$	0.1	represents	$\frac{1}{2}$
0.01	"	$\frac{1}{10^2}$	0.01	"	$\frac{1}{2^2}$
0.001	"	$\frac{1}{10^3}$	0.001	"	$\frac{1}{2^3}$ etc.

Thus, in the binary scale, all digits are either 0 or 1, and, for example,

0.101	represents	$\frac{5}{8}$
0.01101	"	$\frac{13}{32}$

1–3 The EDSAC. In order to be able to construct programs, some knowledge of the main units of the machine and their interconnection is required, although it is not necessary to understand the precise mode of functioning of the various electronic circuits. There are, from the point of view of the programmer, four main parts to the machine: the *store*, or *memory*, the *arithmetical unit*, and the *input* and *output* mechanisms. There is also the control unit which emits the electrical signals that control the action of the other units. Figure 1 shows the connections between the various units.

The main things which a programmer needs to know about a machine are:

(1) Its *order code*, that is, the different elementary operations the machine can carry out, and how each is specified to the machine.

(2) The forms in which numbers and orders are represented within the machine.

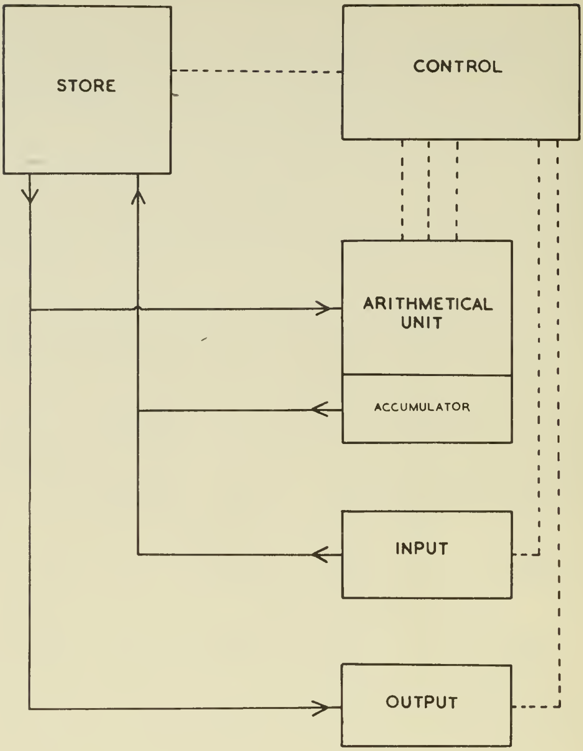


FIG. 1. Schematic diagram of the EDSAC

(3) How the machine, having carried out the operation specified by one instruction, determines the next instruction.

(4) The form in which the program and any numerical data required are supplied to the machine, and the form in which results are provided by it.

In the following sections the order code of the EDSAC is introduced gradually, so that the student can first, by practice, become accustomed to the significance of some of the more commonly used orders, and does not need to become familiar right from the start with the full order code. The full order code is given for purposes of reference in Appendix 2. Inside the machine, numbers and orders are expressed in the binary forms described in Sections 1–6 and 1–7. However, as already mentioned, the programmer need not be familiar with binary arithmetic; for most purposes, other than shifting operations, he can forget that numbers are represented in this form in the machine. Five-hole punched paper tape is used for both input and output. The machine normally reads in the whole program, and then proceeds to execute the orders it has read.

1–4 Store. The store is divided into a number of registers or *storage locations;* the content of a storage location is a sequence of 0's and 1's, and may represent an order or a number. The term *word* is used for the content of a storage location if it is desired to refer to it without specifying whether it represents a number or an order.

In order to identify the storage locations, each is labelled by a number called its *address* (or the address of its content). The notation $C(n)$ will be used for "the content of storage location n."

Each storage location in the EDSAC holds 17 binary digits. In words representing numbers, the binary point is regarded as being to the right of the extreme left-hand digit; this digit (the most significant digit) is used as a sign indicator and is referred to as the *sign digit.*

1–5 Arithmetical unit. The central part of the arithmetical unit is a register called the *accumulator* which plays the part of the result register in a desk machine; it accumulates the sum of numbers added into it until it is cleared. In order to provide facilities which will be explained later (in Section 2–10) the capacity of the accumulator is 70 digits; there is, therefore, plenty of room to hold the full 33-digit product of two 17-digit numbers. As in a storage location, the conventional position of the binary point is immediately to the right of the extreme left-hand digit.

The content of the accumulator will be written $C(\text{Acc})$. The arithmetical unit has also a register for holding one of the factors in a multiplication; this register is called the *multiplier register,* and its content will be written $C(R)$.

At any stage in a calculation the most significant 17 digits of the content of the accumulator can be placed in any specified storage location, say n, by means of an appropriate order. When this is done, the previous content of location n is destroyed and replaced by the word transferred from the accumulator.

1–6 Form of numbers in the machine. Positive numbers inside the machine are all less than 1, and have a 0 as the sign digit. A negative number $-x$ (where $0 < x \leq 1$) is represented by a 1 in the sign-digit position, followed by the digits of $(1 - x)$; for example,

$$1.1100\ldots \quad \text{represents} \quad -(1 - \tfrac{3}{4}) = -\tfrac{1}{4}.$$

(Note that the number does not represent $1\frac{3}{4}$.) Another way of explaining the representation of negative numbers is to regard the sign digit as an ordinary numerical digit, and to say that $-x$ is stored as the number $(2 - x)$. Note in particular that

$$1.0000\ldots \quad \text{represents} \quad -1.$$

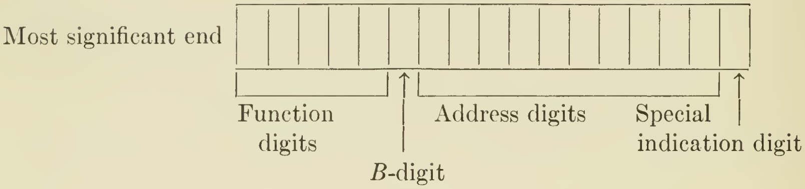
The range of numbers which can be represented in the machine is therefore $-1 \leq x < 1$. Scale factors must be introduced in order to deal with numbers outside this range (see Section 2–12.)

Any attempt to work with numbers outside the range just specified will cause the capacity of the accumulator to be exceeded, and will usually give a wrong answer; for example, addition of $\frac{1}{2}$ (=0.10) and $\frac{3}{4}$ (=0.11) gives 1.01, which has a 1 in the sign-digit position, and will be treated by the machine as $-\frac{3}{4}$ and not as $1\frac{1}{4}$.

A storage location, or the accumulator, is said to be *clear* when its content is 0. An operation which makes the content 0 is said to clear the storage location, or accumulator.

1–7 Form of orders in the machine. Orders are of the *one-address* type; that is, each order which refers to the store (some do not) refers to one address only in the store.

In a word representing an order the significance of the digits is as follows:



The first five digits (counting from the left-hand end) specify what kind of operation (e.g., addition, subtraction, transfer to store) is to be carried out; the next digit is the *B*-digit, whose function will be explained in Section 1–13, the next ten digits specify the address of the storage location involved in the operation (e.g., address of addend, address in the store to which the content of the accumulator is to be transferred). The last digit is used for special indications, as will be explained later.

The function digits 00101, for example, specify the operation of transferring from the accumulator to the store, and 10101 is the binary form of 21; interpreted as an order, therefore, the word

00101, 0, 0000010101, 0

means "transfer the content of the accumulator to storage location 21." Note that the commas are written here to make clear the structure of the word; they do not represent anything in the store or in the accumulator of the machine.

1–8 Storage of orders. Orders are normally placed in storage locations numbered in the sequence in which the machine is required to carry out the operations specified. The control unit is designed so that the machine, having carried out the operation specified by the order in m, automatically takes $C(m + 1)$ as the next order, unless the order in m specifies otherwise.

1–9 Written form of orders. When orders are written the function digits are specified by a single letter, and the address is written in decimal form; for example, the order given above is written $T\ 21$. The written forms for some of the more important operations are:

$A\ n$	Add $C(n)$ to $C(\text{Acc})$, placing the result in the accumulator.
$S\ n$	Subtract $C(n)$ from $C(\text{Acc})$, placing the result in the accumulator.
$T\ n$	Transfer $C(\text{Acc})$ to storage location n, and *clear* the accumulator.
$U\ n$	Copy $C(\text{Acc})$ into storage location n, and *retain* in the accumulator.
$H\ n$	Replace $C(R)$ by $C(n)$.
$V\ n$	Multiply $C(n)$ by $C(R)$, and add the result into the accumulator.
Z	Stop (no address required).

Other orders will be introduced later.

Note that the content of storage location n is not affected by the orders A, S, H, or V, which may be thought of as taking a copy of the content of the location, which is itself left undisturbed. The content of storage location n is, however, destroyed, and is replaced by the current content of the accumulator, as a result of the execution of a T- or U-order. The content of the multiplier register, set by an H-order, remains unchanged until it is reset to a new value by another H-order.

1–10 Some simple examples. We shall now consider what orders are required to carry out some simple calculations. These are to be thought of as forming part of a larger calculation, and the numbers operated on are to be thought of as having been calculated and placed where they are stated to be, at an earlier stage in the work. Unless the contrary is specified, it is supposed that the accumulator is initially clear, and is to be left clear on the completion of each example.

Example 1. $C(10) = x$ (i.e., the number x is in storage location 10), $C(14) = y$; to form $x + y$ and place it in 8. The orders required, and the content of the accumulator after each order has been carried out, are:

Order	$C(\text{Acc})$	Notes
A 10	x	Accumulator initially clear
A 14	$x + y$	
T 8	0	

the orders being taken in this sequence. Now the control unit takes the orders from successive storage locations unless explicitly instructed to do otherwise; hence we arrange for these orders to be placed in consecutive storage locations, say 100, 101, 102, as follows:

Storage location	Content
100	A 10
101	A 14
102	T 8

EXAMPLE 2. $C(10) = a$, $C(11) = b$, $C(20) = x$, $C(30) = y$; to place ax in 12 and $y(ax + by)$ in 13.

Storage location	Content	$C(\text{Acc})$	Notes
100	H 20	—	x to multiplier register
101	V 10	ax	Accumulator initially clear
102	U 12	ax	Place ax in 12, and retain in accumulator
103	H 30	—	y to multiplier register
104	V 11	$ax + by$	
105	T 0	0	Place $(ax + by)$ in 0
106	V 0	$(ax + by) \cdot y$	
107	T 13	0	

Notes: (1) Since the content of the multiplier register set by an H-order remains unaltered until reset by a further H-order, $C(R)$, set equal to y by the order in 103 and used by the V-order in 104, remains available for further use by the V-order in 106.

(2) The multiplicand must be the content of a storage location; hence the quantity $(ax + by)$ formed in the accumulator must be moved out into a storage location before it can be multiplied by y. Any spare storage location may be used for such temporary storage of a number which is going to be used again almost immediately afterwards, but it is often convenient to use location 0 for this purpose.

(3) The result of a multiplication of two 17-digit numbers is a 33-digit number, but only 17 of these digits are transferred to the store by a *T*- or *U*-order; the rest are cleared out by the *T*-order and lost. The products should strictly be rounded off before being transferred to the store, but this refinement will be ignored for the present.

Exercises A

1. Given x in 20 and y in 22, place $(x + y)$ in 30 and $(2x - y)$ in 32.
2. Replace $C(0)$ by its square.
3. Replace $C(0)$ by its cube.
4. Divide $C(4)$ by π^2, returning the result to 4. It may be assumed that $1/\pi^2$ is in 10.
5. Given $z = x + iy$, where $x = C(4)$, $y = C(6)$, place the real and imaginary parts of z^2 in 8 and 10 respectively, and $|z|^2$ in 0.
6. Evaluate $x^3 + x^4$, where $x = C(10)$, and place the result in 0.
7. Evaluate $ab + cd + ef$, where a, b, c, d, e, f are in 10, 11, 12, 13, 14, 15, respectively. Place the result in 16.
8. a, b, c, and x are the contents of 100, 102, 104 and 60, respectively. Form $ax^2 + bx + c$ and place it in 4.
9. The dimensions in inches of a rectangular block are given in 50, 51, and 52. Place its surface area (in square inches) in 100, and its volume (in cubic inches) in 101.

1–11 Jump orders. The process of taking the next order out of the sequence in which the orders are stored is known as a *jump* or *transfer of control.* After a jump, orders are taken serially starting from the new address, until another jump order is reached. Such a jump can be made either unconditionally or conditionally, according to the value of some quantity obtained in the course of the calculation. In the EDSAC, conditional jumps are conditional on the sign of the content of the accumulator. The written forms of three kinds of jump order (there are others which will be introduced later) are:

F n	Take $C(n)$ as the next order.
G n	If $C(\text{Acc})$ is *negative* (i.e., has a 1 in the sign-digit position), take $C(n)$ as the next order; otherwise proceed serially (i.e., do not jump).
E n	If $C(\text{Acc})$ is positive or zero (i.e., has a 0 in the sign-digit position), take $C(n)$ as the next order; otherwise proceed serially (i.e., do not jump).

F n gives an absolute transfer of control. *E n* and *G n* give conditional transfers of control, and it will be observed that *E n* causes a jump to take place when $G\ n$ would not do so, and vice versa. Conditional jump orders enable the machine to choose between one course of action and another, according to some specified criterion. Use of G- or E-orders for this purpose requires that any criterion used shall be expressed in such a form that it depends on the sign digit of the number in the accumulator.

Note that the order $F\ n$, if placed in storage location n, is repeated indefinitely once control has been sent to that location, so that the machine is to all intents and purposes brought to a stop. This is known as a *dynamic stop*, and may be used instead of an ordinary stop brought about by a Z-order.

It often happens that when a conditional jump has taken place the content of the accumulator is no longer required, having perhaps only been formed in order that its sign might provide the criterion for the jump. The accumulator must then be cleared before the calculation can proceed. This could always be done by a T-order, but both time and storage space are saved, and programming is simplified, if this clearing operation is incorporated in the jump order. This can be done (in the case of an E- or a G-order) by making the special indication digit (the extreme right-hand digit) a 1. A digit in this position in an order is not normally used as a numerical digit, and to emphasize this it is represented by a special symbol π in the written form of the order. Thus:

$G\ n\ \pi$ — If C(Acc) is negative, take $C(n)$ as the next order, and clear the accumulator; otherwise proceed serially (without clearing the accumulator).

$E\ n\ \pi$ — If C(Acc) is positive or zero, take $C(n)$ as the next order, and clear the accumulator; otherwise proceed serially (without clearing the accumulator).

Note that the accumulator is cleared only if a jump does occur.

Example 3. Replace $C(4)$ by its modulus; that is, if $C(4)$ is negative, place $-C(4)$ in 4, otherwise leave $C(4)$ unaltered. Let x be written for the original $C(4)$.

Location	Content	C(Acc)	Notes
100	S 4	$-x$	Accumulator initially clear
101	G 103 π	$-x$	Test sign of $-x$, jump and clear the accumulator if negative
102	T 4	0	If x negative, place $-x$ in 4
101 → 103	Next order		

If x is negative, when the conditional jump order in 101 is encountered the content of the accumulator is positive and equal to $|x|$, the quantity required. The machine therefore proceeds serially and takes its next order from 102 and plants $|x|$ in 4. If x is positive, the content of the accumulator is negative when the jump order is encountered; in this case location 4 already contains $|x|$ and all that is required is that the machine should proceed to the next part of the program with the accumulator clear. This is just the situation provided for by an order of the form $G\ n\ \pi$. Note the use of the arrow to indicate that control can arrive at 103, having been transferred from 101.

Exercises B

1. Place $-|C(4)|$ in 0.

2. Place $|C(4) - C(6)|$ in 0.

3. Given $C(0) \geq 0$, place the larger of $C(0)$ and $|C(4)|$ in 0.

4. If $C(4)$ and $C(6)$ have the same sign, place the product of $C(4)$ and $C(6)$ in 0; if they differ in sign, place $|C(4) - C(6)|$ in 0.

5. $C(6)$, $C(7)$, and $C(8)$ are all positive and less than $\frac{1}{2}$. Stop the machine if it is not possible for these numbers to represent the lengths of the sides of a triangle. (*Note:* the sum of any two sides of a triangle must be greater than the third side.)

6. If $C(0) < 0$, put $C(4)$ in 6, and if $C(0) > 0$, put $C(8)$ in 6. In either case reverse the sign of $C(0)$.

7. What happens in your program for question 6 if $C(0) = 0$? Modify your program if necessary so that the machine stops in this case.

8. 50 contains either $\frac{5}{8}$ or $\frac{3}{16}$. Whichever it is, replace it by the other.

1-12 Repeated groups of orders. Most extensive calculations involve performing the same, or closely similar, groups of operations repeatedly on different sets of numbers. Such repeated groups of operations are represented in a program by groups of orders which are executed a number of times by the machine; it will be seen later that minor modifications may be made to the orders between successive uses. The number of repetitions required is sometimes known in advance and is sometimes not known. In any case a jump order is required to return control from the end of the repeated group of orders to the beginning, and this jump must be conditional in order that the machine may pass on to the next part of the program after a sufficient number of repetitions.

Example 4. $C(4)$ is negative. Add $C(0)$ repeatedly to it until the result becomes positive, and place the result in 4. Let $C(4) = -x$ (x is positive), $C(0) \doteq y$.

Storage location	Order	$C(\text{Acc})$	Notes
100	A 4	$-x$	
102 → 101	A 0	$-x + y, -x + 2y, -x + 3y$ $-x + 4y$ . . . successively	
102	G 101		Test and jump if $C(\text{Acc}) < 0$
103	T 4		

Notes: (1) A procedure of this kind could be used for reducing a negative angle θ to the first or second quadrant, by successive addition of π, as a preliminary to calculating its sine or cosine. Since numbers in the machine must lie in the range $-1 \leq x < 1$, it would be necessary to work with $k\theta$ and $k\pi$ instead of θ and π, where k is a scale factor chosen to bring both quantities within the capacity of the machine.

(2) In this example the number of repetitions is not known in advance, and is controlled by the sign of the number calculated.

EXAMPLE 5. $C(10) = a \ (\leq \frac{1}{2})$, $C(20) = y$. To form

$$(a + ay + ay^2 + \cdots + ay^9)$$

and place it in 0.

There are several ways in which the required result could be evaluated. We shall use the procedure represented by the formula

$$[\{(a \cdot y + a) \cdot y + a\} \cdot y + a] \cdot y + a \ldots$$

This is a special case of a standard general procedure for evaluating polynomials. Thus we want repeatedly to multiply by y and add a, and to carry out this pair of operations the right number of times. Since there is a sequence of multiplications by y, let y be placed in the multiplier register. If S_n is the partial result after n repetitions of the process, then

$$S_{n+1} = yS_n + a.$$

Since the final result is required in 0, let each S_n, as it is formed, be placed in 0. Then the repeated group of orders (y having been placed in the multiplier register as part of the preliminary preparation) is:

	Order	$C(\text{Acc})$	Notes
(1)	V 0	yS_n	
	A 10	$yS_n + a = S_{n+1}$	
	T 0	0	$C(0) = S_{n+1}$

This has to be carried out 9 times starting with $C(0) = a$, or 10 times starting with $C(0) = 0$; we shall adopt the first of these alternatives. It is necessary to count the number of times this repeated group of orders has been carried out, to provide a criterion for the completion of the calculation.

The following is one method of counting. It depends on the use of a storage location as a counting register or *counter;* the content of this register is sometimes called the *count.* To count 9 repetitions, we arrange to place -9 in the counter as part of the preliminary preparation, and to increase this number by unity after each repetition; then

after first time, count $= -8$,
after second time, count $= -7$,
.
.
.
after ninth time, count $= 0$.

In the EDSAC, counting is usually done in units of 2^{-15}; thus in the present example the machine would start with $-9 \cdot 2^{-15}$ as the content of the counter, and would add $1 \cdot 2^{-15}$ after each repetition. The part of the program concerned with counting is given below. Count is kept in storage location 4, $1 \cdot 2^{-15}$ is stored in 2, and $9 \cdot 2^{-15}$ is stored in 5. n denotes the number of times the sequence of orders has been repeated.

Location	Content	C(Acc)	Notes
	Numbers		
2	$1 \cdot 2^{-15}$		
4	$-(9 - n) \cdot 2^{-15}$		Count, initially $-9 \cdot 2^{-15}$
5	$9 \cdot 2^{-15}$		
	Orders		
98	S 5	$-9 \cdot 2^{-15}$	Set count
$p + 102 \rightarrow$ 99	T 4	0	
100	....		Orders to be repeated 9 times (p in number)
	....		
	....		
$p + 100$	A 4	$-(9 - n) \cdot 2^{-15}$	
$p + 101$	A 2	$-(9 - n - 1) \cdot 2^{-15}$	Count and test for completion
$p + 102$	G 99	$-(9 - n - 1) \cdot 2^{-15}$	
$p + 103$	Next order of program.		

If, when the order in $p + 102$ is reached, an insufficient number of repetitions have been performed, the content of the accumulator is negative, and is equal to the new value of the count; it is placed in the counter by the action of the order in 99, which is reached by a jump from $p + 102$. If, however, the full number of repetitions has been performed, the content of the accumulator is zero, and the machine proceeds to take the next order from $p + 103$.

Notes: (1) The process of counting and testing is carried out after the operation to be counted, and leaves the accumulator clear.

(2) If more, or fewer, repetitions have to be made, only $C(5)$ has to be changed.

(3) The above group of orders includes the initial setting of the count, so it can be used any number of times in the course of a calculation; this is expressed by calling it "self-resetting."

The complete program required for the example is as follows:

Location	Content	C(Acc)	Notes
	Numbers		
2	$1 \cdot 2^{-15}$		
4	$-(9 - n) \cdot 2^{-15}$		Initially $-9 \cdot 2^{-15}$
5	$9 \cdot 2^{-15}$		
	Orders		
95	H 20		Set $C(R) = y$
96	A 10	a	Set first multiplicand and clear accumulator
97	T 0	0	
98	S 5	$-9 \cdot 2^{-15}$	Set count
105 → 99	T 4	0	
100	V 0	yS_n	
101	A 10	$yS_n + a = S_{n+1}$	
102	T 0	0	
103	A 4	$-(9 - n) \cdot 2^{-15}$	
104	A 2	$-(9 - n - 1) \cdot 2^{-15}$	Count and test for completion
105	G 99	$-(9 - n - 1) \cdot 2^{-15}$	
106	Next order of program		

This example illustrates the procedure which it is often convenient to use in programming a repetitive process:

(1) program the process to be repeated,

(2) program how to get out of the repetitive process,

(3) program how to enter it, and any preparatory steps.

1–13 The use of the *B*-register. Situations in which a given group of orders must be obeyed, exactly as they stand, a number of times before the machine passes to the next part of the calculation are comparatively rare, but it is very common to find a situation in which a group of orders must be repeated a number of times with a slight modification each time. Usually the modification concerns the addresses specified in certain of the orders.

EXAMPLE 6. $C(10) = a_0$, $C(11) = a_1, \ldots, C(19) = a_9$; $C(20) = y$. To form the polynomial $(a_0y^9 + a_1y^8 + \cdots + a_8y + a_9)$ and place it in 0. This example is similar to, but more general than, the one considered in the last section. The polynomial may be evaluated by repeated application of the formula $S_{n+1} = yS_n + a_n$, which differs from the corresponding formula in the earlier case only by the occurrence of a_n instead of a. The same group of orders from Section 1–12 can be used, provided that it is arranged that the address of the number referred to by the second order is increased by 1 on each repetition. This can be contrived by making use of the *B-register.*[*]

The *B*-register is capable of holding a single integer which is placed there by the order

B m	Replace the content of the *B*-register by the number *m* (*N.B. not* the content of location *m*).

It is possible for the programmer to arrange that the content of the *B*-register is added to the address specified in any order, as that order is on its way from the store to the control unit of the machine. He indicates that this is to happen by writing the letter *S* after the function letter of the order. If the *B*-register contains the number 10 the sequence of orders

```
V  0
AS 0
T  0
```

thus has precisely the same effect as that labelled (1) in Section 1–12. If, however, the sequence is followed by an order which increases the number in the *B*-register by 1, it will, if repeated, be equivalent to

```
V  0
A 11
T  0
```

[*] This device was originated at Manchester University and given the name *B-line*; other names are *index register* and *modifier register*.

The number in the B-register can be increased by means of the order

$BS\ m$ | Increase the content of the B-register by m (*not* by the content of location m).

Here the letters BS are best regarded as being equivalent to a single function letter. It is sometimes necessary to decrease the content of the B-register, for example to subtract m from it. This is indicated on the program sheet by writing the letter S (which in this context stands for subtract) after the address of the order. Thus we have

$BS\ mS$ | Decrease the content of the B-register by m.

Similarly, we have

$B\ mS$ | Replace the content of the B-register by $-m$.

It will be seen that the letter S is used in different senses according to the type of order, and according to whether it follows the function letter or the address. Inside the machine, orders which are punched with an S following the function letter have a 1 in the position following the function digits; those which have no S punched after the function letter have a 0 in this position. The digit 0 or 1 is therefore called the *B-digit.* (See Section 1–7.)

It will now be seen that the formula given at the beginning of this Section can be evaluated by the repetition, an appropriate number of times, of the sequence of orders given below; it is assumed that initially location 0 is clear, that the multiplier register contains y, and that the B-register contains the number 10.

Location	Content	Notes
101	$V\ \ 0$	Replace S_n in 0 by $S_{n+1} = yS_n + a_n$
102	$AS\ 0$	
103	$T\ \ 0$	
104	$BS\ 1$	Increase number in B-register by 1

In order to arrange for these orders to be repeated the appropriate number of times it would be quite possible to make use of a counting sequence similar to that described in Section 1–12. However, a simpler method can be used, since an order is provided in the order code of the EDSAC which makes it possible to use the number in the B-register for counting. This order gives a conditional transfer of control, and is as follows:

J n	If the content of the *B*-register is not zero take $C(n)$ as the next order; otherwise proceed serially.

This order thus transfers control unless the content of the *B*-register is zero. It is therefore necessary to arrange that the number in the *B*-register either starts by being positive and decreases to zero, or starts by being negative and increases to zero. The latter procedure is appropriate in the present case and the orders, assuming location 0 is initially clear, are as follows:

```
       100 | B    10 S | Place $-10$ in B-register
105 →  101 | BS    1   | Increase number in B-register by 1
       102 | V     0   | ┐
       103 | AS   19   | │ Replace $S_n$ in 0 by $S_{n+1} = yS_n + a_n$
       104 | T     0   | ┘
       105 | J   101   | Jump unless number in B-register is
           |           |   zero
       106 | ......... |
```

If the constants a_0, a_1, a_2, etc., were stored in reverse order, that is, a_9 in storage location 10, a_8 in storage location 11, etc., it would be necessary to use a decreasing count as follows:

```
       100 | B    10   | Place 10 in B-register
105 →  101 | BS    1 S | Decrease number in B-register by 1
       102 | V     0   | ┐
       103 | AS   10   | │ Replace $S_n$ in 0 by $S_{n+1} = yS_n + a_n$
       104 | T     0   | ┘
       105 | J   101   | Jump unless number in B-register is
           |           |   zero
       106 | ......... |
```

Notes: (1) The same set of orders can be used if more terms are to be included in the sum, or if the coefficients are to be taken from a different set of (consecutive) storage locations, provided that the orders in 100 and 103 are suitably modified.

(2) The nonrepetitive sequence of orders *V* 0, *A* 19, *T* 0, *V* 0, *A* 18, *T* 0, etc., would be faster in operation, but would take up much more storage space and would lack the flexibility indicated in note (1).

(3) If the sum of the address in the order and the number in the *B*-register exceeds 1024, the sum is formed mod 1024; overflow into the *B*-digit position does not take place. It is only the address digits and the special indication digit which may be modified by the use of the *B*-register.

Exercises C

1. Clear storage locations 100 through 199.
2. Place in 4 the sum of the moduli of the contents of 100 through 199.
3. Place in 4 the sum of the squares of the contents of 100 through 199.
4. Place $C(n + 2)$ in n, for $n = 20$ through 40.
5. Test the numbers in storage locations 150, 151, 152, . . . one after another until a number x, satisfying $\frac{3}{16} \leq x \leq \frac{4}{16}$ is found; place x in 0.
6. Storage locations 200 through 249 contain a series of positive numbers. If any one of these numbers is greater than $\frac{1}{10}$, multiply all the numbers by $\frac{1}{10}$.
7. a, b, c are the contents of 100, 102, 104, respectively, and it is assumed that $a > 0$, $b > 0$, $C(60) = h(>0)$, and that a quantity x_n takes the values 0, h, $2h$, $3h$, ... Place in 4 the value of x_n for which $y = ax_n{}^4 - bx_n + c$ is least, and in 6 the corresponding value of y.
8. Given $x = C(10)$, place x^{13} in 0,
   (i) taking the shortest possible machine time,
   (ii) using the fewest possible orders.
9. Given $x = C(4)$, place
   $(x^9 + 2x^8 + 3x^7 + \ldots + 9x + 10) \cdot 2^{-6}$ in 6,
   (i) using a table of coefficients $1 \cdot 2^{-6}$, $2 \cdot 2^{-6}$ etc., and
   (ii) forming each coefficient from the previous one as it is required.

Compare the machine times and the numbers of orders required.

1–14 Equivalence between orders and numbers; pseudo-orders. In the machine a word which is intended to be used as an order consists of a sequence of digits 0 and 1, just as does a word intended to be used as a number. There is thus an equivalence between orders and numbers. Consider, for example, the order T 21 whose form inside the machine is given in Section 1–7; we have

$$T\ 21 \equiv 0.0101000000101010.$$

The first five digits represent the operation of transfer, and are, by themselves, equivalent to the number $5 \cdot 2^{-4}$; the remaining digits represent the address, and are equivalent to $21 \cdot 2^{-15}$. We thus have

$$T\ 21 \equiv 5 \cdot 2^{-4} + 21 \cdot 2^{-15}.$$

Every function letter similarly has its numerical equivalent; if $\bar{x}$ is the numerical equivalent of $\bar{X}$, then

$$\bar{X}\ n \equiv \bar{x} \cdot 2^{-4} + n \cdot 2^{-15}.$$

The full list of numerical equivalents is given in Appendix 1; a few of the more common are

A	-4
S	12
E	3
G	-5

The B-digit has the value 2^{-5}, and the "special indication digit" the value 2^{-16}; if one or both of these digits are 1's, 2^{-5} or 2^{-16}, or both, must be added to the numerical equivalent of the order; for example,

$$TS\ \ 21 \quad \equiv 5 \cdot 2^{-4} + 2^{-5} + 21 \cdot 2^{-15}$$

$$E\ \ 100\ \pi \equiv 3 \cdot 2^{-4} + 100 \cdot 2^{-15} + 2^{-16}$$

It is sometimes convenient to use the equivalence between orders and numbers to express as an order a word which in the course of a calculation is going to be used as a number. For example, for $-\frac{1}{4}$ we may write A 0. This device is particularly useful when a few numbers have to be put into the machine along with a sequence of orders; it is also useful when the arithmetical unit is used to modify orders, as described in Section 1–15.

Numbers written as orders, but not intended to be treated as such by the machine, are called *pseudo-orders.*

It will be observed that the list in Appendix 1 includes letters corresponding to all numbers in the range $-16 \leq \bar{x} < 16$, whether the letters appear in the order code of the EDSAC or not. All these letters may be used in pseudo-orders. Note, in particular, that the numerical equivalent of P is 0, so that

$$P\ \ n \equiv n \cdot 2^{-15}.$$

Thus, in the program of Section 1–12, the constants 2^{-15} and $9 \cdot 2^{-15}$, in 2 and 5, could be written as pseudo-orders as follows:

Location	Content
2	‖ P 1
5	‖ P 9

Pseudo-orders are conventionally indicated in written programs by the vertical double lines as shown. Note that the constant $2^{-15} \equiv P$ 1, which is frequently required in programs, is usually kept permanently in 2.

***1–15 Use of the arithmetical unit for constructing or modifying orders.** Use of the same form within the machine for numbers and for orders makes it possible to use the facilities of the arithmetical unit for constructing or modifying orders during the course of the calculation. Before the *B*-register was fitted to the EDSAC, very extensive use indeed was made of this facility by programmers, as a study of the first edition of this book will show. A considerable amount of use is still made of it, even now that the *B*-register is available. For this reason, and because many machines do not have a *B*-register, a full treatment of the subject will be given.

EXAMPLE 7. To add the numbers in storage locations 250, 251, 252, . . . , 299, without using the *B*-register.

The following orders add $C(250)$ to the number in 0:

Location	Content
100	A 0
101	A 250
102	T 0

We first show how orders may be written which will increase the address in the second order by 1.

If the words A 250 and P 1 are added as if they were numbers, the result is A 251; for we have (since the numerical value of the function digits corresponding to A is $-4 \cdot 2^{-4}$)

$$\begin{array}{lcl} A\ \ 250 \equiv & -4 \cdot 2^{-4} + 250 \cdot 2^{-15} & \\ P\ \ \ \ 1 \equiv & 0 \cdot 2^{-4} + \ \ \ 1 \cdot 2^{-15} & \\ \hline \text{sum} \ \ = & -4 \cdot 2^{-4} + 251 \cdot 2^{-15} & \equiv A\ \ 251. \end{array}$$

The orders required to increase the address of the second order in the above sequence by 1 are therefore (it being assumed that, as usual, $C(2) = 2^{-15}$)

Location	Content
103	A 101
104	A 2
105	T 101

* This section may be omitted on first reading.

These orders are shown as being placed in locations immediately following those containing the original sequence. If the combined sequence of six orders is operated repeatedly, the effect is to form in 0 the sum of $C(250)$, $C(251)$, $C(252)$, etc. It is only necessary to provide some means of stopping the process at the right time in order to obtain a program for adding up the given sequence of 50 numbers. This may be done by using a counting sequence similar to the one described in Section 1–12. A simpler procedure is to make use of the address in the variable order for counting; for this purpose the T-order in 105 is replaced by a U-order, so that the newly modified order shall remain in the accumulator where it can be tested to see whether the modification has been carried far enough. This is done by subtracting the pseudo-order A 300 (stored in a convenient location, say 50) and testing (by means of a G-order) whether the remainder is positive or negative. The complete program is given below. It is assumed that initially $C(0) = 0$. The entries in the column headed $C(\mathrm{Acc})$ refer to the first time the orders are executed.

Location	Content	$C(\mathrm{Acc})$	Notes
50	∥ A 300		Pseudo-order
107 → 100	A 0	0	Add next number to partial sum
101	A 250	$C(250)$	
102	T 0	0	
103	A 101	A 250	Modify order in 101
104	A 2	A 251	
105	U 101	A 251	
106	S 50	$-49 \cdot 2^{-15}$	Test for end
107	G 100 π	0	
108	. . . .		

Note that the G 100 π order used for testing causes the accumulator to be cleared when control is transferred, and that when this order is encountered on the 50th repetition the accumulator contains zero, so that the machine passes to the next part of the program with the accumulator clear.

When the process is complete, 101 contains A 300. If the orders are to be used again later, means must be provided to reset $C(101)$ to its original value, namely, A 250. This may be done by storing the pseudo-order A 250 in 51, and preceding the orders by the following:

Location	Content	C(Acc)	Notes
98	A 51	A 250	Set $C(101) = A$ 250
99	T 101	0	

The sequence is then self-resetting.

There are various alternative ways in which a program for this example could be written. It is not necessary to go into these in detail, since no new principles are involved, but the following may be given as an example. As before, the entries in the column headed C(Acc) refer to the first time the orders are executed.

Location	Content	C(Acc)	Notes
50	A 300		Pseudo-orders
51	A 301		
52	P 51		
97	S 52	$-51 \cdot 2^{-15}$	
105 → 98	A 51	A 250	
99	T 101	0	
100	A 0	0	Add next number to partial sum. A 250 planted by order in 99
101	A 250	$C(250)$	
102	T 0	0	
103	A 101	A 250	
104	S 50	$-50 \cdot 2^{-15}$	
105	G 98		
106	....		

In the example just given the machine was used to modify the address in an order in the course of the calculation; it is sometimes desired to alter the function digits (and possibly the address as well).

Example 8. The accumulator contains A 100; to place E 102 in 150.

The numerical values of the function letters A and E are given by

$$a = -4 \cdot 2^{-4}, \qquad e = 3 \cdot 2^{-4};$$

for the difference of these, we have

$$e - a = 7 \cdot 2^{-4},$$

and this is the numerical equivalent of U.

Hence we have

$$\begin{array}{rcl} A\ \ 100 \equiv & -4\cdot 2^{-4} + 100\cdot 2^{-15} & \\ U\ \ \ \ 2 \equiv & 7\cdot 2^{-4} + \ \ \ 2\cdot 2^{-15} & \\ \hline \text{sum}\ = & 3\cdot 2^{-4} + 102\cdot 2^{-15} & \equiv E\ \ 102. \end{array}$$

We can thus construct the required order by adding the pseudo-order U 2 to the order A 100 in the accumulator. The orders required for this purpose are

Location	Content	C(Acc)	Notes
3	‖ U 2		Pseudo-order
		A 100	Initial content
100	A 3	E 102	Form E 102
101	T 150	0	Plant E 102 in 150

***1–16 The mix order.** This order was added to the order code of the EDSAC, shortly after the machine was completed, with a view to facilitating the modification of the functions of orders by rendering it unnecessary for the programmer to have regard to the numerical values of function letters. Many other machines have a similar facility. The mix order is as follows

M n	Clear the six most significant digits of the content of the accumulator, and add $C(n)$ to what remains in the accumulator.

As normally used, the effect of this order is to replace the function digits of C(Acc) by those of $C(n)$, and to add the address digits of $C(n)$ to those of C(Acc).

Example 9. To form the sum of products $\Sigma C(n)\cdot C(100+n)$ from $n = 200$ through 249.

The general procedure is similar to that of the calculations considered in earlier examples of this Section, except that there are now two orders in which the addresses have to be advanced by unity for each repetition. The program is as follows:

* This section may be omitted on first reading.

Location	Content	C(Acc)	Notes
50	V 350		Pseudo-orders
51	V 100		
110 → 100	A 0	0	Add next number to partial sum
101	H 200	0	
102	V 300	$C(200) \cdot C(300)$	
103	T 0	0	
104	A 101	H 200	Modify orders in 101 and 102
105	A 2	H 201	
106	U 101	H 201	
107	M 51	V 301	
108	U 102	V 301	
109	S 50	$-99 \cdot 2^{-15}$	Test for end
110	G 100 π	0	

Note: The M-order cannot easily be used to decrease the address in an order in the accumulator; in this example the address in the H-order is lower than that in the V-order, and the M-order is therefore used to form the V-order from the H-order, and not vice versa.

Exercises D

1. Given the number $n \cdot 2^{-15}$ in location 4, clear location n.

2. Locations 300 through 399 contain a series of numbers $a_0, a_1, \ldots, a_{99}$, respectively. Given the number $n \cdot 2^{-15}$ in location 4, replace a_n by its modulus.

3. The quantity y is defined by the infinite series

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots$$

The number x is given in 0, and n, the number of terms necessary to produce the required degree of precision, is given by the number $n \cdot 2^{-15}$ in 4. The coefficients $a_0, a_1, \ldots$ are in 150, 151, 152, . . . Place y in 6.

4. Modify your program for the last exercise so that the coefficients are assumed to be in m, $m + 1$, $m + 2$, . . . , where m is defined by the number $m \cdot 2^{-15}$ in 5, instead of in 150, 151, . . .

CHAPTER 2

SUBROUTINES

2–1 Introduction. Since an automatic computing machine can perform only a very limited number of basic operations, a mathematical calculation usually requires an extended sequence of orders. It is often convenient to break up this sequence of orders into self-contained groups of orders called *subroutines.* Each subroutine is responsible for a specific distinct part of the calculation, such as the evaluation of a sine or a cosine, or the extraction of a cube root. In simple calculations the subroutines can be placed end to end, with perhaps a few additional orders introduced for the purpose of connecting them, but in more complicated problems it is better to make use of a *master routine* as described in Section 1–9.

The initial construction, the testing, and (if necessary) the correcting of a program are all facilitated by the use of a master routine and subroutines. Further, the same subroutines can often be used over again in other programs, and the labor of drawing up a program for a particular problem is often greatly reduced if standard subroutines for performing the more common computing operations are available. If it is intended that an electronic computing machine shall be used on a wide variety of problems, it is, therefore, worth while to spend some effort on the establishment of an adequate library of such subroutines. Many subroutines are, however, constructed for particular problems, and it is not intended that they should be kept permanently in a library. Most of what is said in this chapter applies to subroutines in general, but attention is also given to the special requirements of subroutines intended for incorporation in a library. A master routine is concerned with the over-all organization of the problem, but it is prepared in the same way as a subroutine.

2–2 Relative numbering of addresses. To illustrate the techniques used in constructing subroutines, we will take as an example a subroutine for replacing the number in storage location 4 by its modulus. In practice, this is much too simple an operation to be dealt with by a subroutine and, indeed, some machines include in their order codes a special order for taking a modulus. A sequence of orders, starting in location 100, for replacing the number in 4 by its modulus has already been given in Example 3, Section 1–11. It is as follows:

Location	Content	Notes
100	S 4	
101	G 103 π	$\vert C(4)\vert$ to 4
102	T 4	

These orders are not, however, written in a sufficiently general form to be very useful in practice as a subroutine, since the address in the second order depends on the first order of the group being placed in location 100, and it might be required to place it somewhere else. In general, if the first order of the group were in location θ, the orders required would be

Location	Content
θ	S 4
$\theta + 1$	G $(\theta + 3)$ π
$\theta + 2$	T 4

Examination of other programs, for example those given in Examples 4 through 7, will show that it very often happens that the addresses in some of the orders depend on where the program as a whole is placed in the store. In the system to be described, subroutines are written with addresses specified relative to an arbitrary starting point, and the necessary adjustments are made when the subroutines are read into the store. Such addresses are called *relative* addresses to distinguish them from true or *absolute* addresses.

2–3 Internal and external forms of orders. The form in which orders are written and punched on the input tape differs from the form they take in the store of the machine. The written and punched form is called the *external* form, and the form inside the machine the *internal* form. Conversion from the external to the internal form takes place automatically, in a way which will be described later, when the input tape is read by the machine.

So far we have written orders in a form which corresponds directly to the binary form which they take inside the machine, and which may be regarded as being a shorthand notation for that form. This internal form (as written) consists of a function letter (followed by an S if the B-digit is 1), an absolute address, and a π if the special indication digit is 1. The external form differs in that the address can be either absolute or relative, and that there is a *terminal code letter* at the end.

The terminal code letter F is used to indicate that the address specified in an order is an absolute address, and the terminal code letter θ is used to indicate that the address specified is relative to that of the first order of a group of orders. The use of θ may be illustrated by showing how a subroutine for taking a modulus would be punched on the tape.

Example 10. $|C(4)|$ to 4.

Location	Content	Tape entry
θ	$S \quad 4$	$S\ 4\ \ F$
$\theta + 1$	$G\ (\theta + 3)\ \pi$	$G\ 3\ \pi\theta$
$\theta + 2$	$T \quad 4$	$T\ 4\ \ F$

An indication must be given to the machine of the address from which the relative numbering, indicated by terminal code letter θ, starts. This is done by punching the letters $G\ K$ as a marker symbol, or *control combination,* at the head of the group of orders in which this value of θ is to be used. The tape is thus punched as follows:

$$\begin{array}{lll} G & & K \\ S & 4 & F \\ G & 3 & \pi\theta \\ T & 4 & F \end{array}$$

When the control combination $G\ K$ is read, the machine makes a note of the number of the storage location into which the next order read from the tape ($S\ 4\ F$) is to be placed. Subsequently, when it encounters an order terminated by θ, it adds this number to the address specified in that order before the order is placed in its final location in the store. Note that the symbols $G\ K$ punched on the tape merely serve to mark the beginning of a subroutine, and do not correspond to anything in the program when it is placed in the store.

Terminal code letters serve two purposes. They act as markers on the input tape to indicate the end of an order, and they indicate whether the addresses are relative or absolute. All orders punched on the tape must have a terminal code letter. In the case of a B-order, the letter S can act as a terminal code letter and has the special significance described in Section 1–13.

Example 11. Storage locations 200 through 249 contain a series of positive numbers. To form, and place in 0, the sum of the cubes of those numbers which are greater than or equal to $1000 \cdot 2^{-15}$.

Location	Content	Tape entry	Notes
		G K	
$\theta + 0$	T 0	T 0 F	Clear 0 initially
$\theta + 1$	B 50	B 50 F	Place 50 in B-register
$\theta + 15 \rightarrow \theta + 2$	BS 1S	BS 1 S	Subtract 1 from number in B-register
$\theta + 3$	AS 200	AS 200 F	Place selected number, a_r, into the accumulator
$\theta + 4$	S $(\theta + 7)$	S 7 θ	Subtract $1000 \cdot 2^{-15}$
$\theta + 5$	G $(\theta + 15)$ π	G 15 $\pi\theta$	Jump if $a_r < 1000 \cdot 2^{-15}$
$\theta + 6$	E $(\theta + 9)$ π	E 9 $\pi\theta$	Jump and clear
$\theta + 7$	$1000 \cdot 2^{-15}$	‖ P 1000 F	Pseudo-order
$\theta + 8$	$a_r{}^2$	‖ (P F)	Temporary storage
$\theta + 6 \rightarrow \theta + 9$	HS 200	HS 200 F	Form $a_r{}^2$ and store it in $(\theta + 8)$
$\theta + 10$	VS 200	VS 200 F	
$\theta + 11$	T $(\theta + 8)$	T 8 θ	
$\theta + 12$	V $(\theta + 8)$	V 8 θ	Form $a_r{}^3$ in accumulator
$\theta + 13$	A 0	A 0 F	Accumulate in 0
$\theta + 14$	T 0	T 0 F	
$\theta + 5 \rightarrow \theta + 15$	J $(\theta + 2)$	J 2 θ	Jump to $(\theta + 2)$ unless B-register contains zero

Notes: (1) $(\theta + 8)$ is used for temporary storage. Something in the form of an order or pseudo-order to be placed in this location must be punched on the tape, otherwise the order HS 200, represented by the next tape entry, would be placed there. What is placed there is irrelevant, since it is overwritten by the order in $\theta + 11$ before being used. In the above, P F is shown as the tape entry, but any other order or pseudo-order would do equally well.

(2) In the written program it is convenient to indicate orders or pseudo-orders which are altered during the course of the calculation by enclosing them in brackets; the brackets are only for the assistance of the programmer and are not punched. Similarly, the vertical lines conventionally used to distinguish pseudo-orders are not punched.

Programming is normally done in terms of the external form of orders.

2–4 Reading of orders from the input tape. This is quite an elaborate operation. It includes the conversion of the decimal form of the address as punched to the binary form in which it is stored, and the conversion of relative addresses to absolute addresses. The reader might assume that the machine is provided with a complicated input unit designed to perform these functions. In fact, as will be seen later, the input circuits are very simple, and the functions just mentioned are performed by a special subroutine known as the *initial input routine.* Full information about the initial input routine will be found in Chapter 3, but it is not necessary for the reader to master that chapter in order to be able to write programs. All that is necessary is that he should understand clearly the nature and significance of the translation process which takes place when orders are read from the tape and converted from their external to their internal form; it is not essential that he should understand the details of the method by which the process is carried out. It will, however, be convenient, later in this chapter, to make use of a piece of nomenclature from Chapter 3. The individual rows of holes specifying an order are read from the tape one by one, and the information they contain is assembled in the accumulator to form the complete order. This order is then transferred to the position it is destined to occupy in the store as part of the program. It is convenient to have a name for the order (in fact, an ordinary T-order) in the initial input routine which affects this transference, and it will be referred to as "the Transfer Order."

The control combination $G\ K$ causes the initial input routine to make a note of the address currently specified in the Transfer Order; this it does by placing a copy of that address in storage location 42. Subsequently all addresses terminated by θ have $C(42)$ added to them.

It is important to remember that in the machine the addresses in all orders are absolute addresses and that there are no equivalents of the terminating code letters F or θ. These, and the control combination $G\ K$, occur on the input tape only.

2-5 Open and closed subroutines. A subroutine such as that given in Example 11 (Section 2-3) is entered immediately after the execution of the order coming before it in the store and, when the subroutine has done its work, control passes to the order immediately following it. Such a subroutine is called an *open* subroutine.

Another kind of subroutine is called a *closed* subroutine. This can be placed anywhere in the store, and is entered by a jump order and left by another jump order. The second jump order is referred to as the *link order.* This separation of the subroutine from the rest of the program often facilitates the construction of programs and their subsequent testing and, if necessary, modification. It also enables the same subroutine to be used at several points in the master routine.

2-6 Entering and leaving a closed subroutine. There are two alternative standard procedures for entering a closed subroutine. Either of these enables the machine automatically to form the link order required for returning control to the proper point in the master routine; formation of the link order takes place as part of the operation of the subroutine. All closed subroutines in the EDSAC library are drafted on the assumption that they will be entered in one of these two ways. The simplest method to understand—and the one now favored—makes use of the B-register, and subroutines designed for this method of entry are called *closed B* subroutines. The other type are known as *closed A* subroutines, and were exclusively used before the B-register was fitted to the machine; the library still contains many subroutines of this type.

2–7 Closed *B* subroutines. Let q be the address of the first order of a closed B subroutine. The subroutine is then entered as follows:

Location	Content	Notes
p	$B\ p$	Record p in B-register
$p + 1$	$F\ q$	Transfer control to subroutine

The address p in the order $B\ p$ is the address of the storage location in which that order itself is placed (the corresponding tape entry on the input tape is $B\ p\ F$ or $B\ p\ \theta$, according as p is an absolute address or an address relative to the beginning of this group of orders). The machine then enters the subroutine—by means of the jump order in $(p + 1)$—with information in the B-register regarding the point in the program from which the subroutine was entered.

If the content of the B-register is not disturbed during the action of the subroutine, the link order can consist simply of the order $FS\ 2$ placed at the end of the subroutine. Since the B-register contains the number p, this order will transfer control to $p + 2$, that is, to a point in the master routine immediately after that at which control was sent to the subroutine. Usually, however, the writer of the subroutine will wish to make use of the B-register within the subroutine itself, for the purpose of modifying orders and for counting. He can do this if he makes use of the following order which enables the content of the B-register to be stored, and subsequently replaced:

$K\ m\ F$	Place the order $B\ b$ in storage location m, where b is the content of the B-register.

Note that the order placed in storage location m is one which will, when executed, reinstate the original content of the B-register, namely b.

A closed B subroutine can now be constructed as follows:

Location	Content	Tape entry	Notes
$\theta + 0$	$K\ (\theta + r)$	$K\ \ r\ \ \theta$	Place $B\ \ p$ in $(\theta + r)$
$\theta + 1$	—	—	
·	·	·	Orders for the calculation carried out by the subroutine
·	·	·	
·	·	·	
$\theta + r - 1$	—	—	
$\theta + r$	$B\ \ p$	$(Z\ \ \ F)$	$B\ \ b$ planted by order in $(\theta + 0)$
$\theta + r + 1$	$FS\ 2$	$FS\ 2\ F$	Link order

Note: The order $Z\ \ F$ shown as punched on the tape to go into $\theta + r$ is overwritten before it is reached, by the action of the order in $\theta + 0$. As pointed out in Example 11, Section 2–3 [Note (1)], something must be punched on the tape to go into $\theta + r$; the advantage of punching $Z\ \ F$ is that if, as a result of some error on the part of the programmer, the order $Z\ \ F$ does not get overwritten, the machine will stop at once. This could happen if the subroutine were not called in correctly.

It will be noted that a closed subroutine can be called in from any part of the program, without restriction. In particular, one subroutine can call in another subroutine. Thus a division subroutine, for example, can be used both by the master routine and by any subroutine which requires it.

*2–8 Closed *A* subroutines.

A closed *A* subroutine may be entered in the following manner (q is the address of the first order of the subroutine):

Location	Content	C(Acc)	Notes
		Zero	
p	$A\ \ p$	$A\ \ p$	Place $A\ \ p$ in accumulator
$p + 1$	$F\ \ q$	$A\ \ p$	Transfer control to subroutine

The address p in the order $A\ \ p$ is the address of the storage location in which that order itself is placed (the corresponding tape entry on the input tape is $A\ \ p\ \ F$ or $A\ \ p\ \ \theta$, according as p is an absolute or relative address).

This time, when the machine enters the subroutine after obeying the jump order in $(p + 1)$, the accumulator contains information, in the form of the address in the order $A\ \ p$, regarding the point in the program from which the subroutine was entered, and from this a link order can be constructed. The link order is conventionally $E\ \ p + 2$. $F\ \ p + 2$ would be

* This section may be omitted on first reading.

better, but the F-order did not exist when the system now being described was designed; a limitation imposed by the use of the E-order is that C(Acc) must be positive or zero when control is returned to the master routine. The link order is produced by adding to the order $A\ p$ in the accumulator the pseudo-order $U\ 2$ [see Section 1–15, Example 8]. Since that pseudo-order is required by all closed A subroutines, there is a convention that it should be kept permanently in storage location 3, where it is placed by the initial input routine.

A closed A subroutine is therefore constructed as follows:

Location	Content	C(Acc)	Tape entry	Notes
			$G\quad K$	
		$A\ p$		On entry to subroutine
$\theta + 0$	$A\ 3$	$E\ (p + 2)$	$A\ 3\ F$	$U\ 2$ added to $A\ p$ to give $E\ (p + 2)$ (link order)
$\theta + 1$	$T\ (\theta + r)$	0	$T\ r\ \theta$	Plant link order in $(\theta + r)$
$\theta + 2$	—	—		
·	·	·		Orders for the calculation carried out by the subroutine
·	·	·		
·	·	·		
$\theta + r - 1$	—	0		
$\theta + r$	$E\ (p + 2)$		$(Z\quad F)$	Link order, planted by order in $(\theta + 1)$

Note: Closed A subroutines are normally designed to leave the accumulator clear.

2–9 Use of library subroutines. In order to be able to use a library subroutine, it is not necessary to have an exact knowledge of how it is constructed or of the precise numerical process which it uses, provided that there is available a concise specification of what it does and how it is to be used. Given a library of subroutines, it is possible to program with very little trouble some relatively elaborate calculations. All that is necessary is to write a *master routine* to which control is sent at the start of the program and which directs control to the various subroutines in turn.

EXAMPLE 12. To read a number $x < 0$, calculate and punch e^x.

Suppose we have, placed in advance in the store, three subroutines with the following specifications:

R Reads one number from the tape and places it in the accumulator. (Starts at address 100.)

E If $C(R) = x(<0)$, this subroutine places e^x in the accumulator. (Starts at address 150.)

P Punches $C(\text{Acc})$ on the output tape. (Starts at address 200.)

A master routine for this calculation is as follows:

Location	Content	Tape entry	Notes
(relative address)		G K	
0	B θ	B 0 θ	Call in R; x to accumulator
1	F 100	F 100 F	
$R \rightarrow 2$	T 0	T 0 F	x to multiplier register ready for E
3	H 0	H 0 F	
4	B $(4+\theta)$	B 4 θ	Call in E; places e^x in accumulator
5	F 150	F 150 F	
$E \rightarrow 6$	B $(6+\theta)$	B 6 θ	Call in P; punches $C(\text{Acc}) = e^x$
7	F 200	F 200 F	
$P \rightarrow 8$	Z 0	Z F	Stop

Note. In the written form of EDSAC programs, orders which transfer control unconditionally are usually underlined, as in the above example.

Exercises E

Construct master routines tp carry out the following operations. You may assume that these subroutines are available in the store:

S which replaces $C(\text{Acc})$ by its square root. (Starts in 70.)

T which replaces $C(\text{Acc})$ by $\frac{1}{2}\cos 180 \cdot C(\text{Acc})^\circ$. (Starts in 110.)

D which replaces $C(\text{Acc})$ by $C(\text{Acc})/C(R)$. (Starts in 160.)

R which replaces $C(\text{Acc})$ by a number read in decimal form from the input tape. (Starts in 200.)

P which punches $C(\text{Acc})$ and destroys it. (Starts in 250.)

The master routine cannot be placed in any location less than 320. The subroutines are all of the closed *B* type, and are liable to change the content of the multiplier register and the contents of locations 0, 1, 4, 5, 6, and 7.

Numbers to be read by subroutine *R* should be punched as decimal fractions, with the decimal point not punched but assumed to be at the left-hand end, and with a + or — sign after the numbers. For example,

$+0.8571$ should be punched as $8571+$

and $-\frac{1}{2}$ should be punched as $5-$

The number of decimal digits punched must not exceed 11.

1. Print the reciprocals of the numbers 2 through 10.

2. Print the values of the function $\sqrt{n/(n+1)}$ for $n = 0, 1, \ldots, 10$.

3. Print the inverse factorials $1/(n!)$ of the numbers $n = 2$ through 10, and their sum.

4. Read a number from the input tape and print its square.

5. Read a sequence of 20 numbers from the input tape and print the sum of their squares.

6. Read a sequence of numbers, all less than $\frac{1}{2}$ in modulus, and print the sum of their squares. The last number of the sequence is indicated by being followed by a number greater than $\frac{1}{2}$.

7. Read a sequence of numbers, as in exercise 6, and print the mean of their squares and the square of their mean. (It may be assumed that the sum of the numbers and the sum of their squares are both within capacity.)

8. Read two positive numbers, x and y, from the input tape, and print $\frac{1}{2}(x + y)$ and $\sqrt{xy}$. Assume that $x + y < 1$.

9. Read two positive numbers, x and y, from the input tape into two storage locations. Replace them in these locations by $\frac{1}{2}(x + y)$ and $\sqrt{xy}$. Carry out this replacement repeatedly until the two numbers differ by not more than 2^{-15}, then print either. Assume that $x + y < 1$.

10. Read a pair of numbers (r, ϕ) from the tape and print $r \cdot \cos 180\phi°$ and $r \cdot \sin 180\phi°$.

11. Solve the equation $x = 100 \cos x°$, by the following method. Put a first estimate (say π) of $x/100$ in a storage location. Replace it by the mean value of $x/100$ and $\cos x°$. (Beware of overflow!) Repeat until the new value differs from the old by not more than 2^{-15}, then print it.

12. Read 17 numbers x_n from the tape and print

$$\sum_{n=0}^{16} x_n \cdot \cos\,(180n/16)°.$$

13. Read three numbers a, b and C, where a and b are the lengths of two sides of a triangle and $100C$ is the included angle in degrees. (You may place before these numbers on the tape any constants that you require.) Print the length of the third side.

14. Read a series of positive numbers followed by a negative number. The positive numbers are the times in minutes, divided by 100, taken by a doctor to deal with successive patients, who arrive promptly and regularly every five minutes. The first patient is dealt with immediately; compute the time for which each succeeding patient must wait, and print the greatest of these times.

2–10 Long numbers. The EDSAC has a facility which enables an even-numbered storage location and the following odd-numbered storage location to be used as a single storage location holding 35 binary digits. Such a double-length storage location is known as a "long" storage location, and its content is known as a *long number*. An order referring to a long number is distinguished by having a 1 in the special indication position (see Section 1–7). This is indicated in the written form of the order by writing π after the address. For example, A 10 π means add into the accumulator the content of the long location formed by combining the short locations 10 and 11. Note that the significance of π (and of a 1 in the special indication position) in arithmetic orders is different from its significance in jump orders.

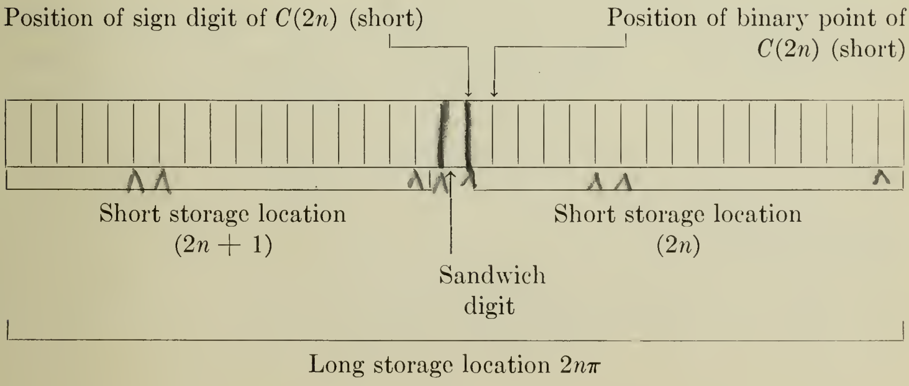
π does not act as a terminal code letter and, in the external form of orders as punched, must be followed by one of the regular terminal code letters. For example, we could have A 10 πF or A 10 $\pi\theta$, according as 10 indicates an absolute or relative address. πF is not often written, since there exists an abbreviation for it which is recognized by the initial input routine. At the end of an order

$$D \equiv \pi F.$$

Thus we would write A 10 D for A 10 πF.

The long location $2n$ may be referred to as $2n\pi$ and its content as $C(2n\pi)$. Alternatively, it may be referred to as $2nD$ and its content as $C(2nD)$. For distinctiveness, the short location $2n$ may be referred to as $2nF$.

In a long number, the extreme left-hand digit is used as a sign digit (referring to the long number as a whole), and the binary point is conventionally taken as immediately following it, as in short numbers occupying single storage locations. Thus in a long number a unit in the least significant digital position represents 2^{-34}. The content of a long storage location relative to the two short storage locations of which it is formed is indicated by the following diagram.


Position of sign digit of $C(2n)$ (short)
Position of binary point of $C(2n)$ (short)
Short storage location $(2n + 1)$
Short storage location $(2n)$
Sandwich digit
Long storage location $2n\pi$

There is one digital position between the end of the short location $(2n + 1)$ and the short location $2n$; the digit in this position is known as the "sandwich digit." No attention need be given to it except when forming a long number from two short numbers.

The multiplier register of the arithmetical unit is of sufficient capacity to hold a long number, and the accumulator is of sufficient capacity to hold the complete (69 binary digit) product of two long numbers.

In some calculations, long numbers may not provide sufficient precision. In such cases, the programmer may make use of what is known as *double-length* or *double-precision* working, in which two long storage locations are used to hold the digits of a single number. For example, a number x with up to 69 binary digits (including the sign digit) may be expressed as

$$x = a + 2^{-34}b,$$

where a and b are ordinary long numbers. The order code of the machine does not enable such numbers to be handled directly, but operations on them may be expressed in terms of operations on their single-length components, and programmed in the ordinary way. Similarly, greater precision still may be achieved by multi-length working, in which three or more long storage locations are used to hold the digits of a single number.

2-11 Some further orders in the order code.

(a) *Shifting.* With numbers expressed in binary form, as they are in the machine, a left shift of p places is equivalent to multiplying by 2^p, and a right shift of p places is equivalent to dividing by 2^p.

A shift is carried out on an operand already in the accumulator, and the result remains in the accumulator, so that in a shift order there is no reference to a storage location; the digits to the right of the function digits can therefore be used to specify the amount of the shift. The external forms of the shift orders are:

$L\ (\frac{1}{4} \cdot 2^p)\ F$	Left-shift $C(\text{Acc})$ by p places $(2 \leq p \leq 11)$ (i.e., multiply $C(\text{Acc})$ by 2^p),
$R\ (\frac{1}{4} \cdot 2^p)\ F$	Right-shift $C(\text{Acc})$ by p places $(2 \leq p \leq 11)$ (i.e., divide $C(\text{Acc})$ by 2^p);

for single shifts:

$L \quad D$	Multiply $C(\text{Acc})$ by 2,
$R \quad D$	Divide $C(\text{Acc})$ by 2.

Example 13. To multiply $C(4)$ by 10 (a process required in binary-decimal conversion).

The number 10 lies outside the range $-1 \leq x < 1$, and cannot therefore be stored in the machine or placed in the multiplier register. Multiplication by 10 can, however, be effected by first multiplying by 10/16 and then shifting the result four places to the left, that is, multiplying it by 16. Suppose $C(6) = 10/16$, and let $C(4) = x$. Then the orders required are:

Order	C(Acc)	Notes
$H\ 6\ F$		10/16 to multiplier register
$V\ 4\ F$	$(10/16)x$	
$L\ 4\ F$	$10x$	
$T\ 4\ F$	0	$10x$ to 4

(b) *Rounding-off.* To round off a number in the accumulator to 34 binary places it is necessary to add 2^{-35}. A special order is provided for this, since the smallest number which can be held in a long storage location is 2^{-34}. The special order is as follows:

$Y \quad F$	Round off C(Acc) to 34 binary places.

Note that the terminal code letter must be F.

(c) *Overflow indication.* It is sometimes convenient intentionally to program a calculation so that the machine may, at some stage, carry out operations of which the correct result would exceed the capacity of the accumulator. But if this is not intended, then it is very useful to have some indication of when such an overflow occurs, either as the result of an error in programming or as a consequence of numbers exceeding the range of values they were expected to have when the program was drawn up. If such an indication is wanted, it is usually wanted before the numbers and orders used in the part of the calculation which led to the overflow have been lost by subsequent operations of the machine. An order for this purpose has the written form:

$\phi\ n\ F$ or $\phi\ n\ D$	Transfer C(Acc) to location n (or $n\pi$) and clear the accumulator. If an overflow has occurred since the accumulator was last cleared, stop the machine, light the "accumulator overflow" light on the control panel, and give audible warning; otherwise proceed to the next order.

The machine can be restarted manually after stopping as the result of a ϕ-order. The ϕ is then equivalent to a T-order supplemented by an indication of an overflow. Note that in the definition of this order, clearing of the accumulator is intended to cover only deliberate clearing by an order which makes the content of the accumulator zero, whatever its previous value; for example, clearing by a T-order or by an $E\ \pi$ order when a jump takes place. It does not cover, for example, the reduction of the content of the accumulator to zero by the addition of an equal and opposite number.

(d) *Jump if nonzero.* The following order is especially useful when numerical checks are being programmed (for an example see p. 77):

$F\ n\ D$	Jump to n if $C(\text{Acc}) \neq 0$

The EDSAC order code contains a few futher orders which are not of interest to the general reader, although they are included in the complete order code given in Appendix 2.

2–12 Scale factors. Often some manipulation of the formulas occurring in a problem is necessary in order that all numbers to be evaluated by the machine may come within the range $-1 \leq x < 1$. This generally involves the introduction of scale factors.

Example 14. Given a division subroutine which replaces $C(\text{Acc})$ by $C(\text{Acc})/C(4D)$ [provided $|C(\text{Acc})| < |C(4D)|$], to evaluate $f = x^2/(1 + x^3)$, where x may lie anywhere in the range $0 < x < 3$.

x cannot be used or stored in the machine, since it goes outside the permitted range. However, $y = \frac{1}{4}x$ is always within range, and we therefore work with y instead of x. In terms of y, we have $f = 16y^2/(1 + 64y^3)$.

The maximum of f occurs when $x = 2^{1/3}$, and is less than unity. The condition of the subroutine $|C(\text{Acc})| < |C(4D)|$ can therefore always be satisfied, and the result will not exceed capacity.

In the expression $16y^2/(1 + 64y^3)$, both numerator and denominator are too large and need scaling down by a common factor. Their greatest values in the relevant range of y are 9 and 28, so that it is convenient to scale them both down by a factor of 32. This leads to the formula

$$f = \tfrac{1}{2}y^2/[1/32 + 2y^3]$$

as the formula to be evaluated. A program for doing this is given on the opposite page.

Let y be the initial content of $0D$, let $C(6F) = 1/32$, and let the division subroutine start at address 500.

Location	Tape entry	C(Acc)	Notes
(relative address)			
	G K		
0	H 0 D		y to multiplier register
1	V 0 D	y^2	
2	Y F		Round off
3	T 8 D	0	y^2 to 8D
4	V 8 D	y^3	
5	L D	$2y^3$	
6	A 6 F	$1/32 + 2y^3$	
7	Y F		
8	T 4 D	0	Set divisor
9	V 0 D	y^2	
10	R D	$\frac{1}{2}y^2$	
11	B 11 θ		Call in division subroutine to place $\frac{1}{2}y^2/(1/32 + 2y^3)$ in the accumulator
12	F 500 F		

Note: y^2 is computed a second time by multiplication rather than by adding in the value in 8D. If y is small, a more precise value of y^2 is thus obtained. Advantage can therefore be taken of the fact that the division subroutine uses as dividend the whole of the double-length accumulator.

2–13 Control combinations. One control combination has already been mentioned. This is the marker symbol G K which indicates the point from which relative numbering of orders is to start. Control combinations are also needed for other purposes, for example:

(1) To specify the storage location into which the first order of a subroutine punched on the tape shall be placed when the tape is read by the machine. When the first order has been placed in the location specified, succeeding orders will be placed in the following locations, and this procedure will continue until another control combination is encountered which breaks the sequence.

(2) To cause the machine to cease reading orders from the tape and to begin carrying out the orders it has placed in the store. A few of the more important control combinations are given below. All control combinations may be distinguished from orders by the fact that they terminate with the letters K or Z; this results in their being treated in a special way by the initial input routine. They are concerned solely with the operation of the initial input routine, and not with the performance of the calculation which has been programmed; they have served their purpose once the

orders are in the store and the calculations have started. The first control combination, *G K*, has already been given.

G K	Place a copy in storage location 42 of the address currently specified in the Transfer Order
T q K	Replace the address specified in the Transfer Order by *q*
E p K *P F*	Cease reading orders from the input tape, and transfer control to storage location *p* with the accumulator clear
P Z	This is used after a "leader" of blank tape at the beginning of a program tape, or in front of a subroutine, and acts as a warning to the initial input routine that the next entry on the tape will be a significant one. (See Chapter 4.)

The reader is now in a position to draw up, in a form suitable for punching on the tape, a complete program for a simple calculation.

EXAMPLE 15. To prepare the program tape for Example 12, including the placing of the subroutine in the store.

The master routine for this example has already been given in Section 2–9. The subroutines are to be placed with their first orders in locations 100, 150, and 200, and we will suppose that it has been decided to place the master routine with its first order in storage location 300. The tape is then punched as follows

P Z

T 100 *K*

$\boxed{R}$

T 150 *K*

$\boxed{E}$

T 200 *K*

$\boxed{P}$

T 300 *K*

$\boxed{\text{Master routine}}$

E 300 *K P F*

EXERCISES F

Prepare the control combinations and tape make-up for Exercises E.

2–14 Relative addresses in control combinations. When the address in the Transfer Order has to be reset it may sometimes be convenient to specify the value to which it is to be reset by a relative address rather than by an absolute address. This is indicated by using the letter Z to terminate the control combination, instead of K. The starting point from which the relative address is reckoned is that specified by the current content of 42 when the control combination is read. Thus $T\ q\ Z$ has the effect of $T\ (q\theta)\ K$. That is: place in the Transfer Order the address which, in relative form, is $q\theta$. It must not, however, be punched as $T\ q\theta\ K$, for then the θ would be interpreted as a terminal code letter.

In the above example, the control combination E 300 $K\ P\ F$ at the end of the master routine could be replaced by E 0 $Z\ P\ F$.

2–15 Extension of the use of relative addresses. So far, a system of relative numbering of addresses (indicated by the terminal code letter θ) has been used for cross reference between orders and pseudo-orders within a single group, the addresses within that group being specified relative to the beginning of the group. It is convenient to extend this use of relative addresses to provide cross referencing between different groups of orders whose final positions in the store may not have been decided on at the stage that has been reached in programming the calculation, and also for reference to numbers whose position in the store may be similarly undecided. This may be done by making use of code letters other than θ to indicate different starting points from which relative addresses may be reckoned.

EXAMPLE 16. a, b, c are the contents of h, $h + 2$, and $h + 4$, respectively, and $x \doteq C(n)$; to put $ax^2 + bx + c$ in 0.

It is assumed that this process forms part of a large program, and that it has not been found convenient at this stage to fix the locations into which the coefficients a, b, c, and x will finally go. It has, however, been decided that a, b, and c should occupy successive even-numbered storage locations. Although the addresses of these locations are undecided, it is necessary to refer to them when drawing up the program. A possible procedure would be to write them in algebraic form, with the letters h and n standing for numbers which have yet to be determined. If this were done it would be necessary to go through the whole program later and substitute for h and n the values which it had been decided to give them. This substitution process is, however, a systematic numerical procedure, and it can perfectly well be done by the machine itself. h and n are regarded as specifying starting points from which two separate systems of relative addresses are reckoned, and the addresses are indicated by terminating them with the code letters H and N, respectively. Conversion to absolute addresses takes place automatically, under the control of the initial input routine, when the tape is read.

Orders for the above example are shown in three forms:

(1) internal form with specified values of h and n (namely $h = 100$, $n = 60$),
(2) internal form with general addresses, and
(3) external form as punched on the tape.

Location	Internal form of orders		External form of orders
(relative address)	with specified addresses	with general addresses	(tape entries)
0	H 60	$H\ n$	$H \quad N$
1	V 100	$V\ h$	$V \quad H$
2	A 102	$A\ (h + 2)$	$A\ 2\ H$
3	T 0	T 0	$T \quad F$
4	V 0	V 0	$V \quad F$
5	A 104	$A\ (h + 4)$	$A\ 4\ H$
6	T 0	T 0	$T \quad F$

The order $A\ 2\ H$, for example, indicates that the address referred to is address 2 relative to the origin indicated by the terminal code letter H. It will be explained later how the starting points for the various systems of relative addresses, specified by the different code letters, are communicated to the machine.

Notes: (1) Address 0, absolute or relative, can be omitted in the written (and punched) external form of an order, as has been done in the last column of the above example.

(2) The storage location whose relative address is p, in the system specified by the terminal code letter H, is sometimes referred to as "location $p\ H$" and its content written as $C(pH)$, and similarly for other terminal code letters.

The following example illustrates the use of terminal code letters in cross references between a master routine and subroutines.

Example 17. To read a number x (<0) from the tape, and to evaluate and punch e^x, given the subroutines

R	which reads one number from tape and places it in 0π; starts at address h,
E	which places exp $[C(R)]$ in the accumulator; starts at address n,
P	which punches $C(\text{Acc})$ on the output tape; starts at address m.

These subroutines are similar to those in Example 12 but now refer to long numbers, and it is supposed that their positions in the store have not been determined when the master routine is drawn up. The master routine is given below in two forms, (1) as it is intended to appear in the store but without numerical values being given to the parameters θ, h, n, and m, and (2) as it is punched on the tape.

Location	Content	Tape entry	Notes
(relative address)			
		$G\quad K$	
0	$B\quad \theta$	$B\quad \theta$	Call in R; x to accumulator
1	$F\quad h$	$F\quad H$	
$R \rightarrow 2$	$T\quad 0\pi$	$T\quad D$	x to multiplier register
3	$H\quad 0\pi$	$H\quad D$	
4	$B\ (4 + \theta)$	$B\ 4\ \theta$	Call in E; places e^x in accumulator
5	$F\quad n$	$F\quad N$	
$E \rightarrow 6$	$B\ (6 + \theta)$	$B\ 6\ \theta$	Call in P; punches e^x on output tape stop
7	$F\quad m$	$F\quad M$	
$P \rightarrow 8$	Z	$Z\quad F$	

This use of relative addresses may seem a complication in a short program for a simple calculation such as this. But it is a very great simplification in long or involved programs because it enables the different parts of the program to be largely "uncoupled" from one another while the program is being drawn up, and yet enables freedom of cross-referencing between them to be fully maintained. Decisions as to where the various parts are to be ultimately placed in the store can be considered separately, and later, when it is known exactly how long each one will be.

2–16 Setting of the constants to be added by terminal code letters. It has already been explained that the code letter θ, when used to terminate an order, causes the content of location 42 to be added to the address punched on the tape. Similarly, the code letters H, N, . . . cause the contents of storage locations 45, 46, . . . to be added. Whereas, however, the content of 42 is set automatically by the control combination $G\ K$, the contents of 45 and 46 have to be set explicitly by entries punched on the tape for that purpose. Suppose, for example, that it is desired to perform the calculation of Example 16 with the parameters h and n set equal to

100 and 60, respectively. Suppose, further, that it is decided to put the program into the store with its first order in location 200. The program tape is then punched as follows

Tape entry	Notes
T 45 *K*	Set address in Transfer Order equal to 45
P 100 *F*	Pseudo-order read into 45
P 60 *F*	Pseudo-order read into 46
T 200 *K*	Set address in Transfer Order equal to 200
G *K*	
H *N*	Orders of program; these go into 200, 201, . . .
V *H*	
A 2 *H*	
T *F*	
V *F*	
A 4 *H*	
T *F*	
. . . .	

No control combination is shown punched at the end of the tape, since it is assumed that the calculation concerned forms part of a larger calculation, the orders for which follow those given.

Notes: (1) Planting of the pseudo-orders *P* 100 *F* and *P* 60 *F* in 45 and 46 is a matter concerned solely with the process of reading the program into the store, and has nothing to do with the process of performing the calculation once the orders have been read in.

(2) Once the program tape has been read the pseudo-orders in 45 and 46 are no longer required, and these locations may be used again for other purposes.

There is no reason why the same values should be set in 45, 46, etc., throughout the reading of the entire tape. It is perfectly permissible, for example, to set certain values during the reading of the master routine, and different values for the reading of each subroutine. Very often, as will be seen later, library subroutines make use of the facilities offered by the code letters *H*, *N*, etc., to enable the values of parameters to be set.

2–17 Complete table of terminal code letters. A complete table of the terminal code letters available for use by the programmer is as follows:

Code letter	Location whose content is added to the order	Content of locations given in Column 2, i.e., number added
F	41	Zero
θ	42	Variable, set by $G\ K$
D	43	2^{-16}
ϕ	44	Variable, set during input
H	45	
N	46	
M	47	
Δ	48	
L	49	
X	50	
G	51	
A	52	
B	53	
C	54	
V	55	

It will be observed that this table includes not only the terminal code letters whose functions we have just been considering, but also the code letters F, θ, D as well. The reason is that all code letters are treated by the initial input routine in a similar manner, the only difference being that different code letters cause the contents of different locations to be added. Location 41 (corresponding to F) always contains zero when a tape is being read, and location 43 (corresponding to D) always contains 2^{-16}; orders terminated by these code letters, therefore, have the special indication digit made equal to 0 or 1, respectively, when they are read, the addresses themselves being unmodified. Note that the symbol π also causes 2^{-16} to be added, but must be followed by another code letter which indicates the end of the order. It is thus possible, by using π and another code letter, to cause both 2^{-16} and some other number to be added to the order before it is put away in the store.

2–18 Parameters. For a subroutine to be useful for inclusion in a library it must be drafted in a sufficiently general form so that it can be used in different contexts without the user having to carry out internal modifications to it. It has already been shown how one important step in this direction can be taken by using the terminal code letter θ to make the external form of a group of orders independent of their positions in the store. A second step can be taken by drawing up subroutines with adjustable *parameters*. In the EDSAC library parameters are of two kinds, *preset* parameters and *program* parameters. The values of preset parameters are incorporated in the subroutine during the reading of orders from the input tape, and cannot be changed during the subsequent execution of the program. Program parameters, on the other hand, are given values when the subroutine is called in, and may be given different values on different occasions.

2–19 Preset parameters. Preset parameters are incorporated in library subroutines by making use of the facilities offered by the code letters H, N, etc.; the values of the parameters are placed in storage locations 45, 46, etc., and incorporated in the subroutine when the input tape is read. In some cases this is simply an application of the relative addressing facility previously discussed. For example, a subroutine may be designed to perform some operation on a sequence of numbers placed in storage locations 0 H onwards. In other cases the quantities placed in 45, or in one of the other locations, may not be the first of a sequence of storage locations at all, but may specify, for example, a scale factor. This use of a preset parameter is illustrated by library subroutine $E6$ which computes the value of exp $(2^p y)$, where y (less than 0) is the content of the multiplier register. p is given by a preset parameter which is set by placing the pseudo-order P p F in storage location 45 before the subroutine is read. The reader will be able to see from the program of this subroutine, given in Part 3, how the value of the parameter is incorporated in the subroutine.

It is to be noted that the whole of a pseudo-order placed in 45, and not only the address, is added to an order terminated by the code letter H, and similarly for other code letters. For example, if F n F is placed in 45, then a tape entry punched as P H becomes F n F. This fact is made use of in the case of certain subroutines, as an examination of Part 2 and Part 3 will show.

2–20 Program parameters. Program parameters for closed subroutines are conventionally punched in the form of pseudo-orders placed immediately after the pair of orders calling in the subroutine. When the subroutine has done its work it automatically returns control to the order in the program coming immediately after the last program parameter. The subroutine contains orders which pick up the pseudo-orders representing program parameters, and plant them where required within itself.

2–21 Standard procedure for setting preset parameters. Suppose the preset parameter corresponding to H has to be set to a certain value by placing the pseudo-order P 100 F in 45, and that the subroutine to which the parameter applies has to go with its first order in 500. This could be done by punching on the tape:

```
T  45 K
P 100 F
T 500 K
```

followed by the subroutine. A slightly different form of punching is, however, adopted for general use. This makes use of the following control combination:

T Z	Make the address specified in the Transfer Order equal to that stored in 42

The tape is punched as follows:

T 500 *K*	Set address specified in Transfer Order equal to 500
G *K*	Store address specified in Transfer Order (i.e., 500) in 42
T 45 *K*	Set address specified in Transfer Order equal to 45
P 100 *F*	Goes into 45
T *Z*	Copy address from 42 (i.e., 500) into Transfer Order

followed by the subroutine. This somewhat roundabout procedure has certain minor advantages which led to its adoption when the procedure for using library subroutine was being decided upon. All types of library subroutine using preset parameters have *T Z* or some equivalent control combination punched at their head.

2–22 Interpretive subroutines. The use of closed subroutines provides the programmer with facilities additional to those which are built into the machine and covered by the basic order code. Each subroutine provides one such facility, and the pair of orders required for calling it in may be regarded as equivalent to the single order which would be required if the facility were covered by the basic order code. *Interpretive subroutines* work on a slightly different principle, and enable the programmer to write the whole, or part, of his program in an order code which may be quite different from the basic order code of the machine. As an illustration, an interpretive subroutine will be given which makes it possible to run on the EDSAC a program for manipulating complex numbers written in terms of the orders given below. These orders will be called *interpretive orders*. They do not enter the control circuits of the machine but are extracted from the program, one by one, by the interpretive subroutine, which examines them, and carries out the appropriate operations.

It will be assumed that the real and imaginary parts of a complex number are stored in consecutive long storage locations; thus the complex number in nD and $(n + 2)D$ means a complex number whose real part is in nD and whose imaginary part is in $(n + 2)D$. Two long locations $6D$ and $8D$ are set aside to hold results; they will be referred to as the "complex accumulator," and play a role similar to that played by the accumulator in ordinary machine operation. The interpretive orders are as follows:

$A\ n\ D$	Add complex number in nD and $(n + 2)D$ into complex accumulator
$S\ n\ D$	Subtract complex number in nD and $(n + 2)D$ from complex accumulator
$U\ n\ D$	Transfer contents of complex accumulator to nD and $(n + 2)D$
$T\ n\ D$	Transfer contents of complex accumulator to nD and $(n + 2)D$ and clear complex accumulator
$V\ n\ D$	Multiply complex number in nD and $(n + 2)D$ by $C(R)$ and add into the complex accumulator [$C(R)$ is real]

When the programmer reaches the point in the master routine at which he wishes to use interpretive orders, he writes down a pair of orders for calling in the interpretive subroutine, similar to those he would use for calling in a closed B subroutine. He then writes a sequence of interpretive orders for performing the operations he requires. When he wishes to return to ordinary coding in terms of machine orders, he writes the interpretive order $F\ n\ F$, which causes control to be transferred from the interpretive subroutine to storage location n.

The orders of the interpretive subroutine are given below. The notes are written on the assumption that the subroutine has been called in by entries in the master routine as shown, and that the first of the interpretive orders, namely $A\ 100\ D$, is being interpreted.

Master routine:

p	$B\ \ p\ F$
$p + 1$	$F\ \ \ N$
$p + 2$	$A\ 100\ D$
$p + 3$	. . . . .

Subroutine:

```
              |  G      K  |
 11 →   0N    |  AS   2 F  |   Select next interpretive order (A 100 D)
              |            |     from master routine
         1    |  U    5 θ  |   Plant A 100 D in 5θ
         2    |  A   12 θ  |  ] Form A 102 D and plant in 8θ
         3    |  T    8 θ  |  ]
         4    |  A    6 D  |                       ] Add real part into
         5    |  (Z     F) |   Becomes A 100 D     ]   complex accumu-
         6    |  T    6 D  |                       ]   lator
              |            |
         7    |  A    8 D  |                       ] Add imaginary part
         8    |  (Z     F) |   Becomes A 102 D     ]   into complex accu-
         9    |  T    8 D  |                       ]   mulator
        10    |  BS   1 F  |  ] Increase number in B-register by 1 and
        11    |  F      θ  |  ]   jump to select next interpretive order
              |------------|
        12    | ||P   2 F  |
```

An example of an interpretive subroutine for performing a wider range of operations with complex numbers is given in Part 3. Interpretive subroutines are also useful for performing operations with floating-binary or floating-decimal numbers.

Exercises G

Construct complete programs to carry out the following calculations. Make reasonable choices of the method to be used and the accuracy to be attained. Any subroutines listed in Part 2 may be used. Prepare every detail required for the punching of the tape and the running of the problem on the machine.

1. Print $\sum_{n=1}^{10} 1/n$ and $\sum_{n=1}^{10} 1/n^2$.

2. Print the values of $z = \operatorname{sech}^{-1} y$ for values of y from 0.1(0.1)0.9. Use the formula

$$z = \log_e \frac{1 - \sqrt{1 - y^2}}{y}.$$

3. Evaluate and print $\int_0^1 e^{-x^2}\,dx$.

4. Print the values of $\int_0^{\pi/3} \cos(x \cos\theta)\,d\theta$ for $x = 0(0.2)5$.

5. Read two numbers x and y, where $2^{-32} < x < 1$ and $2^{-32} < y < 1$, and print x^y.

6. Tabulate Airy's integral $Ai(-x)$ for $x = 0(0.2)5$ by solving the differential equation $d^2y/dx^2 + xy = 0$ with $y = 0.35503$, $dy/dx = 0.25882$ when $x = 0$.

7. A traffic census is taken using a tape punch as follows. Whenever a bicycle passes a B is punched, and whenever a motor vehicle passes a V is punched. Every minute an M is punched, except at every tenth minute, when a T is punched. At the end of the tape an E is punched. Rows of blank tape, and erase, carriage return, and line feed symbols may appear anywhere.

Prepare a program to process this tape as follows. (a) Check that apart from rows of blank tape and erase, carriage return, and line feed symbols, only the symbols B, V, M, T, E appear on the tape. (b) Check that exactly 9 M's intervene between consecutive T's. (c) Print the greatest number of bicycles that pass within any 15 consecutive minutes, and the greatest number of motor vehicles that pass within any 15 consecutive minutes.

CHAPTER 3

PROGRAMMING FOR OTHER MACHINES

3–1 Introduction. It is a general experience that anyone who is accustomed to writing programs for a particular digital computer has very little difficulty in learning to write programs for a different digital computer. This is in spite of the fact that different machines can have order codes of widely differing types. The reason is that the principles on which programs are constructed are the same for all machines, and that the expedients and devices appropriate to one order code are paralleled by corresponding expedients and devices in other order codes. A programmer faced with an unfamiliar machine does not, therefore, need to learn programming all over again; all he needs to do is to acquire experience with the new order code.

The order code of the machine reflects the underlying logical design of the machine itself. The designer of a machine may be predisposed towards a certain type of logical design, or he may decide that one type rather than another is indicated by his terms of reference. The logical design will then determine the general nature of the order code, and the designer will do his best, within the limitations thus imposed, to meet the requirements of the programmers. The result is that the order codes of machines fall into several fairly clearly defined classes.

Although, within each class, order codes are very similar in underlying structure, there can be such wide differences in the ways in which orders are written that it is not always easy to recognize this similarity when looking at program sheets. EDSAC programs, for example, would look entirely different if the numerical values of function letters and code letters were written, instead of the letters themselves, although such a difference would be a superficial one only. In some computing centers addresses are conventionally written in the scales of 8, 16, or 32, instead of in the decimal scale as is done for the EDSAC. If the scale of 16 is used it is necessary to have 6 extra symbols, over and above the ten figures 0 through 9, to represent the digits ten through fifteen; for example, the letters *a* through *f*, or *u* through *z*, may be used. In the scale of 32 it is necessary to have 32 separate characters, and the whole teleprinter alphabet, including $, @, and /, may be pressed into service. This is quite effective, but it must be admitted that the resulting program sheets often present a somewhat bizarre appearance.

Although the differences referred to in the last paragraph are superficial rather than fundamental, they are not, in our view, without importance. An arbitrary notation which is not easily memorized can act as a serious stumbling block to a programmer who approaches an unfamiliar machine. It would therefore appear to be desirable that, when a new type of machine is first put into action, great care should be taken to choose a satisfactory notation for orders. This is of special importance in the case of a machine which is likely to be used by many different people.

Most of the remarks which follow apply equally well to both binary and decimal machines. In the latter, since all large capacity stores presently in use are binary in nature, it is usual to represent decimal digits, in numbers, by means of a binary code, and it is natural that the same code should be used for addresses in orders. It may also be convenient to use the same code for the function part of an order, allocating (say) two decimal digits for the purpose; alternatively, the function part may be treated purely as a set of binary digits, in the same way as in a binary machine.

3–2 Single-address codes. The EDSAC is fairly typical of single-address machines whose order codes contain between 16 and 32 orders, although many machines also have an order for division. Apart from division, the order codes of other machines may contain orders which are individually slightly different from those in the EDSAC, but which, in the aggregate, provide very similar facilities. For example, in the EDSAC it is usual to clear the accumulator at the end of a sequence of operations, most often by a T- or a ϕ-order which transfers the result to the store, whereas many order codes provide for the accumulator to be cleared at the beginning of a sequence of operations by means of a *clear and add* order; this first clears the accumulator, and then causes the content of some storage location to be added in. Again, instead of the H- and V-orders in the EDSAC order code, some machines have an order which causes the number in the accumulator to be multiplied by the content of a given storage location, and the result to be left in the accumulator. It will readily be seen that these do not amount to large differences. Differences in the facilities provided for discrimination, that is, in the range and variety of conditional orders, are slightly more significant. If only one or two conditional orders are provided, the programmer must, if he wants to make his program as short as possible, plan ahead to make sure that all discriminations, when they arise, will be in the exact form required by one of the available conditional orders. If a wider variety of conditional orders is provided, the programmer is spared some of this careful planning.

Quite apart from any question of convenience for the programmer, the addition of new orders to the order code of a machine will have the effect of increasing slightly its average speed, since there are bound to be some programs which could be shortened by using a new order. Orders which enable some common operation to be performed more simply yield the greatest returns in this respect. In the case of the EDSAC, an example would be an order, similar to the one just mentioned, for multiplying the number in the accumulator by a number from the store. Orders for performing comparatively rare operations, such as the extraction of a cube root, have little effect on the speed of the majority of programs, although it is always possible to quote examples of situations in which they would give a substantial increase in speed. It might be noted that slight variants of orders already existing in a machine can often be added with little increase in the amount of equipment in the machine; however, the number of digits available in the order code to specify the function of an order may impose a strict limitation on what is possible in this respect.

If plenty of function digits are available or, perhaps one should say, if the designer of the machine chooses, as one of the innumerable decisions he must make in fixing the logical design of the machine, to make them available, it is possible to have a more direct relation between the individual function digits and the function of the order. For example, a certain function digit might be a 1 if access to the store were required, and a 0 otherwise, while another digit might indicate whether the arithmetical unit was to be set to perform an addition or a subtraction. An example of a machine designed on these principles is the machine built at the Institute for Advanced Study, Princeton, which has ten function digits in each order. Such machines provide the programmer with a large number of different orders to choose from, although not all the possible combinations of function digits correspond to valid or useful orders.

In most single-address machines, two orders are stored in a full word, as is done in the EDSAC. Facilities for working with either short or long numbers are not, however, usually provided, all numbers occupying a full word.

3–3 Multi-address codes. In the codes so far discussed, each order makes reference, at most, to one location in the store. It is possible to have order codes in which more than one such reference is made. Of these the commonest are three-address codes, in which each order makes reference to three locations in the store. In a typical three-address code an order for multiplication might be written as follows:

$$M \quad \alpha \quad \beta \quad \gamma \qquad \text{Form } C(\alpha)\cdot C(\beta) \text{ and place in } \gamma.$$

In a binary machine with a 35-digit word-length, the first five digits of a word representing an order might define the function of the order (in this case multiplication), the next ten digits might define the first address, the next ten digits the second address, and the remaining ten digits the third address. The order would thus be equivalent in the machine to the number

$$m \cdot 2^{-4} + \alpha \cdot 2^{-14} + \beta \cdot 2^{-24} + \gamma \cdot 2^{-34}.$$

It is assumed that the binary point comes immediately after the most significant digit.

In a machine with a three-address code, orders may be modified by performing arithmetic operations on them in a manner precisely similar to that used in a machine with a single-address code. For example, if we add to the order written above the pseudo-order

$$P \ 1 \ 1 \ 0,$$

which is equivalent to $2^{-14} + 2^{-24}$ (it being assumed that the numerical equivalent of P is zero), we obtain the order

$$M \ \alpha + 1 \ \beta + 1 \ \gamma.$$

Other examples of arithmetic and conditional orders in a three-address code are:

$A \ \alpha \ \beta \ \gamma$ Form $C(\alpha) + C(\beta)$ and place in γ,

$S \ \alpha \ \beta \ \gamma$ Form $C(\alpha) - C(\beta)$ and place in γ,

$R \ \alpha \ \beta \ \gamma$ Shift $C(\alpha)$ β digits to the right and place in γ,

$E \ \alpha \ \beta \ \gamma$ If $C(\alpha) \geq C(\beta)$ take next order from γ; otherwise proceed serially.

Other conditional orders [for example, one which transfers control if $C(\alpha) = C(\beta)$] and orders for performing logical operations can similarly be included in a three-address code. It will be noted that in a machine using this type of three-address code there is no accumulator or other provision for the storage of numbers in the arithmetical unit between the execution of orders.

EXAMPLE 18. To form $\sum_0^{19} a_r b_r$ and place in 0, where $a_r = C(10 + r)$ and $b_r = C(30 + r)$.

```
        100 │  S   0   0   0 │ Clear 0 for sum
        101 │  A   0 107 102 │ Set multiply order in 102
105 → 102 │ (              ) │ Form a_r b_r
        103 │  A   1   0   0 │ Add to sum in 0
        104 │  A 102 109 102 │ Increase r by 1
        105 │  E 108 102 102 │ Test for end
        106 │  Z   0   0   0 │ Stop
        107 ││ M  10  30   1 │
        108 ││ M  30  50   1 │
        109 ││ P   1   1   0 │
```

This example will suffice to show that the principles on which programs are constructed are exactly the same for a three-address code as for a single-address code.

One order in a three-address code takes up more space in the store than one order in a single-address code. If the EDSAC had been designed to work with a three-address code, one long storage location would be required to hold each order, instead of one short storage location. One order in a three-address code, however, causes a more complicated operation to be performed than one order in a single-address code; for example, the single order $A\ \ \alpha\ \ \beta\ \ \gamma$ has the same effect as the group of order $A\ \ \alpha$, $A\ \ \beta$, $T\ \ \gamma$, in the EDSAC order code, and would require one long storage location instead of three short ones. However, use of a three address code does not always enable a similar saving to be made; for example, to add the four numbers in storage locations α, β, γ, δ and to place the result in storage location ϵ, the following three orders are required:

$$\begin{array}{llll} A & \alpha & \beta & \epsilon \\ A & \epsilon & \gamma & \epsilon \\ A & \epsilon & \delta & \epsilon \end{array}$$

In the EDSAC, four A-orders and one T-order would be required, and would take up five short storage locations, instead of three long ones. Thus, in this case, the orders in the single-address code actually take up less space than those in the three address code, the reason being that when using the single-address code the programmer can take advantage of the fact that sums can be accumulated in the accumulator. Whether or not there is any over-all saving in the amount of storage required to hold the orders for a typical program in a given three-address code, compared with a given one-address code, depends on the relative extents of the facilities provided in the two codes. The remarks made below have some bearing on this matter. The difference, in any case, would not be very great.

3-4 Multiplication and division. A difficulty arises with the simplest type of three-address code, illustrated above, because the result obtained when two numbers are multiplied together is a double-length number, whereas the location to which this result is to be sent can accommodate only a single-length number. An exactly similar difficulty arises in connection with division, where there are a quotient and a remainder to be considered. One or two early machines were so constructed that the less significant half of the product and the remainder after division were not available to the programmer, but this is now recognized as imposing a severe limitation on the utility of a machine.

One solution to the difficulty is to provide two multiplication orders, one of which causes the more significant half of the product to be transmitted to the store, and another which causes the less significant half to be transmitted. This implies that when both halves of the product are required the multiplication must be performed twice, with resulting waste of time. In practice, in machines adopting this solution, a third multiplication order which transmits the product rounded-off to single length (rather than merely truncated) is also provided.

An alternative solution is to have an extra address in each order, making four addresses in all. In orders for multiplication, the extra address can be used to specify the location to which the less significant half of the product is to be transmitted, and in orders for division, to indicate the address to which the remainder is to be transmitted. The number of digits available in an order, however, would generally restrict the use of this method to machines with small high-speed stores. Yet another method, which has also been used in practice, is to arrange that an order for multiplication causes the more significant half of the product to be transmitted to γ, the location specified in the order, and the less significant half to $\gamma + 1$; a similar provision can be made in the case of orders for division.

An alternative line of approach, which has been followed in a number of later machines, is to provide storage in the arithmetical unit in which the less significant half of a product, or a remainder, can be held for subsequent transfer, if required, to the store. The minimum amount of storage required can be provided by a single register. If, however, an accumulator (preferably double-length) is used, it is possible also to provide facilities for accumulating sums and products. This does away with the clumsiness shown by the simpler forms of three-address order codes when a number of sums or products must be added. A double-length accumulator is a great convenience in any machine, since it enables a useful amount of double-length working to be achieved without extra complication; this is especially true if the machine is provided with *B*-registers or with three-address orders which enable counting and order modification to be done without clearing the accumulator. A double-length accumulator also facilitates division with a double-length dividend (cf. Section 5–4).

Three-address machines usually have an order which performs one of the logical operations (usually "and" or "or"), sometimes combined with a shift of the result. There is sometimes an order for taking a number from the store, shifting it a specified number of places, and planting it in an order to replace one of the addresses already there.

3–5 Source-destination codes. In any digital computer, all basic operations consist of the transfer of a number from one register or storage location to another register or storage location, possibly with the performance of some operation on the way or with the initiation of some action which is to take place after the transfer is complete. For example, in the EDSAC, the *H*-order causes a simple transfer of a number from a storage location to the multiplier register, while the *A*-order causes the transfer of a number from a storage location to the accumulator register via the adder. Similarly, an *O*-order causes the five most significant digits of a number in a storage location to be transferred to a special register associated with the output punch, and initiates the operation of punching a row of holes in the tape. The orders of machines with conventional single-address order codes conceal this source-destination character of the fundamental machine operations. Other machines, however, have orders whose structure shows it explicitly.

In the purest form of source-destination coding, addresses are assigned to all sources from which numbers may come and to all destinations to which they may be sent. Sources include storage locations, the accumulator, and input mechanisms. Destinations include storage locations, the multiplier register, the accumulator register, and output mechanisms. Some units of the machine have more than one destination associated with them; for example, a number sent to one destination may pass into the accumulator via an adding circuit, whereas if it is sent to another destination it may pass into the accumulator via a subtracting circuit. Addresses may also be given to sources supplying frequently wanted constants, such as the number zero and the number whose digits are all 1's. The following is a hypothetical example designed to illustrate source-destination coding. Addresses 0 through 7 are assigned in the manner indicated below, and will be referred to as *special addresses;* addresses greater than or equal to 8 refer to the store.

0	accumulator, via adder	(destination)
1	accumulator, via subtractor	(destination)
2	accumulator direct	(source or destination)

3	multiplier register	(source or destination)
4	*B*-register	(source or destination)
5	output	(destination)
6	input	(source)
7	number zero	(source)

Orders consist of a source followed by a destination; for example:

100,0	add $C(100)$ into accumulator
100,1	subtract $C(100)$ from accumulator
100,5	transfer $C(100)$ to output mechanism and print (or punch)
2,100	transfer $C(\text{Acc})$ to 100
2,3	transfer $C(\text{Acc})$ to multiplier register
7,4	clear *B*-register

An increased number of facilities may be provided, with a reduced number of addresses, if a few extra binary digits are added to each order. These digits, which may be regarded as rudimentary function digits, specify an operation which is to be performed on the number as it is transferred from the source to the destination; for example, they may specify that the sign of the number shall be changed, or that its modulus shall be taken. This type of order code was used in the Automatic Sequence Controlled Calculator (Harvard Mark I), which, as explained in Chapter 1, was an electromechanical machine in which the orders were stored on (wide) punched tape. This simple form of source-destination coding is not, however, very suitable for use in a machine in which the program is stored in an electronic store, because of the large number of digits required in each order in relation to the complexity of the operation performed. The reason so many digits are required is that it must be possible for either the source or the destination to be a location in the store, although it would not be a serious restriction if one of these were always one of the special sources or destinations whose addresses contain only a few binary digits. An improvement can, therefore, be obtained by having an order structure which allows for two addresses of unequal lengths, one (the short address) having just enough binary digits to specify the address of a special source or destination, and the other (the long address) having sufficient binary digits to specify a location in the main store; which is the source and which is the destination is indicated by means of an extra function digit. It is to be observed, however, that such an order structure differs in no material way from that used in a conventional single-address code, since the function digits and the short-address digits can be grouped together and regarded as specifying the function of the order, the operand being given by the long-address digits. Whether this or the original point of view should be taken is largely a matter of convenience.

Readers who are interested in the evolution of multi-address order codes are advised to study the order codes of machines such as SEAC, SWAC, DYSEAC, ERA 1103 (now Univac Scientific Computer), RAYDAC, NORC, and the Harvard machines. Information about some of these computers is given in publications listed in the Bibliography; information about the others will be found in literature issued by the manufacturers.

3–6 Representation of negative numbers. In many machines negative numbers are represented by complements. Sometimes, however, instead of true complements as used in the EDSAC, use is made of *ones complements* or, in a decimal machine, of *nines complements.* Ones complements or nines complements can be converted to true complements by adding one unit in the least significant place. When ones complements or nines complements are used, there are two forms for zero, namely, a number consisting entirely of 0's and a number consisting entirely of 1's or 9's, as the case may be. Occasionally the programmer may need to take account of the existence of these two forms, but in a modern, well-designed machine these occasions should be rare. The range in which numbers can lie is slightly more restricted in a machine using ones or nines complements than it would be in an otherwise similar machine using true complements, being, for example, $-1 < x < 1$ instead of $-1 \leq x < 1$.

In some machines all numbers are represented on a "sign and modulus" convention, similar to that used in ordinary writing. For example, in a binary machine one digit would be a sign digit and would indicate whether the number was positive or negative, while the remaining digits would specify the modulus of the number. In this system the sign digit behaves quite differently from other digits, whereas when complements are used the sign digit is treated, during addition and subtraction, in exactly the same way as the other digits, but acquires a special significance by reason of the conventions adopted. Two forms of zero, namely, $+0$ and -0, exist in this system also, but again, in a well-designed machine, no confusion is likely to result. As regards the range in which numbers can lie, a machine using the sign and modulus convention resembles one using ones or nines complements, the range being rather more restricted than in a machine using true complements.

Although it is desirable that a programmer should understand the system by which numbers are represented inside the machine, it is only on rather special occasions that he need pay much heed to it.

3–7 Miscellaneous facilities. It will be convenient to mention here a number of facilities which are provided in the order codes of various machines. They all have their value, but it is not to be expected that any one machine should provide them all.

Floating-point. Some machines automatically carry out operations on numbers expressed in floating-decimal or floating-binary form. In such machines numbers may be represented in the form $2^p \cdot a$ (or $10^p \cdot a$ in a decimal machine); usually a single word is used to hold the number, some of the digits being allocated to p and some to a. The programming of many calculations is much simpler than when fixed-point working is used, since scaling is rarely required.

Standardizing order. This order causes the number in the accumulator to be shifted to the left as far as is possible without causing an overflow; the number of places (binary or decimal) through which shifting has taken place is recorded, either in one of the registers of the arithmetical unit or in a location in the store. Standardizing orders are often found in machines which do not have built-in floating-point facilities, and their purpose, for which they are very effective, is to facilitate the programming of scaling operations and the construction of floating-point subroutines.

Multi-length working. An order is sometimes provided to facilitate the addition of numbers whose digits occupy more than one storage location. For example, an order might be fitted to the EDSAC which would cause the following operations to be carried out: "Transfer the number in the more significant half of the accumulator to the store (as is done by a T-order) but make the sign digit a 0, whatever its original value; if the sign digit were originally a 1, leave the number 2^{-34} in the accumulator, but otherwise leave the accumulator empty." If the components of the multi-length numbers are added in sequence, beginning with the least significant, the use of this order enables the carry-over from less to more significant components of the result to be programmed very simply. For example, it will be found that two double-length numbers, each stored in two locations, can be added and the result put in two locations in the store, by means of six orders.

Counting and repetition. B-registers, the idea of which was originated at Manchester University, provide the most flexible and most generally useful means of facilitating the organization of repetitive loops and operations involving counting. Other systems in which the machine can be made to execute an order a certain number of times with automatic modification of addresses are sometimes provided and, in some circumstances, can lead to very fast operation. In providing such systems the designer of a machine usually has some special purpose in mind, such as the transfer of information to and from a magnetic drum.

Entering of closed subroutines. It is quite common to find an order which both records, in a register of the arithmetical unit or in the store, the point reached in the program, and also transfers control to a closed subroutine. Such an order would replace the pair of orders used for this purpose in the EDSAC.

Logical orders. The collate order of the EDSAC performs the logical operation "and." Sometimes the logical operations "or" and "not equivalent to" are provided. In most applications one logical operation will usually be found sufficient.

Manual input. Many machines are provided with a set of switches on which a single word can be set up by hand and transferred to a selected storage location. The EDSAC uses a telephone dial for a similar purpose. Sometimes it is possible to read the number set up on the switches into the machine during the course of a program, a special input order being provided for the purpose. Facilities of this type are valuable, but should not be used to such an extent that the speed of the machine is limited by that of the operator.

The above list is not complete, and the reader may encounter other facilities in particular machines, for example, facilities for taking a modulus or for exchanging a number held in the accumulator with one in the store. For the most part, however, the purpose and use of such facilities will be sufficiently plain without further discussion here.

3-8 Minimum-access coding. In many machines the orders are executed, as in the EDSAC, in the serial order in which they stand in the store, except when transfer of control is brought about by the action of a jump order. An alternative system is to include in each order a specification of the location from which the next order is to be taken. This means that each order must include an extra address, and we will speak of a $1 + 1$ address code or a $3 + 1$ address code, or as the case may be. This system has advantages in the case of a machine which uses a magnetic drum for its main store. Words stored on a drum are available only at certain times in a fixed cycle. If a number or order is to be extracted from a location chosen at random there will therefore be a delay equal, on the average, to half the rotation time. If, however, the programmer has control over the location from which the next word is to be obtained, he can reduce this delay by placing the orders and numbers, so far as possible, in locations chosen so that they become available at the moment they are required. Such a procedure is known as *minimum-access* coding or, alternatively, as *optimum* coding. This system has also been applied in the case of machines which use ultrasonic tanks for the main store.

Since both orders and numbers are stored on the same drum, the programmer is able to do very little towards arranging for both to become available when required, unless he is provided with a number of special storage registers which have an access-time short compared with that of the main store. Usually such registers each hold a single word; in some machines they can be used to hold either orders or numbers, while in other machines they can be used for numbers only.

The procedure of minimum-access coding can best be explained by means of an example. The order code used will be that of the EDSAC, with an extra address written to specify the location of the next order.

In the machine used for the purpose of our example, it will be assumed that all orders and numbers are of the same length, and that the store consists of a magnetic drum with a number of tracks, each holding 64 words. Addresses will be written in the form p,n where p is the number of the track and n is the number of the word on the track. Locations 0,0 through 0,3 refer, not to the drum, but to four rapid-access registers. An example of an order is

$A\ p,n\ q,m$	add $C(p,n)$ into the accumulator and take next order from q,m.

The execution of orders with function letters A,S,H,T, etc., takes a minimum of two word-times, and the execution of orders with function letters V,N takes a minimum of 36 word-times. These minimum times will be exceeded if the machine has to wait for access to the storage locations on the drum specified in the orders. The example taken will be to form $x(ay + bz + c)$ and leave the result in the accumulator. x,y,z are in 0,0, 0,1, 0,2; a,b,c are constants which can be placed on the drum in positions chosen to minimize access time. The program, written to be placed on track 1, is as follows:

```
1,0  | H  1,1   1,2  ]
1,1  |     a         ]  Executed in first revolution
1,2  | V  0,1   1,38 ]
. . .|               |
. . .|               |
1,12 | A  1,13  1,14 ]
1,13 |     c         ]
1,14 | T  0,3   1,16 ]
1,15 |               ]  Executed in second revolution
1,16 | H  0,0   1,18 ]
1,17 |               ]
1,18 | V  0,3   1,54 ]
. . .|               |
 ...
1,38 | H  1,39  1,40 | ]
1,39 |     b         | ]  Executed in first revolution
1,40 | V  0,2   1,12 | ]
 ...
 ...
1,54 | Z             |    Executed in second revolution
 ...
```

In the case of this simple program there is no difficulty in reducing to an absolute minimum the effective access time of the store. In longer programs it will often be found that a location into which it is desired to put an order is already full, so that the order must be placed in a later location, with consequent loss of time when the program is run. There is clearly room for the exercise of some ingenuity in finding the best arrangement of orders which will minimize access time. In minimum-access programming there is a tendency to write out orders at length rather than to use repetitive loops since, unless a loop occupies an integral number of drum revolutions, some time is inevitably lost. It is convenient, if possible, to confine each subroutine to one or two tracks, in order to facilitate program assembly. Constants are usually placed on the drum in such positions that they can be obtained with a minimum of delay when required; when the same constant is required at more than one point, copies of it may be placed in more than one location. The success of the programmer in minimizing access time depends largely on the skill with which he can make use of the rapid-access registers; it may sometimes be convenient to transfer numbers from rapid-access registers into locations on the drum where they will be subsequently useful, thus freeing the rapid-access registers.

Although it is worth while optimizing as carefully as possible library subroutines and those sections of calculations which account for a large part of the running time, there is little advantage to be gained in optimizing the remainder of the program. Unless this fact is clearly realized and acted upon, a great deal of effort can be spent to little advantage.

3-9 The evaluation of an order code. There has been much discussion among programmers as to the relative merits of different types of order code. Many of the so-called arguments put forward are not arguments at all, but expressions of taste. For example, some codes are said to be more "natural" than others, but this clearly depends on the speaker's point of view.

A proper evaluation of a machine must be made by forming an estimate of its over-all effective speed on typical calculations, and relating this to the cost of installation and operation. Many things besides the order code must be taken into account in forming such an estimate; these include time to perform an operation, size and type of main store, size and type of auxiliary store, reliability, speed of input and output, etc. On the whole, a prospective purchaser of a machine will choose the one which gives him the performance he requires at the least cost. Programmers have, however, every right to demand that a machine shall be reasonably easy and straightforward to program and, indeed, the cost of operation depends to some extent on this being the case. The authors believe that for a machine with a random-access store, whatever the type of basic order code, provided the order code has been designed with some intelligent regard to the technical requirements of the programmer, then a satisfactory and easily handled system of programming can be established by paying proper attention to the design of the initial input routine. The authors do not like minimum-access coding, but are prepared to tolerate it so long as it enables relatively cheap machines, fitted with magnetic drum stores, to compete in speed with more expensive machines using random-access stores. They look forward to the day, however, when technical advances in the design of random-access stores will remove this economic advantage in favor of the magnetic drum.

3–10 Use of an auxiliary store. Many modern machines, especially the larger ones, are provided with auxiliary stores in addition to the main store. An auxiliary store can take the form either of a magnetic drum or of one or more magnetic tapes. Other machines have fast methods of input and output which enable information to be temporarily put out of the machine with a view to subsequent re-input, or which enable fresh blocks of information to be taken in as required. Naturally, the manipulation of an auxiliary store is a matter of much concern to the programmer, and the efficiency with which calculations are performed frequently depends on his skill in organizing the necessary transfers of information between it and the main store. However, the subject is not one on which it is easy to write in general terms, since so much depends on the type of auxiliary store provided, the size of the blocks of information which are normally transferred, the time taken for a transfer, the size of the main store, and the over-all computing speed of the machine. All these factors must be taken into account when planning a particular computation.

In some calculations, for example those arising in connection with partial differential equations or with matrices, there may be very many numbers to be handled, but relatively few orders. In these cases the natural thing to do is to keep the program in the main store, and to transfer numbers to and from the high-speed store in blocks. Since the transfer of a block of information from the auxiliary store normally requires a time which is long compared with that required to execute a single order, it is desirable to organize the calculation in such a way that as few transfers as possible are required. This requirement may influence the programmer in his choice between one numerical method and another. For example, it is usually found that if a large matrix is so disposed in the auxiliary store that rapid access can be obtained to all the elements of a single row, then access to all the elements of a single column will be relatively slow; methods which involve the alternate scanning of rows and columns of a large matrix are therefore at a disadvantage compared with methods which deal with rows or columns only.

In other problems the handling of numbers is relatively straightforward, and the principal question confronting the programmer is how to deal with a program which is too long to be accommodated entirely within the main store. Here, much depends on the size of the block of information which can be transferred from the auxiliary store in a single operation, and on the time taken to make the transfer. At the one extreme is the case of a machine with a high-speed magnetic drum from which a block of, say, 50 words can be transferred to the main store in a single rotation time. Here it may pay the programmer to transfer single subroutines from the drum as he requires them. At the other end of the scale is a machine using magnetic tape for auxiliary storage, in which the transfer of information takes an appreciable fraction of a second, but in which the whole of the main store, or as much of it as is required, can be replenished in a single transfer operation. In using such a machine the programmer would tend to divide his calculation into a number of stages, each of which was performed by a separate program with its own master routine and subroutines. Somewhat the same kind of method can be used in a machine which has rapid punched-card input, the programs for the various stages being loaded in the card reader and called for by the machine as required. The examples given are extreme cases, and the best method to adopt in a particular case would depend on the nature of the problem to be solved and on the facilities provided by the machine available.

CHAPTER 4

INPUT AND OUTPUT

4–1 Introduction. In the EDSAC, as in many other machines, punched paper tape is used both for input (of orders and numbers) and for output. Input and output are carried out by the machine as the result of orders in the program, just as other operations are. The unit input operation, which is performed as the result of a single input order, is the reading of one row of holes on the input tape and, correspondingly, the unit output operation is the punching of a single row of holes. To do anything more elaborate, such as to read an order or a number composed of several decimal digits, a subroutine is needed.

4–2 Input of numbers. The paper tape used for input is prepared by means of a keyboard perforator. There are five positions across the tape in which holes may, or may not, be punched, and one row of holes may therefore be said to represent a five-digit binary number. The keyboard perforator has 32 keys labelled with letters, figures, and other symbols, as in the case of an ordinary teleprinter keyboard. Each key causes one row of holes to be punched on the tape, according to the code given in Appendix 1. The corresponding five-digit binary numbers are also given in this appendix.* It will be seen that the figures 0 through 9 are represented by their binary equivalents; for example, 5 is represented by 00101, 6 by 00110, etc. Most keys are marked both with a letter and with a figure or other symbol. For example, the key marked *T* is also marked 5, so that *T* and 5 both correspond to the same row of holes on the tape.

The input order of the EDSAC is given below. It is assumed that the numerical equivalent of the row of holes standing under the reading head of the tape reader is x.

$I\ n$	Place $x \cdot 2^{-16}$ in storage location n and advance the tape by one row of holes.

The five binary digits read from the tape are thus transferred to the five least significant positions of a short storage location.

Suppose that it is required to put the number 0.21973 into the machine. The successive digits of this number are punched, in order, on the input

* A hole in the tape represents the binary digit 1, except in the case of the most significant digit, where a 1 is represented by the absence of a hole. This is done in order to avoid having to represent the number 00000 by blank tape.

tape. When the tape is read by the machine, acting under the control of a succession of *I* orders in the program, the binary equivalents of the following numbers will be transferred to the store in succession:

$$2 \cdot 2^{-16}$$
$$1 \cdot 2^{-16}$$
$$9 \cdot 2^{-16}$$
$$7 \cdot 2^{-16}$$
$$3 \cdot 2^{-16}$$

The program contains orders which cause the first of these numbers to be multiplied by 10^4, the second by 10^3, the third by 10^2, the fourth by 10, the last by 1, and the results to be added. This calculation is carried out in the binary scale, so that the binary equivalent of $21973 \cdot 2^{-16}$ is now to be found in the store. A further multiplication by $2^{16} \cdot 10^{-5}$ forms the required number in its binary form. It will be seen that the decisive step in the conversion of the number to the binary scale takes place in the keyboard perforator, which converts the individual decimal digits of the number to their binary form.

In practice, multiplication by successive powers of ten is not performed exactly in the manner implied above but by making use of the cyclic procedure for the evaluation of a polynomial, which was shown in Example 6 of Section 1–13. In the notation of this example, y is taken equal to ten, and a_0, a_1, a_2, . . . are the decimal digits in decreasing order of significance. Since it is necessary, in drawing up the program, to avoid the use of numbers that lie outside the range $-1 \leq x < 1$, it is not possible to multiply by 10 directly; instead, we multiply by 10/16 and shift the result four places to the left in the manner illustrated in Example 13 of Section 2–11. The detailed program of an input subroutine will be found in Part 3 (see subroutine *R* 2).

4–3 Output of numbers. Conversion of numbers to their decimal form during output is done in an analogous manner. However, certain complications have been introduced to reduce the likelihood of numbers correctly computed being incorrectly printed owing to errors in the output mechanism. Originally a teleprinter, directly connected to the machine, was used for output. This accepted a five-digit binary number and printed the corresponding character. As in the case of input, the code was so chosen that the binary numbers 0 through 9 were printed as the corresponding decimal figures; for example, 00101 was printed as 5, 00110 as 6, etc. It was realized that occasionally the wrong five-digit binary number might be set up on the teleprinter, and for this reason means were provided whereby the programmer could cause the number so set up to be read back into the store and used for checking purposes. This method of checking, however, required the use of a nonstandard teleprinter, and it was found that faults due to the reading-back system were common; moreover, it could not readily be adapted for use with an output punch, which it was proposed to substitute for the teleprinter. The system of checking by reading back was therefore abandoned in favor of a "logical" system using a "two-out-of-five" code.*

The code used for output is given in Appendix 1, along with that used for input. It will be seen, in the output code, that each of the figures 0 through 9 is represented by a five-digit binary number in which two, and only two, of the digits are 1's. The corresponding rows of holes on the output tape have two holes and three blanks, and the teleprinter used for final printing of the results is arranged to accept this code. It will be seen that a decimal digit can be replaced by another decimal digit only as a result of the occurrence in the output system of two compensating errors, one of which turns a hole into a blank and the other a blank into a hole. The degree of security against error is therefore good and, moreover, is not lost if output tapes are copied. The symbols +, −, and "space" are represented on the tape by rows consisting of four holes and one blank. These can be changed into rows representing decimal digits, and vice versa, by two like errors. The inherent degree of security against this type of error is not therefore quite so great but, on the other hand, such errors would usually give rise to quite conspicuous irregularities of layout.

Output subroutines are slightly complicated by the use of the two-out-of-five code. The procedure adopted is first to compute, in binary form, the decimal digits to be printed, and then to convert these digits to the output code by means of a ten-entry table included in the subroutine.

The output order of the EDSAC is as follows:

O n	Punch a row of holes corresponding to the five most significant digits (including the sign digit) in storage location *n*; a hole corresponds to a 1 and a blank to a 0.

The principle of the method by which the successive decimal digits are computed is as follows. The number (assumed to be positive and less

* The order which caused the number set up on the teleprinter to be read back into the store had function letter *F*. Some time after the facility provided by this order had been removed from the EDSAC, the function letter *F* was re-used for the present *F*-order, which gives an absolute transfer of control. Anyone referring to the first edition of this book should avoid confusing the two orders.

than unity) is multiplied repeatedly by ten, and the integral part removed each time. If the number were expressed as a decimal fraction, this method would clearly isolate the successive digits, beginning with the most significant. The same is true if the number is expressed as a binary fraction (the multiplication being by ten in its binary form, that is, by the binary number 1010) except that the digits are then obtained in the form of the corresponding binary numbers. When this method is programmed for the EDSAC it is necessary, in order to avoid using numbers outside the range $-1 \leq x < 1$, to multiply by 10/16 instead of by ten, and to take the four digits which come immediately after the binary point. The remainder is shifted four places to the left before a further multiplication is performed. An example of an output subroutine, which illustrates the above procedure in detail, will be found in Part 3 (see subroutine *P*40).

4-4 Input of orders. The only way in which a symbol punched on the tape can be read into the machine is by the operation of an *I*-order, which reads a single row of holes. To enable a program tape to be read, therefore, means are provided in the EDSAC whereby a short group of orders can be placed in the store independently of the input mechanism. These orders, which form the initial input routine referred to in Chapter 2, are wired in binary form on a set of stepping switches (uniselectors) and are automatically transferred to the store (and called into action) when the Start button is pressed. The initial input routine is needed only while the program tape is being read, and the space it occupies in the store may be used again for other purposes during the course of the calculation.

It has already been explained that the translation of the orders, from the external form in which they are written and punched to the internal form which they take inside the machine, is performed by the initial input routine. This translation includes the conversion of the address to decimal form, the addition of any constant called for by the terminal code letter, the assembly of the complete order, and its placing in the correct storage location. It is important to realize that the relation between the form in which orders are punched on the tape and the form in which they appear in the store is determined solely by the initial input routine. By making the two forms more similar (for example, by punching the address in the scale of 8 or 16), or by omitting the facilities offered by the terminal code letters, it would be possible to simplify the initial input routine. There would, however, be no advantage in doing this, and it would mean that more work would be left to the programmer. It is highly desirable that the machine itself should carry out as much as possible of the clerical work involved in drawing up the program; the chance of error is then reduced, and the programmer is left free to concentrate his attention on the more essential aspects of the program.

It follows that the choice of the initial input routine, and thus of the form in which orders are punched, is a matter for careful consideration, since upon it depends the ease with which all programs are constructed. Once the choice has been made, a library of subroutines formed, and a number of large jobs begun, any change in the form of writing and punching orders will entail a big reorganization. The form used with the EDSAC was changed in September 1949, after only a few simple programs had been run; it was not long, however, before any further change would have been practically out of the question, even if it had been desired.

The initial input routine used in the EDSAC is given in full in Appendix 3. The action of the routine is, however, somewhat complicated to follow, partly because of the complexity of the task it performs, and partly because it was necessary to use every legitimate trick known to the programmer in order to make the routine short enough to be accommodated on the uniselectors. The reader may find it of assistance, therefore, to study the simplified initial input routine given below. This provides for the reading of orders and the interpretation of terminal code letters, but does not provide for the recognition of control combinations. This omission would, of course, render the routine quite useless in practice, since there would be no way of stopping the reading of the program tape and causing control to be sent to the first order of the program. The order in 25, which places the assembled order in the correct location in the store, is known as the "Transfer Order."

	Order	Notes	
0	*T* *F*	Clear accumulator	Later used for temporary storage
1	*A* 30 *F*	Plant 2^{-16} in 43	
2	*T* 43 *F*		
3	*T* 41 *F*	Clear 41	
4	*H* 31 *F*	Set $C(R) = 10/32$	
29 → 5	*I* 1 *F*	Read, shift, and store function digits	
6	*A* 1 *F*		
7	*L* 1024 *F*		
8	*T* 2 *F*		
9	*F* 14 *F*	Jump to 14	
18 → 10	*A* 32 *F*		
11	*R* 4 *F*	Decimal-binary conversion; address is built up in 0	
12	*V* *F*		
13	*L* 8 *F*		
9 → 14	*T* *F*		
15	*I* 1 *F*	Read next character, x say, from tape and place $x \cdot 2^{-16}$ in accumulator	
16	*A* 1 *F*		

(continued)

17	*S* 32 *F*	Test and jump if $x < 10$	
18	*G* 10 *F*		
19	*L* *D*	Form and plant order to add $C(y)$ where y is specified by code letter	
20	*A* 33 *F*		
21	*T* 24 *F*		
22	*A* 2 *F*	Add function digits	
23	*A* *F*	Add address	
24	( )	Planted by 21; add $C(y)$	
25	(*T* 56 *F*)	Transfer Order	
26	*A* 25 *F*	Increase address in Transfer Order by unity	
27	*A* 34 *F*		
28	*T* 25 *F*		
29	*F* 5 *F*	Jump to read next order	
30	*P* *D*	$\equiv 2^{-16}$	Pseudo-orders
31	*T* *F*	$\equiv 10/32$	
32	*P* 5 *F*	$\equiv 10 \cdot 2^{-16}$	
33	*A* 34 *F*	Base order	
34	*P* 1 *F*		

Although the space occupied by the initial input routine of the EDSAC may be used again for other purposes during the execution of the program, it is usual, when writing programs, to arrange that locations 2 and 3, which contain the pseudo-orders *P* 1 *F*, *U* 2 *F*, have their contents undisturbed. These two pseudo-orders are frequently required, and library subroutines are written on the assumption that they are in the locations stated. It will be remembered that the pseudo-order *U* 2 *F* is used by closed *A* subroutines for forming the link order.

The first few inches of a program tape are always left blank, and the tape is inserted in the tape reader with the reading head somewhere on the blank portion. It is not necessary to set the first row of holes under the reading head, since the initial orders are designed to have the property of ignoring blank tape, in the sense that they do not erase anything of importance from the store when it is read. It is, however, necessary to punch a control combination at the end of the blank tape and immediately in front of the orders. The usual control combination is *P Z T m K*, which causes orders to go into the store starting at location *m*. It is often convenient also to have a few rows of blank tape in front of each subroutine, in order to facilitate locating the subroutines when making corrections to the tape. At least two such blank rows must be left and the control combination *P Z* must be punched to follow them.

As explained earlier, the control combination *E a K P F* causes control to be transferred to location *a*, which ordinarily contains the first order of the program. It may be that the programmer wishes further orders to be read from the tape after the program has proceeded some distance. He may do this by including in the program orders which place 10/32 in the multiplier register and transfer control to storage location 25. This has the effect of calling the initial input routine in again. The accumulator need not be empty when control is transferred to 25, but the first group of symbols to be read from the tape must be a control combination that will set the Transfer Order, for example, $T\ n\ K$.* If it is intended to use the initial input routine again in this way, care must be taken to see that its orders are not written over by numbers during the course of the program. If the initial orders have been written over, they may be replaced (after the machine has stopped) by pressing the Start button again; the contents of other parts of the store will be left undisturbed.

4–5 Recognition of the code letter *S*. The initial input routine used with the EDSAC was designed before the B-register was fitted, and does not provide for the interpretation of the letter S (see Section 2–13). Therefore a short subroutine which provides this extra facility is punched at the head of each program tape which requires it. The subroutine ($R30$) is given in Part 3, and it will be seen that, when it is read into the store, jump orders are planted in $27F$ and $28F$ so that $R30$ behaves as though it were an integral part of the initial input routine.

4–6 Economy of input and output time. The EDSAC is so designed that after an input order has been executed, calculation can proceed while the tape is being advanced in the tape reader to bring the next row of holes under the reading head. If another input order occurs before the tape is fully advanced, however, the machine waits. The amount of calculation required for binary-decimal conversion is not normally enough to take full advantage of the time available between the reading of characters and, when a series of numbers is being read, the speed of the machine is limited by that of the tape reader. If, however, a sufficient amount of calculation takes place between the reading of successive numbers, the method described below may be used to remove this limitation. It will be supposed that the numbers to be read all consist of n characters punched on the tape. The programmer places in the program, at convenient points, n separate input orders, each of which causes a character to be read from the tape into one of a set of n consecutive storage locations. It does not matter where these orders are placed, but they should be separated in time by an amount sufficient for the tape reader to come to rest after advancing the tape one row. If this is not done the full gain in speed offered by the

* Alternatively, the Transfer Order may be replaced by orders included in the program, and control sent, with the accumulator clear, to 34.

present method will not be obtained. The program is otherwise made up in the ordinary way, but the input subroutine used must be specially designed to take successive characters from the set of n locations in the store, instead of from the input tape; a standard subroutine can be easily modified for this purpose. The modified input subroutine runs at full speed, and the over-all speed of the machine is limited by the speed at which calculations can be performed, and not by the speed of the tape reader. A similar method may be used to economize output time.

4–7 Some features of input systems used with other machines. Many machines use punched paper tape for input. Some use five-hole teleprinter tape, as in the case of the EDSAC, and some seven-hole Flexowriter tape. Many other machines use I.B.M. (Hollerith) punched cards. So far, few machines use magnetic tape for input and output, although a good many use it for auxiliary storage. We shall not attempt, in this section, to give any general survey of the input systems used with digital computers, but will mention a number of points of direct concern to the programmer.

4–8 Punched tape. In the EDSAC, the basic input operation—that called for by a single input order—is to read a single row of holes on the tape, and to place the resulting five digits in the store. In many other machines, by contrast, the basic input operation is to read a number of rows of holes, and to assemble the resulting digits side by side to form a complete word; if this system were applied to the EDSAC the logical arrangement would be for seven rows to be read, giving 35 binary digits which would just form a complete long word. If it were necessary to choose between one facility and the other, the authors would prefer the reading of a single row of holes as the basic input operation, since this enables the various facilities which have been described in this chapter and the preceding one to be secured with a minimum of complication in the program. However, if both facilities can be provided, the possibility of reading a complete word direct from the tape by a single order can be used to advantage. Perhaps the most important advantage is that it enables a program tape to be read without the use of a permanently wired-in initial input routine. This will be explained in relation to the EDSAC, it being supposed that an order is provided which causes seven successive rows of holes to be read, and the resulting 35 binary digits, arranged in a row, to be transferred to the store as a long number. It will also be supposed that means exist whereby, on pressing a button, the operator can cause such an order to be introduced into storage location 0, and control to be sent to that location.

The method of initial input to be described depends on the fact that the order in storage location 0 can call for the input of a long word, containing two orders, from the tape and that these two orders can each call for the input of two further orders, and so on. Starting with the single order in 0, it is thus possible to put in as many orders as are desired. In practice, this method would be used to introduce into the machine a standard initial input routine, which would then be used to read the program tape. Thus every program tape would begin with the initial input routine punched in this special way.

The input order need not actually be introduced into storage location 0, provided that the pressing of the Start button sets up such connections in the control unit of the machine that the machine behaves as if such an order had been introduced. However, a simple method of achieving the desired result without any extra apparatus whatever is to allocate to the special input order function digits which are all zero; thus, in EDSAC notation we should have

```
P n F | Read a long word (seven rows) from the tape, and place
      |   it in storage location nD.
```

The order *P F* (in which all the digits are 0) can then be introduced into storage location 0 (and into all other storage locations as well) by clearing the store; if the register which holds the address of the next order to be executed is cleared at the same time, control will be sent, on pressing the Start button, to location 0. A tape punched in the manner shown below will then be read; this tape has the orders of an initial input routine interleaved with special input orders in such a way that they are placed in the store in locations 4 through 41. When all the orders are in the store, control is sent to location 4. The orders are written here, for convenience, in the standard form used for writing EDSAC orders but, in practice, it would be necessary to punch them in binary form, with seven rows of holes to each pair.

```
P    F  ]  Order pair read into 0D
P  2 F  ]
P    F  ]  Order pair read into 2D
F    F  ]
P  4 F  ]  Order pair read into 0D
P  6 F  ]
. . . . ]
. . . . ]  1st four orders of initial input routine;
. . . . ]    read into 4D and 6D
. . . . ]
P  8 F
P 10 F
```

(*continued*)

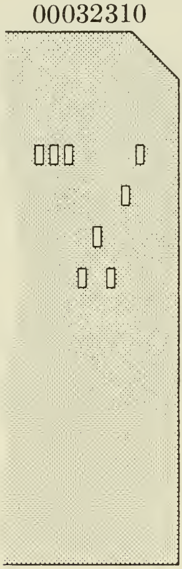
```
  . . . .
  . . . .     2nd four orders of initial input routine;
  . . . .       read into 8D and 10D
  . . . .
     .
     .
     .
 P 40 F
 F  4 F       Order pair read into 0D
  . . . .     Last two orders of initial input routine;
  . . . .       read into 40D
```

4–9 Punched cards. In all, or nearly all, digital computers which use punched cards for input, the cards are read row by row by means of a card reader similar to that used in ordinary tabulators. As each row is read, the corresponding binary number—in which a 1 corresponds to a hole and a 0 to a blank—is placed in the store. Cards have 80 columns, but in most machines it is not possible to read digits from all the columns during a single passage of the card; most often the number of columns which can be read in a single passage is equal to the number of digits in a word, although in some machines it is possible to read twice this number. It is customary to provide a plug-board which enables the operator to select, in advance, the columns to be read.

When used for primary input of data to a binary machine, cards are normally punched in the usual punched-card code, that is, one decimal digit is punched in each column, in a one-out-of-ten code. However, when information is put out of the machine on cards, with a view to subsequent re-input, it can be punched in the binary system, each row of holes corresponding to a binary word (or to two binary words) in the machine. Cards punched in this way provide a very efficient form of storage, and are frequently used for subroutines, standard programs, and other often-wanted data. There is little which need be said about the use of cards for storing information in binary form, but the use of cards punched in decimal code for input and output raises the problem of binary-decimal conversion.

Some machines are provided with an input order which causes a single card to be read, and a set of twelve words, one corresponding to each row of holes, to be placed in the store. For example, if the card has punched on it the number 7896821..., the word corresponding to the bottom row of holes (in which 9's are punched) will be 0010000..., that corresponding to the bottom but one row of holes (in which 8's are punched) will be 0100100..., etc. The twelve words may be said to form an *image* of the card in the store. In machines of this type, conversion of the numbers punched on the card to the binary scale must take place after the whole card has been read. The conversion is best done by a method in which the decimal digits are isolated in turn (by scanning the image, or by applying some equivalent procedure) and the usual method based on the evaluation of a polynomial applied. The following program in EDSAC code illustrates one way in which a decimal-binary conversion subroutine may be written.

For simplicity, it will be assumed that positive numbers only are involved. The card image consists of 12 long words in storage locations 100*D* through 122*D*, as shown in the example below.



Row	Location	Card image Word	Converted card image
		2^{-34} ↓	
X	100*D*	...00000	...00000
Y	102*D*	...00000	...00000
0	104*D*	...00001	...11110
1	106*D*	...00010	...11100
2	108*D*	...01000	...10100
3	110*D*	...10100	...00000
4	112*D*	...00000	...00000
5	114*D*	...00000	...00000
6	116*D*	...00000	...00000
7	118*D*	...00000	...00000
8	120*D*	...00000	...00000
9	122*D*	...00000	...00000

Conversion takes place in two stages. During the first, the card image is converted to the form shown on the right, in which the number of 1's in each column is equal to the decimal digit punched in the corresponding column of the card. This stage carries with it a check against mispunching or misreading of the card. During the second stage, the columns are summed successively, and the binary form of the number built up.

The card image occupies digits 2^{-25} through 2^{-34} of each of the storage locations concerned. For the first stage a number consisting of 1's in each of the above digital positions, and 0's everywhere else, is placed in the accumulator; $C(104D)$ is then subtracted from this number, and the result placed back in 104*D*. This will have a 1 wherever $C(104D)$ had a 0 and a 0 where $C(104D)$ had a 1. $C(106D)$ is next subtracted from the number remaining in the accumulator, and the result placed back in 106*D*. This will have 1's everywhere except in those positions where $C(104D)$ or $C(106D)$ originally had 1's. The process is repeated for $C(108D)$ through $C(122D)$, after which $C(\text{Acc})$ is tested to see that it is zero. This will be so if each column of the card image had one and only one digit equal to 1. A check of the correct punching and reading of the card is thus obtained. At this point the converted card image is available in the original storage locations.

The second stage consists of the summation of the individual columns of the converted image, and the building up of the binary form of the number by a process which is equivalent to the usual process for the evaluation of a power series. Both stages should be made clear by a study of the program which will now be given; it assumes the presence of the following three constants:

10D	2^{-24}	This is a number with a single 1 in the left-hand column position
12D	$2^{-24}-2^{-34}$	This is a number with a 1 in each column position
14D	$2^{-24}-2^{-33}$	This is a number with a 1 in each column except the least significant

		Order			Notes	
		G		*K*		
	0	*B*	20	*S*	First stage (form new card image)	
	1	*A*	12	*D*		
5 →	2	*BS*	2	*F*		
	3	*SS*	122	*D*		
	4	*US*	122	*D*		
	5	*J*	2	θ		
6 →	6	*F*		$6\pi\theta$	Dynamic stop if check fails	
	7	*T*		*D*	Clear 0D	
	8	*A*	10	*D*		
20 →	9	*T*	4	*D*	Plant test digit in multiplier register	Second stage (form converted integer in 0D)
	10	*H*	4	*D*		
	11	*B*	20	*F*		
16 →	12	*BS*	4	*S*	Decimal-binary conversion cycle (repeated 5 times)	
	13	*CS*	104	*D*		
	14	*CS*	106	*D*		
	15	*A*		*D*		
	16	*J*	12	θ		
	17	*T*		*D*		
	18	*C*	14	*D*	Build up integer in 0D	
	19	*R*		*D*	Shift test digit	
	20	*F*		$9\pi\theta$	Test for end of cycle	

The above method is applicable in machines in which a single input order causes a whole card to be read. In other machines the card must first be set in motion by a special order included in the program, and each row of holes must be read, as it arrives under the reading brushes, by a separate reading order. From the point of view of the programmer, the card is then very much like a piece of broad paper tape with, however, the important difference that once set in motion it cannot be arrested until all the rows have passed the reading brushes. It is open to the programmer to include orders, for any purpose connected with the program, between the input orders which read successive rows of holes, but he must take care that the execution of these orders does not take longer than the time available. If an input order is encountered in the program before a row of holes has arrived at the reading brushes, the machine will wait, but if the row has passed the reading brushes when the order arrives, the information contained in that row is irretrievably lost.

In a machine working on the principle just described, it is possible to use some of the time between the reading of the rows of holes for performing part of the decimal-binary conversion. Suppose that the first row of holes to be read (which has holes punched where the corresponding decimal digits are 9's) gives a binary word whose rth digit is a_r, where $a_r = 0$ or 1. The contribution from this row of holes to the number being read is then $9\Sigma a_r 10^r$. Similarly, if b_r is the rth digit of the word corresponding to the second row of holes, the contribution from this row of holes is $8\Sigma b_r 10^r$; and so on. The various sums, of which $\Sigma a_r 10^r$ is typical, may be evaluated immediately after the reading of the corresponding row of holes, and a sequence of orders for doing this is given below. Given this sequence, a subroutine for reading a decimal number from a card and converting it to binary form may be constructed without difficulty. The sequence is entered at its first order, with the binary word read from a row on the card in $6D$. It is assumed that the decimal numbers have ten digits, and that the corresponding binary words occupy the ten binary positions immediately to the right of the binary point. It is further assumed that $C(4D) = 2^{-34}$ and $C(R) = 10/16$.

```
             | G      K  |
          0  | B   10 F  |   Set count
   11 →   1  | T      D  |
          2  | A    6 D  | ┐
          3  | L      D  | │ Test next digit in binary word and
          4  | U    6 D  | │   jump if it is a 0
          5  | E    8πθ  | ┘
          6  | T    8 F  |   Clear accumulator
          7  | A    4 D  |   Add unity
```

(*continued*)

```
5 →  8 | V      D |  ┐
     9 | L    4 F |  ┘ Multiply partial result by ten
    10 | BS   1 S |  ┐
    11 | J    1 θ |  ┘ Count and test for end
    12 | ........ |
```

It will be seen that this is yet another example of the evaluation of a power series, the coefficients in this case being either 0 or 1.

In the case of binary-decimal conversion performed as a preliminary to output, it is necessary that the whole number should be converted to decimal form before the first row of holes is punched, and there is not much scope for saving time by performing part of the binary-decimal conversion between the punching of the rows. Whatever the precise method used in a particular machine for controlling the card, it will usually be found that a convenient procedure is to form in the store an image of the card to be punched, before punching starts.

Orders may be punched on cards in a notation similar to that used in the EDSAC, use being made of the standard two-hole combinations used to represent letters in ordinary punched-card practice. The number of orders which can be accommodated on a card depends on the number of columns which can be read during one passage. Programs punched in this way are relatively bulky, and when a subroutine or program has been tested, it is convenient to put it out of the machine in binary form onto a new set of cards; these can then be preserved, and used for subsequent re-input. Punching the original cards by hand in binary form is not to be recommended, since punching and checking of programs is then a highly skilled operation.

If, as seems possible, fast card readers which read the card column by column, instead of row by row, come into general use, the scheme of punching one or two 6-digit characters in each column might commend itself. If this were done each card would hold 160 characters, the equivalent of 16 inches of 6-hole tape. Existing keyboard perforators, which are designed to punch only one or two holes per column, would not, however, lend themselves to this method of punching.

So far, while discussing punched cards, we have had binary machines in mind. The most obvious arrangement likely to occur to the designer of a decimal or alpha-numeric machine is to use cards punched in the standard punched-card code, and to provide circuits for automatic conversion to whatever binary code is used internally. The advantages of cards in providing compact storage and rapid input of subroutines and standard data, however, are more fully realized if it is possible to punch and read back information expressed in coded binary form. If automatic conversion to and from standard punched-card code is not provided, the problem of programming such conversion raises considerations very similar to those raised by binary-decimal conversion in binary machines.

CHAPTER 5

THE LIBRARY OF SUBROUTINES

5–1 Introduction. A library will contain subroutines of different types. Some will be subroutines of general utility and will be in constant use by programmers. Others will be of more specialized interest and will have been written for particular projects undertaken with the machine. These subroutines are preserved in the library partly because they may be of further use for the same or later projects, and partly because they embody the result of experience which has generally been obtained by the expenditure of much time and trouble. Intelligently used in this way, a library can help to prevent the experience of a group being dissipated when individual programmers leave, and can provide a source of information for newcomers about the methods which have been found effective with the particular machine in use.

This chapter is concerned primarily with the EDSAC library and with subroutines of the former type, that is, those of general interest. However, some attention is given to the special requirements of other machines where these differ from those of the EDSAC.

5–2 Library catalog. The catalog of the library of the EDSAC is drawn up in two sections. One gives a concise specification of the action of each subroutine, together with sufficient information to enable a programmer to make use of it; this includes information about operating time, precision, and the storage space occupied. The other section gives the orders of the subroutines in full. An abbreviated version of the first section of the current catalog of the EDSAC library is given in Part 2 of this book, and an abbreviated version of the second section in Part 3. New subroutines are continually being added, and older subroutines displaced. Part 3 contains the program of at least one subroutine of most types, and a number of other subroutines which are thought to have points of special interest. Some of the subroutines date from a time before the *B*-register was fitted to the EDSAC.

In addition to subroutines, the library contains a number of complete programs for well-defined problems which repeatedly recur, such as the solution of linear algebraic equations. It also includes post-mortem routines.

In a number of cases, it will be found that there are two or more subroutines which perform very similar operations. Usually they differ in time of operation and in storage space occupied. Normally, the one with the shortest operating time would be chosen for any particular application, but if the program occupies nearly all the main store, then storage space may become the major consideration. Time of operation is of great importance only if the subroutine is called in many times during the program, thus consuming a large proportion of the total time.

Subroutines may also differ in numerical accuracy and in the number of parameters which may be varied to suit particular applications. If a subroutine has many parameters it is sometimes useful to provide separate versions to deal with cases that commonly arise.

5-3 Input subroutines. All input subroutines read numbers punched on the tape in the scale of ten, convert them to the scale of two, and place them in the store. Some subroutines read a single number only each time they are called in, while others read a series of numbers and place them in consecutive locations in the store. The subroutines may be further classified into those which read decimal fractions from the tape and those which read integers; in the latter case, when an integer n is read from the tape, $n \cdot 2^{-16}$ or $n \cdot 2^{-34}$ is put into the store, according as short or long numbers are being used. Some input subroutines are designed to read numbers punched in input code, and some to read numbers punched in output code; the latter are useful for reading back into the machine information which has earlier been put out. Most input subroutines are designed to ignore blank tape and layout symbols such as "carriage return" and "line feed." This makes it possible to punch information in such a way that it can be printed on a page teleprinter for purposes of proofreading or record, and yet be accepted by the machine.

Many subroutines contain numbers as well as orders. Short numbers are easily put in as pseudo-orders. A sequence of long numbers requires the use of an input subroutine, but if only one or two long numbers occur they may be put in as pairs of pseudo-orders. It is, however, necessary that the sandwich digit (see Section 2-10) should be zero, and this may make it necessary to use the complement of a number instead of the number itself. When a subroutine needs a series of constants, such as the coefficients of a power series, it is common practice to write it on the assumption that subroutine $R9$ will be in the store when the subroutine is read.

5-4 Output subroutines. These convert numbers in the store to decimal form and cause them to be punched, in output code, on the output tape. Most output subroutines also punch layout symbols such as space, carriage return, and line feed, different subroutines providing different page layouts. Some output routines deal with fractions, and others with integers. The most complicated output subroutine is $P57$, which offers a wide range of layouts and enables the programmer to change the layout in the course of a calculation.

Subroutines which print fractions are normally arranged so that the number to be printed is rounded off to the correct number of decimal places. This is done by adding the round-off constant, $\frac{1}{2} \cdot 10^{-n}$, where n is the number of decimal places, to the modulus of the number before printing. n is specified by means of a preset parameter, and the round-off constant is calculated in the following manner during the reading of the input tape, by means of what is known as an *interlude.* The short sequence of orders used to calculate the round-off constant is first read from the tape and placed in the store. Control is then set to the first order of this sequence, by means of a control combination punched on the tape. Finally, when the calculation of the round-off constant is complete, the last order of the sequence returns to the initial input routine, and the reading of the tape is resumed (see Section 4–4). The orders of the interlude are then overwritten by orders of the subroutine. Usually an output order which causes a figure shift symbol to be punched on the output tape is included in the interlude. Interludes are used for a variety of other purposes in library subroutines, and further examples will be found in subroutines such as *C*31, *F*7, and *G*12 in Part 3.

5–5 Division subroutines. The order code of the EDSAC does not include an order for division, which must therefore be carried out by means of a division subroutine. Several division subroutines are included in the library; the one most often used is *D*11, which divides the double-length (69-digit) number held in the accumulator by a single-length number. This enables an expression of the type ab/c to be evaluated in a straightforward manner, even if ab is less than 2^{-34}. If the only form of division subroutine available is one which divides a single-length number by another single-length number, it is not possible to write a program which will evaluate ab/c without risk of serious loss of precision, unless use is made of conditional operations; it is assumed that a, b, c and ab/c all lie within the range required by the machine, but that a, b, and c are otherwise unrestricted. The same difficulty arises in a machine with a built-in divider which is not capable of handling a double-length dividend. Of the subroutines described in Parts 2 and 3 it might be noted that *D*6 uses an iterative formula, and is designed to give the greatest accuracy. *D*7 and *D*11 use a simpler repetitive formula.

5–6 Trigonometric and other functions. When values of trigonometric and other functions are required for arbitrary values of the argument, it is usually better to use a subroutine which calculates them from a power series or other formula, rather than to place a table in the store and to interpolate into it. However, tables are of value if speed is important, if storage space can be spared, and if the circumstances allow the interval of tabulation to be chosen so that tabular values only are required; interpolation takes an appreciable time and, when it is necessary, the use of a table is less attractive.

When values of a function are to be calculated from a formula, it is, in many cases, quickest and simplest to use a power series economized by the use of Chebyshev polynomials. A method of doing this is given in the next section. It may be advantageous to express the power series in terms of an auxiliary variable; for example, in the case of $\log x$ it is convenient to use $y = (1 - x)/(1 + x)$. If the machine has a divider, use may be made of formulas involving the ratio of two polynomials. The use of continued fractions is also sometimes convenient with such machines.

Approximations to functions suitable for use in a digital computer may often be arrived at by semi-empirical methods. Useful information about such approximations is given by C. Hastings (see Bibliography).

Sometimes repetitive methods, based on very simple formulas and needing very few orders, are available; these are, however, usually slow. An example is to be found in subroutine *S*3, in which the required answer is built up digit by digit.

When a series of sines or cosines is required, with equal increments of the argument, subroutines based on a recurrence formula may be used. This situation occurs when a differential equation involving a sine or cosine of the independent variable is being solved.

The above remarks apply also in the case of inverse trigonometrical functions, and one or two subroutines for computing such functions should be included in a library. A subroutine designed to convert cartesian coordinates to polar coordinates may also be found useful.

5–7 The economization of a power series by the use of Chebyshev polynomials. The first few Chebyshev polynomials, defined in the range $-1 \leq x \leq 1$, are as follows:

$$\left.\begin{aligned} T_0(x) &= 1, \\ T_1(x) &= x, \\ T_2(x) &= 2x^2 - 1, \\ T_3(x) &= 4x^3 - 3x. \end{aligned}\right] \tag{1}$$

This set of equations may be extended by use of the recurrence relation

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x). \tag{2}$$

The Chebyshev polynomials have the property that their maximum and minimum values in the range $-1 \leq x \leq 1$ are all 1 and -1, respectively; thus $|T_n(x)| \leq 1$.

Suppose that it is known that the following power series represents the function $f(x)$ with sufficient accuracy in the range $-1 \leq x \leq 1$:

$$f(x) = \sum_{0}^{n} a_r x^r. \tag{3}$$

By the use of Chebyshev polynomials it is often possible to obtain a new power series, with fewer terms, which will give similar accuracy. The principle of the method by which this is done may be explained as follows. Suppose that equations (1), extended as far as $T_n(x)$, are used to eliminate x from (3); we then have, instead of a power series, a series of Chebyshev polynomials; i.e.,

$$f(x) = \sum_{0}^{n} a_r x^r \equiv \sum_{0}^{n} b_r T_r(x).$$

All that has been done so far is to make an algebraic transformation. It will often be found, when numerical values are put into the Chebyshev series, that the contributions from the last few terms are negligible to the accuracy required. If these terms are dropped, and the resulting truncated Chebyshev series reconverted to a power series, the required economized power series is obtained. Since $|T_r(x)| \leq 1$ in the given range, the condition for the term $b_r T_r(x)$ to be negligible is simply that b_r should be negligible.

In practice, complete conversion of the power series to a Chebyshev series is not necessary, and the economized series can be obtained directly as follows. Evaluate $|a_n/t_{n,0}|$, where a_n is the coefficient of x^n in the given power series, and $t_{n,0}(= 2^{n-1})$ is the coefficient of x^n in $T_n(x)$. If this quantity is negligible to the accuracy required, subtract $(a_n/t_{n,0})T_n(x)$ from the given series; this gives an economized series of degree $n - 1$. If $|a_n/t_{n,0}|$ is not negligible, the given power series is not capable of economization.

The above procedure is repeated until further economization is not possible. When carrying out the economization process it is desirable to work to slightly more accuracy than the given series itself warrants, in order to control the effect of rounding-off errors. It is possible to program a digital computer to perform the process of economization, and such a program is included in the EDSAC library, although it is too long to be given in this book.

A table of coefficients for the Chebyshev polynomials

$$T_n(x) = t_{n,0}x^n - t_{n,2}x^{n-2} + t_{n,4}x^{n-4} - \cdots$$

for $n \leq 20$, taken from Jones, Miller, Conn and Pankhurst, 1946 (see Bibliography), is given on page 85. More extensive tables are given by the National Bureau of Standards in Applied Mathematics Series 9 (1952).

	$t_{n,2p}$										
$n\backslash 2p$	0	2	4	6	8	10	12	14	16	18	20
0	1										
1	1										
2	2	1									
3	4	3									
4	8	8	1								
5	16	20	5								
6	32	48	18	1							
7	64	112	56	7							
8	128	256	160	32	1						
9	256	576	432	120	9						
10	512	1280	1120	400	50	1					
11	1024	2816	2816	1232	220	11					
12	2048	6144	6912	3584	840	72	1				
13	4096	13312	16640	9984	2912	364	13				
14	8192	28672	39424	26880	9408	1568	98	1			
15	16384	61440	92160	70400	28800	6048	560	15			
16	32768	1 31072	2 12992	1 80224	84480	21504	2688	128	1		
17	65536	2 78528	4 87424	4 52608	2 39360	71808	11424	816	17		
18	1 31072	5 89824	11 05920	11 18208	6 58944	2 28096	44352	4320	162	1	
19	2 62144	12 45184	24 90368	27 23840	17 70496	6 95552	1 60512	20064	1140	19	
20	5 24288	26 21440	55 70560	65 53600	46 59200	20 50048	5 49120	84480	6600	200	1

EXAMPLE. To economize the following series:

$$\text{arc tan}\left(\tfrac{1}{2}x\right) = \frac{x}{2} - \frac{x^3}{2^3 \cdot 3} + \frac{x^5}{2^5 \cdot 5}$$

$$\simeq 0.5x - 0.0417x^3 + 0.0062x^5.$$

This series gives arc tan $(\frac{1}{2}x)$ to slightly less than 3-decimal accuracy in the range $-1 \leq x \leq 1$. For this series, we have

$$|a_5/t_{5,0}| = 1/(2^9 \cdot 5),$$

which is negligible to 3-decimal accuracy. An economized series can, therefore, be obtained by subtracting from the given series the following quantity:

$$[1/(2^9 \cdot 5)]T_5(x) = [1/(2^9 \cdot 5)](5x - 20x^3 + 16x^5)$$

$$\simeq 0.0020x - 0.0078x^3 + 0.0062x^5.$$

We then obtain

$$\text{arc tan}\left(\tfrac{1}{2}x\right) \simeq 0.4980x - 0.0339x^3.$$

Since 0.0339/4 is not negligible, further economization is not possible.

5–8 Quadrature. Section *Q* of the EDSAC library contains subroutines for computing definite integrals. When the integrand is given by an algebraic expression, a subroutine based on a Gauss formula is ordinarily to be preferred. The desk computer's objection to Gauss formulas, namely, that the function has to be computed for awkward values of the argument, is of no account when using an automatic machine; this is an illustration of the different considerations which apply when selecting methods for automatic computing as compared with desk computing. *Q*3 is based on the ordinary 6-point Gauss formula, and *Q*19 on a 6-point formula with an exponential weighting function. In each case an auxiliary subroutine for calculating values of the function to be integrated must be constructed by the user; this is called in as required by the integration subroutine. Other subroutines, based on formulas with larger numbers of points (up to 14), are available in the library. Generally, however, it is perhaps safer to cover an extended range of integration by the repeated application of, say, a 6-point formula, rather than by the use of a single formula with many points (see Hildebrand, pp. 79–82, 334).

In some cases, for example if the function to be integrated is obtained by integrating a differential equation, a formula which uses equally spaced ordinates may be more suitable than a Gauss formula. In these circumstances *Q*20, which is based on Gregory's formula, may be used. A subroutine based on Simpson's rule was formerly included in the library, but this method is so simply programmed with the aid of the *B*-register that a library subroutine is hardly necessary.

5–9 Integration of ordinary differential equations. The EDSAC library includes a number of subroutines for integrating ordinary differential equations. *G*4 enables a second order differential equation with first derivative absent to be integrated; this is based on a conventional iterative method in which use is made in each interval of values of the function calculated in previous intervals. Special methods are needed for starting the integration.

*G*12 and *G*13 are subroutines for integrating sets of simultaneous, first-order, differential equations, not necessarily linear, using methods of the Runge-Kutta type. *G*13 uses the standard Runge-Kutta process, and *G*12 uses a variant due to Gill which has the advantage of requiring one less storage location per variable. A description of the Runge-Kutta-Gill method is given in Section 5–10.

Any differential equation or set of differential equations can be reduced to a system of first-order equations; for example, the equation

$$x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (n^2 - x^2)y = 0$$

may be written

$$y' = z,$$

$$z' = [-xz - (n^2 - x^2)y]/x^2.$$

Subroutines *G*12 and *G*13 are thus of wide utility. On many occasions, however, methods of the Runge-Kutta type are relatively slow in operation, although they have the advantage that a special starting procedure is not necessary. *G*13 is somewhat faster than *G*12.

When particular functions occur in the coefficients of a set of differential equations, it is often more convenient to add extra differential equations than to make use of subroutines for the purpose. For example, in the case of

$$y' = \sin y,$$

the set of differential equations to be solved may be taken as

$$y' = z,$$

$$z' = u,$$

$$u' = (1 - 2z^2)z.$$

$G14$ is a subroutine which will locate a zero of a function defined by a differential equation. For this purpose it makes use of $G12$ or $G13$ as an auxiliary subroutine.

5–10 Library subroutines $G12$ and $G13$: Runge-Kutta processes. These subroutines handle sets of simultaneous, first-order, differential equations, in which each derivative is expressed in terms of the variables

$$\begin{aligned} y'_1 &= f_1(y_1, y_2, \ldots, y_n), \\ y'_2 &= f_2(y_1, y_2, \ldots, y_n), \\ &\vdots \\ y'_n &= f_n(y_1, y_2, \ldots, y_n). \end{aligned}$$

The case in which the functions f involve the independent variable can be treated by the methods described in Section 5–11.

The subroutine $G12$ or $G13$ carries out one step of the integration each time it is called in. In doing so, it makes use of an auxiliary subroutine which evaluates the functions $f_1, \ldots, f_n$. The auxiliary subroutine must be provided for the individual problem; it is called into play four times during each step.

The auxiliary subroutine is asked to provide the quantities hy' multiplied by a suitable scale factor 2^m, where h is the length of the interval and m is chosen to be as high as possible without exceeding capacity.

Apart from the $2n$ long storage locations used to store y and $2^m hy'$, n additional storage locations are used by $G12$ as temporary storage for the quantities $2^m q$ (see Section 5–12), and $2n$ additional storage locations are needed by $G13$. The numbers left standing in the first n of these locations, at the end of a step, are $3 \cdot 2^m$ times the rounding-off errors of the quantities y; they are taken into account during the following step, and serve to prevent the rapid accumulation of rounding-off errors. As a result, the effective numerical accuracy is m digits more than the capacity of the storage locations. At the beginning of a range the locations used for temporary storage must be cleared, otherwise the integration routine will add spurious "corrections" to the variables. Apart from planting the initial values of the variables, this is the only preparation required before starting an integration. The truncation error in one step is of order h^5.

5–11 The independent variable. If the independent variable occurs in the functions f, it may be obtained by integrating the equation $x' = 1$. x is treated as an additional dependent variable, for which the auxiliary subroutine has to provide the quantity $2^m hx' = 2^m h$. The latter may be planted at the beginning of the integration and left there, so that the auxiliary subroutine is relieved of the task. If the independent variable does not appear in any of the f's, but is merely wanted for indication purposes, it is quicker to use a simple counter in the master routine.

When x is generated by integrating $x' = 1$, the values which it assumes during the four applications of the auxiliary subroutine within one step are x_0, $(x_0 + \frac{1}{2}h)$, $(x_0 + \frac{1}{2}h)$, and $(x_0 + h)$, respectively, where x_0 is the beginning of the step. This has two implications. First, if time is of great importance, x may be generated by using a binary switch in the auxiliary subroutine, so that $\frac{1}{2}h$ is added every other time the subroutine is used; x may then be used in calculating the f's but does not require the introduction of an additional dependent variable. Second, if the f's involve a function of x which is tabulated at equal intervals, it will be necessary to employ only the tabulated values, or values interpolated at simple fractions of the tabular interval.

If h cannot be expressed exactly in binary form, there is a numerical advantage in generating x by integrating $x' = 1$. Owing to the high digital accuracy afforded by the "bridging" values of $2^m q$ which are carried over from one step to the next, the accumulation of rounding-off errors in x occurs much less rapidly than it would if x were obtained by the repeated addition of h.

5–12 Definition of the Runge-Kutta-Gill process. This process is defined by the equations below. y_{i0} is the value of the ith variable at the beginning of a step; y_{i4} is its value at the end of the step. While the $2^m k_{ip}$'s for one value of p are being calculated by the auxiliary subroutine, the corresponding y_{ip} and $2^m q_{ip}$ $(i = 1, \ldots, n)$ are stored. The quantities r_{ip} are used only in the formation of the corresponding y_{ip} and q_{ip}, and do not need to be carried over to the following value of p. Each r is rounded off to the same number of places as y.

y_{i4} and q_{i4} become y_{i0} and q_{i0} for the following step. The scale factor 2^m employed in storing k and q is left out for simplicity.

$$\left.\begin{aligned}
k_{i0} &= hf_i(y_{00}, y_{10}, \ldots) \\
r_{i1} &= \tfrac{1}{2}k_{i0} - q_{i0} \\
y_{i1} &= y_{i0} + r_{i1} \\
q_{i1} &= q_{i0} + 3r_{i1} - \tfrac{1}{2}k_{i0} \\
k_{i1} &= hf_i(y_{01}, y_{11}, \ldots) \\
r_{i2} &= (1 - \sqrt{\tfrac{1}{2}})\,(k_{i1} - q_{i1}) \\
y_{i2} &= y_{i1} + r_{i2} \\
q_{i2} &= q_{i1} + 3r_{i2} - (1 - \sqrt{\tfrac{1}{2}})\,k_{i1} \\
k_{i2} &= hf_i(y_{02}, y_{12}, \ldots) \\
r_{i3} &= (1 + \sqrt{\tfrac{1}{2}})\,(k_{i2} - q_{i2}) \\
y_{i3} &= y_{i2} + r_{i3} \\
q_{i3} &= q_{i2} + 3r_{i3} - (1 + \sqrt{\tfrac{1}{2}})\,k_{i2} \\
k_{i3} &= hf_i(y_{03}, y_{13}, \ldots) \\
r_{i4} &= \tfrac{1}{6}(k_{i3} - 2q_{i3}) \\
y_{i4} &= y_{i3} + r_{i4} \\
q_{i4} &= q_{i3} + 3r_{i4} - \tfrac{1}{2}k_{i3}
\end{aligned}\right] \qquad i = 1, \ldots, n$$

5–13 Taylor-series method. A program, too long to give in Part 3, has been constructed for solving

$$p(x)\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} + r(x)y = 0,$$

where $p(x)$, $q(x)$, $r(x)$ are polynomials of degree two, with arbitrary coefficients. The program is based on the use of Taylor's formula, derivatives being calculated by means of the recurrence formula

$$py^{(n+2)} + (np' + q)y^{(n+1)} + \left[\frac{n(n-1)}{2}p'' + nq' + r\right]y^{(n)} + \left[\frac{n(n-1)}{2}q'' + r'\right]y^{(n-1)} + \frac{n(n-1)}{2}r''\,y^{(n-2)} = 0.$$

The program is arranged to use as many terms of the Taylor series in each step as are necessary to give the accuracy required. Automatic scaling down, to prevent an overflow, is provided. Methods based on the use of Taylor's series have the advantage that a very large interval in the independent variable may be used if desired.

5–14 Interpretive subroutines. The use of interpretive subroutines (see Section 2–21) effectively extends the order code of the machine by increasing the complexity of the operations which may be performed in response to a single order. However, the resulting gain in expediency of programming is offset by the considerable increase in the time required by the machine to carry out the calculation, because of the higher percentage of orders concerned purely with organizing the calculations.

The EDSAC library contains interpretive subroutines for carrying out arithmetical operations on complex numbers (subroutine $B2$) and on double-length numbers (subroutine $N2$). The code of interpretive orders used in connection with $B2$ closely resembles the basic order code of the EDSAC.

5–15 Floating-point subroutines. $A11$ is an interpretive subroutine for performing the operations of addition, subtraction, and multiplication on numbers expressed in floating-decimal form. Each number is expressed in the form $a \cdot 10^p$, where $-10 \leq a \leq 10$ and $-63 \leq p < 63$, and is represented in the store by $a \cdot 2^{-11} + p \cdot 2^{-6}$. $A9$ and $A10$ are input and and output subroutines designed to be used in conjunction with $A11$.

$A30$ is an example of an interpretive subroutine which enables the programmer to work with an order code quite different from the basic order code of the machine; it provides him, in effect, with an entirely new machine. The action of $A30$ is best described in terms of this hypothetical machine.

Numbers are expressed in the form $a \cdot 10^p$, where a and p are packed into a single storage location. The number of digits defining p may be varied from 4 to 15 by means of a preset parameter, so that a suitable value for the permissible range of variation of numbers may be selected for a given calculation. The "arithmetical unit" contains an accumulator and a multiplier register; the arithmetic orders provided are very similar to those of the EDSAC except that they work with floating-point numbers. There is a B-register, and the interpretive orders may be B-modified. The facilities of $A30$ may be extended by the use of special auxiliary subroutines which allow for input, output, the extraction of a square root, etc. Assembly facilities for the subroutines are also provided, along the lines described in Section 8–3.

Although the use of floating-point operation can simplify the programmer's task by relieving him of undue preoccupation with scaling, it must not be thought that it solves all his difficulties. In particular, the loss of significant digits resulting from the subtraction of a number from a nearly equal number can have serious consequences unless proper precautions are taken. In floating-point operation, effects of this kind can present themselves in unfamiliar guise, and the programmer should be on his guard.

CHAPTER 6

DIAGNOSIS OF ERRORS IN PROGRAMS

6–1 Introduction. Even a first-class computer will sometimes make a mistake (although he will not allow it to remain undetected for long). In the same way a programmer will sometimes make a mistake in the master routine, in a subroutine, or in the make-up of a program tape. Some mistakes may cause the answer to be in error. Others may cause the machine to obey a wrong sequence of orders, or to try to treat as an order some word intended as a number or pseudo-order. In the latter case the machine will perhaps stop on a meaningless order, or go into a closed loop—that is, repeat a short sequence of instructions over and over again. The machine may print a number of symbols, or it may print nothing at all.

Experience has shown that such mistakes are much more difficult to avoid than might be expected. It is, in fact, rare for a program to work correctly the first time it is tried, and often several attempts must be made before all errors are eliminated. Since much machine time can be lost in this way, some importance attaches to the adoption of efficient techniques for avoiding errors, for detecting them before the program is put on the machine, and for locating, with a minimum expenditure of machine time, any which remain undetected up to that point.

Library subroutines are all checked on the machine before being put into the library, and the programmer may regard them as being almost certainly free from error. This in itself would be a sufficient reason for having a library, quite apart from other considerations. When subroutines are specially made for a particular program, it is good practice to test them beforehand, by means of short programs constructed for the purpose.

It is easier to avoid and detect errors if the program is drawn up in an orderly and logical manner. For example, if six quantities x_1, x_2, x_3, y_1, y_2, y_3 occur, they should be placed in consecutive storage locations, and not scattered about the store. Similarly, orders and pseudo-orders used for counting purposes should be arranged on some plan, and not placed at random in the store. In the early stages of drawing up a complicated program, the programmer should not hesitate to copy it out in a more logical layout when necessary. The parallel case of hand computation will suggest itself; good computers generally pay close attention to the arrangement of their work sheets.

It is of great assistance, both to a programmer and to a person checking the program, to provide notes describing the actions of the orders, as is done for the examples given in Part 1 of this book and for the library subroutines given in Part 3. The notation for entry points, pseudo-orders, etc., summarized at the beginning of Part 3, is also designed to help in understanding programs.

6-2 Proofreading of programs. Some idea of the types of mistake which can occur in programs is given by the following list of points which should be checked before a program is punched. Many of the mistakes are of a purely clerical nature, and could be detected by a person without great mathematical ability; others require an understanding of the particular calculation. Although many errors can be detected if the program is checked by a second competent programmer, this usually requires so much time as to be uneconomic in practice.

1. No two subroutines may occupy the same storage locations, unless one is used only temporarily before the other is inserted.
2. All conditions contained in the specification of each library subroutine used must be met. For example, if it is necessary that the subroutine should start in an even location, this point should be checked. It should also be made certain that all parameters have been correctly specified.
3. All subroutines should be correctly called in, according to the system in use. For example, in the EDSAC, the accumulator must be clear when closed *A* subroutines are called in.
4. When alterations have been made to programs, it should be verified that any necessary renumbering has been correctly done.
5. Counting operations should give the correct number of repetitions.
6. The program should be prepared in such a way that locations are left for any orders which are planted by the program itself. In the case of the EDSAC this is usually done by writing a dummy order such as *Z F*, or *P F*.
7. It should be verified that control is directed to the correct place to start the program.
8. No item of information in the store should be overwritten unless it is no longer required; in particular, no wanted information should be left in locations that are used for temporary storage by a subroutine.
9. It must not be assumed that the content of the multiplier register is unaltered by a subroutine.

The above list, which is based on EDSAC experience, is not exhaustive, but will serve to indicate the type of error that anyone checking a program should be on the lookout for.

6-3 Punching. A program once written and checked must be transformed into a form which can be accepted by the machine. Usually, this means that it must be punched onto cards or paper tape. When cards are used, ordinary punched card equipment is available for punching, verifying, correcting, and copying. When tape is used, special equipment is required.

Corrections to a program punched on a deck of cards are easily made by replacing one or more of the cards by new ones. When corrections have to be made to a program punched on paper tape, it is best to copy the whole tape with the aid of a device which enables corrections to be incorporated in the copy. Sometimes small hand punches, which enable individual holes or rows of holes to be punched, can be used. In some centers, corrections are made by splicing tapes and by sticking patches over holes. We feel that, if reliable tape duplicating equipment with adequate editing facilities is available, programmers are likely to find such expedients messy when used for correcting individual subroutines. When very long program tapes are in use, however, the joining of longish sections by splicing can save much tedious copying of tapes.

It is usual to reserve the combination in which a hole is punched in each position across the tape for use as an "erase" symbol. If a punch operator presses a wrong key, he backspaces and presses the "erase" key, and then punches the correct symbol. The initial input routine can be designed to ignore erase symbols, but in the EDSAC we use a specially designed tape duplicator to remove erase symbols from program tapes before they are read into the machine. Many of our number input subroutines, however, ignore erase symbols.

Experience shows that tape editing and verifying equipment working at telegraph speed (about 7 rows per second) is too slow for convenient use in a digital computer laboratory; speeds of 12–15 rows per second are acceptable, but still higher speeds are desirable.

6–4 Locating mistakes in a program. Most machines have a push button by which the operator can make the machine execute a program one order at a time. Many machines also have monitors, attached to the arithmetic unit and the store, which display the numbers and orders contained therein. It might be thought that a good way of finding errors in a program would be to make the machine proceed order by order, and to study the progress of the calculation by watching the monitors. This, however, usually turns out to be a very slow and inefficient process, especially in a machine in which the numbers are displayed in binary form. Methods have therefore been developed in which the machine proceeds unhindered by the operator, but prints a permanent record of the progress of the calculation; this record can be studied at leisure and will assist in understanding the nature of the mistake.

One such method is to wait until the machine has stopped (or to stop it deliberately) and then, without clearing the whole store, to insert a small program which will print in suitable form the contents of part of the store. This has come to be known as the *post-mortem* method. Various post-mortem routines are available in the EDSAC library and copies of the tapes are kept available near the EDSAC. Some of these print the numbers standing in consecutive storage locations, starting from any desired point; others print orders. A refinement of this method, which takes advantage of the fact that much of the information in the store is unchanged by the program, is known as the *comparison post-mortem* method. In this method the program is read a second time into the machine, and only those orders or numbers which have been changed during the course of the calculation are printed. The second reading of the program may be avoided by placing it at the outset in an auxiliary store.

Stopping of the program at a suitable point for the post-mortem method to be applied is facilitated if the machine is provided with one or more *break-point orders* or *conditional-stop orders.* Such an order causes the machine to stop if a key on the control desk is in the depressed condition; if the key is in the normal position, the break-point order has no effect. Break-point orders, if available, should be included at strategic points in a program when it is first drawn up, with a view to facilitating the subsequent location of errors. The EDSAC has a break-point order, written as $Z\ D$ (see Appendix 2).

The post-mortem method yields only a static picture of the store as it was when the calculation stopped. Other methods have been developed to provide information about the whole course of the calculation. These necessarily involve modifying the program to cause extra printing. This may be done either by making alterations to the tape or cards on which the program is punched, or by reading into the machine, when the program is already there, a specially prepared sequence of orders which will modify the original orders where necessary.

One simple plan is to place output orders at various points in the program, for example at the beginning of the master routine and in front of each subroutine, so that the completion of the various stages of the program will be recorded by the printing of suitably chosen symbols. If, by reason of a programmer's mistake, the machine stops in the middle of the program, the symbols printed will enable the error to be localized. When a teleprinter is used for output, letter and figure shifts must also be inserted if letters are required for indication purposes and if it is desired that the ordinary printing of numbers called for by the program shall take place correctly. When the program has been made to work satisfactorily, the extra printing orders may be removed. It is a good plan to include extra printing orders of the type described here in new programs when they are first drawn up, rather than to wait until the program has been tried and found to fail.

6–5 Subroutines for checking programs. In many cases the modifications to a program required for error diagnosis are quite extensive. It has been found possible to construct subroutines which make these modifications automatically, and which are sufficiently general to be applied to any program. These form category *C* of the EDSAC library.

A particularly useful type of error-diagnosis subroutine is known as a *check-point* subroutine. When such a subroutine is read into the store, it causes a number of unconditional jump orders to be planted at specified points in the program. During subsequent running of the program, control is transferred, at these points, to the check-point subroutine, which causes information required for diagnostic purposes to be printed; control is then returned to the program, which runs in the normal way until another of the planted jump orders is reached. The information printed by the check-point subroutine may consist simply of symbols which indicate, for example, that the various subroutines are being operated in the correct order, and that repetitions are taking place the correct number of times. Alternatively, it may cause the numbers or orders standing in specified storage locations to be printed, and thus provide information which the programmer can use to locate the point, if any, at which the program fails to do what he intended it to do.

Error-diagnosis subroutines can also be constructed on the interpretive principle. Such a subroutine is placed in the store along with the original program, and control is sent to the subroutine. The subroutine treats the original program as though it consisted of interpretive orders (see Section 2–21); the subroutine extracts the orders from the program and causes them to be executed, but at the same time it prints additional information. A useful error-diagnosis subroutine of this type prints the function letters of orders as they are executed, starting a new line of printing whenever a transfer of control takes place. This provides a very convenient means by which the programmer can locate with precision the point at which the machine departs from the sequence of operations he intended to lay down when he wrote the program. An example of the use of such a subroutine is given in Chapter 7, Example 2. The printed sequence of function letters is sometimes known as a *trace*.

An alternative form of trace consists of a list of locations to which, and from which, control was transferred by jump orders during the running of the program. Since most programs contain simple loops in which a sequence of orders is repeatedly executed many times, it is a convenience if the error-diagnosis subroutine is designed to detect such loops and to print the information in abbreviated form; for example,

```
150 — 50      67 — 53(100)      82 — 207
```

might indicate that control was transferred from 150 to 50 and that transfer from 67 to 53 took place 100 times, after which control was transferred from 82 to 207.

Trace-printing subroutines can be designed to have a delayed start, for example to start printing after control has passed through a specified location a given number of times, or when control is first sent to a specified location. They can also be designed in such a way that no trace is printed of closed subroutines, which therefore run at full speed. Alternatively, a subroutine can be arranged to store the trace, without printing, in a limited number of storage locations, earlier information being progressively overwritten by later information. Then, when the machine has stopped, it is possible to obtain a printed record of, for example, the last twenty occasions on which control was transferred or a single loop entered.

Although interpretive methods of error diagnosis are very powerful, they suffer from the general disadvantage of all interpretive subroutines, namely, that they slow down, very appreciably, the operation of the program. Moreover, if the error is not encountered until the program has been running for some time, a great deal of time is wasted in unnecessary printing. For these reasons interpretive methods of error diagnosis have not proved to be capable of such universal application as at one time seemed likely. They are, however, extremely powerful, and are hard to beat for finding errors in relatively short programs which contain intricate switching.

A general point to bear in mind when assessing the value of procedures for error diagnosis is that those which are simplest to use are likely to become the most popular among programmers. What is potentially a good method can be spoiled if the method of applying it has not been well enough worked out. Even details of organization are important—for example, the convenience with which an operator can lay his hands on the right error-diagnosis subroutine when working under pressure on the machine. The mistake of making error-diagnosis subroutines too complicated in an attempt to make them of very general application should be avoided.

6-6 The development of a program. Many mistakes in programs cause the machine to stop or to proceed on some course of action which makes it quite obvious that a mistake is present. These mistakes can be located by the methods which have just been discussed. Some mistakes, however, cause the numbers operated on to be in error, without immediately affecting the sequence in which the orders are obeyed. It cannot, therefore, be assumed that if a program apparently operates correctly it is giving correct results, and careful numerical checks must always be applied.

A numerical fault may be due to a mistake in a single order or constant or to a more fundamental mistake, such as a wrong choice of scale factors, that causes some numbers to take values outside the range permitted in the machine. Such mistakes can be quite difficult to diagnose, although an overflow alarm, such as that provided with the ϕ-order in the EDSAC, will often pinpoint the error.

When a mistake in a program has been located the next task is to correct it. It is best to do this in a way which will minimize the likelihood of further errors being introduced in the process. Although it is worth taking pains to see that programs, when first written, have a logical layout, once the task of making a program work has been seriously undertaken, the programmer should resist the temptation continually to rewrite it in a more elegant form, since every time he does so he is likely to introduce new mistakes. It is best to make corrections in such a way that as little as possible of the program is disturbed. Suppose, for example, that a correction involving the introduction of additional orders has to be made at a certain point in the master routine. The existing order at that point may be replaced by a jump order which will send control to another part of the store, where the displaced order and the necessary correcting orders can be placed; these are followed by another jump order which sends control back to the master routine. Fewer mistakes are likely to be made if this procedure is used than if the extra orders are inserted in the middle of the routine and the subsequent orders renumbered.

When corrections to a program have been decided upon, the appropriate alterations must be made in the tape or cards used for input. This may be done either by correcting the errors where they occur and making any necessary additions, or by adding at the end a short piece of tape or a few extra cards on which the corrections are punched. These corrections are read into the store after the original program and, where necessary, overwrite the original orders. It is common practice, when programs are being developed for the EDSAC, to punch at the end of the program tape a control combination which will cause the machine to stop until the operator presses the Reset button. This provides an opportunity for the insertion of a correction tape if necessary. Later, when the program is fully developed, a fresh copy of the program tape is made, with the corrections added at the end and with the control combination omitted.

CHAPTER 7

EXAMPLES OF COMPLETE PROGRAMS FOR THE EDSAC

Example 1. Calculation of $e^{-\sin x}$. This program calculates and punches values of $e^{-\sin x}$ for various positive values of the argument x, which are supplied on a data tape. The results, when printed, consist of two columns, the left-hand column giving the argument and the right-hand column the corresponding value of the function. The quantities in each column are printed to nine decimal places. Each value of x is read and then punched on the output tape, and the corresponding value of the function is calculated and punched before the next value of x is read. The program stops when a negative number is read from the data tape.

Five library subroutines and a master routine specially written for this problem are used; they are positioned in the store as follows:

Routine	*Location of first order*	*Number of storage locations occupied*	*Type*
*R*9	56	15	Special (used by *E*4 and *T*7 during input)
*T*7 (sine)	72*	36	Closed *A*
*E*4 (exponential)	110*	36	Closed *A*
*R*37 (read fraction)	150*	34	Closed *B*
*P*30 (punch fraction)	190*	48	Closed *A*
Master	250	23	

* First order must be in an even-numbered storage location.

Master routine

		G K	
3 →	0	Z F	Wait until data tape is inserted in reader and Reset button pressed
22 →	1	B 1 θ	Call in $R37$, which reads x and leaves it in accumulator
	2	F 150 F	
$R37$ →	3	G $\pi\theta$	Jump if x is negative, thus stopping program
	4	T 8 D	Place x in $8D$
	5	H 8 D	Place x in multiplier register ready for $P30$
	6	A 6 θ	Call in $P30$, at its first order, to print argument on a new line
	7	F 190 F	
$P30$ →	8	A 8 D	Place $\frac{1}{2}x$ in $4D$ ready for $T7$
	9	R D	
	10	T 4 D	
	11	A 11 θ	Call in $T7$ to place $\frac{1}{2}\sin x$ in $4D$
	12	F 72 F	
$T7$ →	13	S 4 D	Place $-\sin x$ in multiplier register
	14	L D	
	15	T D	
	16	H D	
	17	A 17 θ	Call in $E4$ to place $e^{-\sin x}$ in $0D$
	18	F 110 F	
$E4$ →	19	H D	Place $e^{-\sin x}$ in multiplier register ready for $P30$
	20	A 20 θ	Call in $P30$, at its third order, to print function on same line as argument
	21	F 192 F	
$P30$ →	22	F 1 θ	Jump to read next value of x

Make-up of program tape

$\boxed{R9}$	*R*9 begins with *PZT* 56 *K*, so that it is automatically placed in storage locations 56 through 70.
space	
PZT 72 *K*	
$\boxed{T7}$	
space	
PZT 110 *K*	
$\boxed{E4}$	
space	
PZT 150 *K*	
$\boxed{R37}$	
space	
PZT 190 *K*	
GK	Places in 42 the address (190) currently specified in the Transfer Order
T 45 *K*	Sets the Transfer Order so that the parameter following goes into 45
P 1568 *F*	*H*-parameter for *P*30; specifies a digit layout of 5 digits, space, 4 digits
$\boxed{P30}$	*P*30 begins with *TZ*, so that the address stored in 42 is replaced in the Transfer Order. *P*30 is then placed in 190–237
space	
PZT 250 *K*	
$\boxed{\text{Master routine}}$	
EZPF	Transfers control to first order of master routine, with accumulator clear.

Make-up of data tape

1234 +	Values of *x*; they are read one at a time, by *R*37, when that subroutine is called in by the master routine
986 +	
74281079 +	
.	
.	
.	
1 −	When this (negative) number is read the program will stop.

Notes: (1) The word "space" in the description of the make-up of the program tape denotes that a few rows of blank tape are left immediately before a subroutine to facilitate tape handling and correction. These few rows must always be followed immediately by the control combination *PZ*. If desired, the blank tape may be omitted; in this case the *PZ* must also be omitted.

(2) It is sometimes desired to place subroutines "head to tail" in the store—that is, with the first order of one in the next storage location to that containing the last order of the previous one. This can be done automatically by omitting the *T m K* control combination following the *PZ* before each subroutine. Care must then be taken to ensure that those subroutines whose first orders must go into even storage locations are in fact so treated. Note also that, in any case, the *GKT* 45 *K* preceding *P*30 cannot be omitted, since this control combination temporarily breaks the regular sequence (of unit increments of address in the Transfer Order) in order to plant the *H*-parameter for *P*30 in storage location 45.

(3) If desired, the data tape could be combined with the program tape. The wait at the beginning of the program could then be by-passed by starting the program with *E* 1 *ZPF* instead of *EZPF*.

(4) At the end of the program the machine stops on the *Z*-order at the beginning of the master routine. If desired, another data tape could then be inserted, and the program restarted by pressing the Reset button.

(5) The whole program has a total of 192 orders, but only 23 have been specially written for this calculation, the rest being provided by library subroutines.

Example 2. The evaluation of a definite integral. To evaluate

$$4\int_0^1 (1 + x^2)^{-1}\,dx \quad (= \pi).$$

In order that all the numbers concerned in the calculation shall be less than 1, the formula is rewritten as

$$\int_0^{1/2} \frac{\frac{3}{16}}{\frac{15}{16}(\frac{1}{4} + x^2)}\,dx \quad (= \pi/10).$$

Evaluation of this integral may be conveniently carried out by a Gauss 6-point quadrature formula, and library subroutine *Q*3 is chosen for the purpose. For *Q*3, an auxiliary, closed *A* subroutine must be constructed to calculate the integrand for a given value of x, placed in 0*D*. The auxiliary subroutine is called in automatically by *Q*3, to which it returns control after calculating the desired value of the integrand and placing it in 0*D*.

A division subroutine is required in the course of calculating the integrand and library subroutine *D*11 is chosen. Two further library subroutines are needed: *R*9, which is required while *Q*3 is being read into the store, and *P*30, which punches the result on the output tape.

The master routine and the auxiliary subroutine make use of a number of constants (pseudo-orders). For convenience, these are all included in a single sequence numbered with reference to the code letter *M*.

Allocation of storage locations

Routine	*Location of first order*	*Number of storage locations occupied*	*Type*
*R*9	56	15	Special
*D*11	72*	30	Closed *B*
*P*30	110*	48	Closed *A*
*Q*3	160*	48	Closed *A*
Auxiliary subroutine	210	16	Closed *A*
Master routine	230	9	
M-sequence	240	3	

M-sequence

M	0	*R* *F*	$4 \cdot 2^{-4} = \frac{1}{4}$	
	1	*E* *F*	$3 \cdot 2^{-4}$	
	2	*K* 2048 *F*	$15 \cdot 2^{-4}$	(See Note)**

* First order must be in an even-numbered storage location.

** *Note:* The quantity $15 \cdot 2^{-4}$ is represented by the pseudo-order *K* 2048 *F*, rather than the more obvious "erase" *F*, for the following reason. It was pointed out in Section 6–3 that the row of holes representing "erase" can be automatically removed from a program tape by copying the tape in a special tape duplicator. If "erase" is used as a function "letter" there is thus a danger of its being unintentionally removed from the tape. For a similar reason, some subroutines represent the quantity —1 by the pseudo-order *K* 4096 *F*, rather than by "blank" *F*, since blank tape can also be removed when a tape is copied in the duplicator.

Master routine

		G		*K*	
	0	*A*		*M*	Set *a* and *h* for *Q*3 (see specification of *Q*3 in Part 2)
	1	*U*		*H*	
	2	*T*	2	*H*	
	3	*A*	3	θ	Call in *Q*3, which places integral in 0*D*
	4	*F*	160	*F*	
*Q*3 →	5	*H*		*D*	Place integral in multiplier register ready for *P*30
	6	*A*	6	θ	Call in *P*30 to punch integral on output tape
	7	*F*	110	*F*	
*P*30 →	8	*Z*		*F*	Stop

Auxiliary subroutine

		G		*K*	
	0	*A*	3	*F*	Form and plant link order
	1	*T*	15	θ	
	2	*H*		*D*	$(x^2 + \frac{1}{4})$ to 0*D*
	3	*V*		*D*	
	4	*A*		*M*	
	5	*Y*		*F*	
	6	*T*		*D*	
	7	*H*		*D*	$\frac{15}{16}(x^2 + \frac{1}{4})$ to 4*D*
	8	*V*	2	*M*	
	9	*Y*		*F*	
	10	*T*	4	*D*	
	11	*A*	1	*M*	Add $\frac{3}{16}$
	12	*B*	12	θ	Call in *D*11 to form *C*(Acc)/*C*(4*D*)(= integrand)
	13	*F*	72	*F*	
*D*11 →	14	*T*		*D*	Integrand to 0*D*
	15	(*Z*		*F*)	Becomes link order

Make-up of program tape

$\boxed{R9}$	*R*9 begins with *PZT* 56 *K*, so that it is automatically placed in storage locations 56 through 70
space	
PZT 72 *K*	
$\boxed{D11}$	
space	
PZT 110 *K*	
GK	
T 45 *K*	
P 1568 *F*	*H*-parameter for *P*30; specifies a digit layout of 5 digits, space, 4 digits
$\boxed{P30}$	
space	
PZT 160 *K*	
GK	
T 45 *K*	
P 10 *D*	*H*-parameter, specifies working space for *Q*3
P 210 *F*	*N*-parameter for *Q*3; specifies location of auxiliary subroutine
P 240 *F*	*M*-parameter
$\boxed{Q3}$	
space	
PZT 210 *K*	
$\boxed{\text{Auxiliary subroutine}}$	
space	
PZT 230 *K*	
$\boxed{\text{Master routine}}$	
space	
PZT 240 *K*	
$\boxed{M\text{-sequence}}$	
ZKPF	Stops machine until Reset button is pressed*
E 250 *KPF*	Transfers control to first order of master routine with accumulator clear

* Most program tapes for the EDSAC include this control combination just before the end. If a modification or checking tape has to be inserted this can be done when the machine stops at this point.

Example 2, *with a check-point subroutine.* By the use of the simple check-point subroutine *C*27 the program of Example 2 can be modified to print extra information to provide some record of its course of operation. The details given below show how the letters *Q*, *D*, or *A* could be printed each time the program calls in *Q*3, *D*11, or the auxiliary subroutine. The printed result from this program would be

*QADADADADADAD*3 1415 9265.

This result is achieved by inserting in the reader a specially punched check-point tape (see below) when the machine stops on the *ZKPF* immediately before the final control combination on the program tape. If the Reset button is then pressed, *C*27 will be read into the store, and will modify the orders in the specified check points of the program. As a result, when the program is started by the control combination at the end of the check-point tape, it will punch the extra symbols required.

Make-up of check-point tape

space	
PZT 400 *K*	
$\boxed{C27}$	
space	
P *Z*	
Q 160 *S*	*Q* will be punched each time control passes through 160*F*
D 72 *S*	*D* will be punched each time control passes through 72*F*
A 210 *S*	*A* will be punched each time control passes through 210*F*
π 110 *S*	Figure shift will be punched each time control passes through 110*F*
E 230 *KPF*	Transfers control to first order of master routine, with accumulator clear

*Application of checking subroutine C*31 *to Example* 2

A more detailed record of the course of action of the program of Example 2 can be obtained by the use of checking subroutine *C*31 to print a trace. For this, a checking tape should be prepared (see below) and inserted into the tape reader before pressing the Reset button after the machine has stopped on the *ZKPF* near the end of the program tape. When *C*31 has been read in, it will take control and start the program at the point specified by the final control combination on the checking tape. As each order of the program is executed, its function letter will then be punched on the output tape. Carriage return and line feed symbols will also be punched whenever a transfer of control occurs in the program.

Make-up of checking tape

space	
PZT 400 *K*	
G *K*	
T 45 *K*	Set *H*-parameter for *C*31, so that checking starts at the order in 230*F*
P 230 *F*	
$\boxed{C31}$	
E 230 *KPF*	Starts program at first order of master routine, with accumulator clear.

*Information printed by teleprinter**	*Corresponding orders*	
AUTAF	master routine	0 through 4
ATTSAUATAHNYTAG	*Q*3	0 through 14
ATHVAYTHVYTABF	auxiliary subroutine	0 through 13
F	*D*11	0
USAAE	*D*11	8 through 12
ASUSALUSAAE	*D*11	2 through 12
ASUSALUSAAEAG	*D*11	2 through 14
THNAYTNYG	*D*11	19 through 27
THNAYTNYG	*D*11	19 through 27
THNAYTNYG	*D*11	19 through 27
THNAYTNYGAF	*D*11	19 through 29
TE	auxiliary subroutine	14 and 15
HVYATAAG	*Q*3	15 through 22
TAHVYTAG	*Q*3	7 through 14
ATHVAYTHVYTABF	auxiliary subroutine	0 through 13
F	*D*11	0
USAAE	*D*11	8 through 12
ASUSALUSAAEAG	*D*11	2 through 14
THNAY etc.	*D*11	19 through 27

**Note:* Numerical results punched by *P*30 would be printed as the corresponding letters, since the teleprinter is set to letter shift by *P*30.

Example 3. Integration of an ordinary differential equation. The equation considered is

$$\frac{dy}{dx} = \frac{y(1 + 2y - 4x)}{x(y - x + A)},$$

where A is a constant. This equation occurs in theoretical astrophysics.

In the vicinity of the origin, a solution for any given value of A has the behavior $y = (Bx)^{1/A}$, where B is arbitrary. Solutions are required for a set of values of A and, for each value of A, for a set of values of B. Each solution is to be tabulated at an interval of 0.01 in x until either $y \geq 0.98$ or $dy/dx \leq -1$, values of y being correct to five decimals.

Method. The formula for dy/dx is formally indeterminate at $x = 0$. It is therefore necessary to start the numerical integration from some small value x_0 of x, at which the value y_0 of y can be evaluated from a series expansion. This starting point can be taken as $x_0 = 0.01$; the corresponding values of y_0, for different values of B, must be calculated separately, and furnished to the machine as input data. The program is so arranged that the machine evaluates automatically the whole set of solutions for a given value of A and different values of y_0.

An input subroutine is needed to read in the values of x_0 and A, and the values of y_0 for which solutions are required. Library subroutine $R37$ is used for this purpose. Two other library subroutines, $G12$ and $P31$, are used for carrying out the step-by-step integration and for printing the results, respectively. $G12$ requires an auxiliary subroutine which must be programmed in detail; it involves a division process, for which library subroutine $D11$ is used. The auxiliary subroutine is required to calculate, and place in $14D$, the quantity

$$2^m h \frac{dy}{dx} = 2^m h \frac{y(1 + 2y - 4x)}{x(y - x + A)}.$$

The quantities x and y are in the range (0,1) and, for the solutions required, A and $2^m h$ are less than 1. However, $4x$ may exceed unity, so we must introduce a scaling factor 2^{-2}, and calculate

$$2^m h \frac{y(\frac{1}{4} + \frac{1}{2}y - x)}{x[\frac{1}{4}(y - x) + \frac{1}{4}A]}.$$

The sequence of operations by which this quantity is evaluated must be planned with care, to ensure that all the intermediate quantities remain within the capacity of the accumulator. The method adopted is to multiply $2^m h$ by $y(\frac{1}{4} + \frac{1}{2}y - x)$, and divide the double-length product by $x\,[\frac{1}{4}(y - x) + \frac{1}{4}A]$.

Mathematical checks. It is necessary to verify that the interval h is small enough for the step-by-step integration process, and that the solution is stable despite rounding-off errors. Mathematical checks are not built into the program and must therefore be carried out separately. A suitable check is to repeat selected runs with a smaller value of h. Another check is to evaluate y', at selected points, from the differential equation and also from a central-difference formula. Results for one special case, namely $A = 0.4$, $x_0 = 0.074515$, $y_0 = 0.049793$, can be obtained from the tabulated solution of Emden's equation.

Allocation of storage space

Routine	Location of first order	Number of locations	Type
$G12$	56	49	Closed B
$D11$	105	30	Closed B
$P31$	136	61	Closed A
$R37$	198	34	Closed B
auxiliary subroutine	240	28	Closed B
master routine	270	40	
H-sequence	315	3	
$R30$	1014	10	Special

Location of constants and variables

$10D$	y	$18D$	$2^m q_1$
$12D$	x	$20D$	$2^m q_2$
$14D$	$2^m hy'$	$22D$	A
$16D$	$2^m h$	$24D$	x_0

H-sequence

0	K 3441 F	$\simeq 0.98$
1	L F	Represents * in output code
2	R F	$4 \cdot 2^{-4} = \frac{1}{4}$

Master routine. This calculates and prints the values of y at the end of each interval until either $y > 0.98$ or $dy/dx < -1$. The integration then stops and the final value of x is printed on a new line, followed by an asterisk.

```
             G       K
start →  0   B       θ   ⎤ Call in R37                         ⎤
         1   F  198  F   ⎦                                     ⎥
R37 →    2   T   16  D     Plant $2^m h$ in 16D                ⎥
         3   B    3  θ   ⎤ Call in R37                         ⎥ Read constants from
         4   F  198  F   ⎦                                     ⎥   end of program
R37 →    5   T   24  D     Plant $x_0$ in 24D                  ⎥   tape
         6   B    6  θ   ⎤ Call in R37 to read A               ⎥
R37 →    7   F  198  F   ⎦                                     ⎥
         8   R    1  F   ⎤                                     ⎥
         9   Y       F   ⎥ Plant $\frac{1}{4}A$ in 22D         ⎦
        10   T   22  D   ⎦
15 →    11   Z       F     Wait while data tape is inserted
39 →    12   T  195  F     Reset layout counter (i.e., clear 60th location
                             of P31)
        13   B   13  θ   ⎤ Call in R37 to read $y_0$                   ⎤
        14   F  198  F   ⎦                                             ⎥ Prepare
R37 →   15   G  11π  θ     Jump to stop order if $y_0$ is negative     ⎥   for in-
        16   T   10  D     Place $y_0$ in 10D                          ⎥   tegra-
        17   A   24  D   ⎤ $x_0$ to 12D                                ⎥   tion
        18   T   12  D   ⎦                                             ⎦
        19   T   18  D   ⎤ Clear $q_1$ and $q_2$ for integration
        20   T   20  D   ⎦
32 →    21   B   21  θ   ⎤ Call in G12 to carry out one step of integration
        22   F   56  F   ⎦
G12 →   23   H   10  D     y to multiplier register
        24   A   24  θ   ⎤ Call in P31 to print y
        25   F  136  F   ⎦
P31 →   26   A   14  D     $2^m hy'$
        27   A   16  D     $2^m h$
        28   G   33  θ     End integration if $+1 < -y'$
        29   T       F
        30   A   10  D   ⎤
        31   S       H   ⎥ Test if $y > 0.98$
        32   G  21π  θ   ⎦
28 →    33   T       F
        34   T  195  F     Reset layout counter of P31
        35   H   12  D     Prepare to print $x_n$
        36   A   36  θ   ⎤ Call in P31 to print $x_n$ on a new line
        37   F  136  F   ⎦
        38   O    1  H     Print asterisk
        39   F   12  θ     Repeat for a new value of $y_0$
```

Auxiliary subroutine. This is a closed B subroutine which evaluates, and places in $14D$, the quantity $2^3h(dy/dx)$, where

$$\frac{dy}{dx} = \frac{y(\frac{1}{4} + \frac{1}{2}y - x)}{x[\frac{1}{4}(y - x) + \frac{1}{4}A]}.$$

```
              T       Z
G12 →   0  |  A   10  D   ⎤
        1  |  S   12  D   ⎥
        2  |  R    1  F   ⎥  $\frac{1}{4}(y - x) + \frac{1}{4}A$ to 0D
        3  |  A   22  D   ⎥
        4  |  Y       F   ⎥
        5  |  T       D   ⎦
        6  |  H       D   ⎤
        7  |  V   12  D   ⎥  Denominator to 4D
        8  |  Y       F   ⎥
        9  |  T    4  D   ⎦
       10  |  A   10  D   ⎤
       11  |  R       D   ⎥
       12  |  S   12  D   ⎥  $(\frac{1}{2}y - x + \frac{1}{4})$ to 0D
       13  |  A    2  H   ⎥
       14  |  Y       F   ⎥
       15  |  T       D   ⎦
       16  |  H       D   ⎤
       17  |  V   10  D   ⎥  $y(\frac{1}{2}y - x + \frac{1}{4})$ to 0D
       18  |  Y       F   ⎥
       19  |  T       D   ⎦
       20  |  H       D   ⎤  Numerator
       21  |  V   16  D   ⎦
       22  |  K   25  θ      Store content of B-register
       23  |  B   23  θ   ⎤  Call in D11
       24  |  F  105  F   ⎦
           ----------------
D11 →  25  | (P       F)     Restore content of B-register for link order
       26  |  T   14  D      Place $2^3hy'$ in 14D
       27  |  FS   2  F      Link order
           ----------------
```

Make-up of program tape

$\boxed{R30}$	Placed in 1014–1023	
space		
PZT 56 *KGK*		
T 45 *K*		Set parameters *H*, *N*, *M*, Δ, *L*, *X* for *G*12
P 10 *D*	Location of y_0	
P 14 *D*	Location of $2^m hy_0'$	
P 18 *D*	Location of q_0	
P 4 *F*	2 variables	
P 1 *F*	$m = 3$	
P 240 *F*	Location of auxiliary subroutine	
$\boxed{G12}$		
space		
PZT 105 *K*		
$\boxed{D11}$		
space		
PZT 136 *KGK*		
T 45 *K*	Set *H*- and *N*-parameters for *P*31, so that numbers are printed to five digits in five columns	
P 1024 *F*		
P 25 *F*		
$\boxed{P31}$		
space		
PZT 198 *K*		
$\boxed{R37}$		
space		
PZT 240 *KGK*		
T 45 *K*		
P 315 *F*	*H*-parameter for auxiliary subroutine and master routine	
$\boxed{\text{auxiliary subroutine}}$		
space		
PZT 270 *K*		
$\boxed{\text{master routine}}$		
space		
PZT 315 *K*		

(*continued*)

$\boxed{H\text{-sequence}}$	
E 270 *KPF*	
08+	$2^m h$
01+	x_0
4+	A

Note: The data tape, which is also needed, contains the sequence of values of y_0 for which solutions are required. These values are punched in the form required for reading by *R*37. The last value is followed by 1−, which stops the program.

Example 4. Evaluation of a Fourier transform. *Statement of problem.* The sums y_n are to be calculated to five decimal places, for $n = 1, 2, \ldots,$ 47. y_n is defined as

$$y_n = \sum_1^{47} x_m \sin\left(\frac{nm\pi}{48}\right),$$

where x_m, $(m = 1(1)47)$, is a group of fractions, each less than $\frac{1}{2}$, read from a data tape.

Method. $\sin(nm\pi/48)$ takes only 96 values, so that it is better to store a table of 96 entries than to evaluate each sine by means of a subroutine. The table is constructed, during a preliminary calculation, using the addition formula for sines and cosines. During the main calculation, the x_m are read into the store by means of subroutine *R*33, and tables of $x_m + x_{48-m}$ and $x_m - x_{48-m}$ are calculated. The following summation formulas can then be used:

$$y_n = \sum_1^{23} x_m \sin\frac{nm\pi}{48} + \sum_1^{23} x_{48-m} \sin\frac{(48 - m)}{48} n\pi + x_{24} \sin\frac{n\pi}{2},$$

that is,

$$y_n = \sum_1^{24} (x_m + x_{48-m}) \sin\frac{nm\pi}{48} \qquad (n \text{ odd}),$$

or

$$\sum_1^{24} (x_m - x_{48-m}) \sin\frac{nm\pi}{48} \qquad (n \text{ even}),$$

where x_{24} has been redefined to have half its former value. To ensure that numbers remain within range, $\frac{1}{10}y_n$ is calculated and printed instead of y_n. This is done most conveniently by using a table of $\frac{1}{10}$ sin $(nm\pi/48)$ instead of sin $(nm\pi/48)$. The results are printed by means of subroutine $P31$.

Allocation of storage locations. It is convenient to make use of preset parameters to specify the locations in which certain sets of quantities are placed in the store.

x_m is stored in location $2(m - 1)H$ $(m = 1, 2, \ldots, 47)$,

$x_m + x_{48-m}$ is stored in location $2(m - 1)N$ $(m = 1, 2, \ldots, 24)$,

$x_m - x_{48-m}$ is stored in location $2(m - 1)M$ $(m = 1, 2, \ldots, 24)$,

$\frac{1}{10}\sin\frac{r\pi}{48}$ is stored in location $2r\Delta$ $(r = 0, 1, \ldots, 95)$,

pseudo-orders are stored in locations $0L - 5L$,

subroutine $R33$ is stored in locations $56 - 101$,

subroutine $P31$ is stored in locations $102 - 162$.

Values will not be finally assigned to the parameters associated with H, N, M, Δ until the program is nearly complete, but it will be assumed that the numbers $x_m + x_{48-m}$ and $x_m - x_{48-m}$ occupy consecutive sequences of locations in the store.

L-sequence

```
L   0 || P    94 F
    1 || V    96 Δ
    2 || V       Δ
    3 || HS   46 N
    4 || P    48 F
    5 || V   192 Δ
```

Master routine

		T		*Z*	
	0	*B*	10	*D*	Call in *R*33 to plant cos $(\pi/48)$, sin $(\pi/48)$, and $\frac{1}{10}$ in locations 10*D*, 12*D*, and 14*D*, respectively
	1	*A*	1	θ	
	2	*F*	56	*F*	
*R*33 →	3	*T*		Δ	$\frac{1}{10}$ sin 0, = 0, to 0Δ
	4	*B*	190	*S*	
17 →	5	*BS*	2	*F*	Construct table of sines and cosines
	6	*H*	12	*D*	
	7	*V*	14	*D*	
	8	*HS*	188	Δ	
	9	*V*	10	*D*	
	10	*Y*		*F*	
	11	*TS*	190	Δ	
	12	*N*	12	*D*	
	13	*H*	10	*D*	
	14	*V*	14	*D*	
	15	*Y*		*F*	
	16	*T*	14	*D*	
	17	*J*	5	θ	
76 →	18	*Z*		*D*	Wait for data tape
	19	*T*	161	*F*	Clear location 59 of *P*31 to start table of results with a new block
	20	*B*		*H*	Call in *R*33 to read x_m into location $(2m - 2)H$, $(m = 1, \ldots, 47)$
	21	*A*	21	θ	
	22	*F*	56	*F*	
*R*33 →	23	*A*	46	*H*	Replace x_{24} by $\frac{1}{2}x_{24}$
	24	*R*		*D*	
	25	*T*	46	*H*	
	26	*B*	48	*S*	Set counts in 31θ and 36θ to −48 and +46
	27	*K*	31	θ	
	28	*B*	48	*F*	
42 →	29	*BS*	2	*S*	
	30	*K*	36	θ	
	31	(*Z*		*F*)	
	32	*BS*	2	*F*	
	33	*K*	31	θ	
	34	*AS*	92	*H*	$x_m + x_{48+m}$ to location $2(m - 1)N$
	35	*U*		*D*	
	36	(*Z*		*F*)	
	37	*AS*		*H*	
	38	*US*		*N*	

(*continued*)

```
        39   S        D
        40   S        D     x_m - x_48+m to location 2(m - 1)M
        41   TS       M
        42   J     29 θ
        43   S        L
  75 →  44   A      1 L
        45   U     58 θ     Set V 2n Δ and P 2n F
        46   S      2 L
        47   U      9 F
        48   L     32 F     Shift left 14 places to test whether n is odd
        49   L     32 F       or even
        50   G     52πθ     Jump if n is odd
        51   M      4 L
  50 →  52   A      3 L
        53   T     57 θ     Set HS 46 N or HS 46 M according as n
                              is odd or even
        54   T        D     Clear 0D for sum
        55   B     48 S     Set b = -48
  69 →  56   BS     2 F
        57   (Z       F)
        58   (Z       F)    Becomes V 2n Δ   Add rth contribution
        59   A        D                        to sum in 0D
        60   Y        F
        61   φ        D
        62   A      9 F
        63   A     58 θ     Increase argument by nπ/48
        64   U     58 θ
        65   S      5 L
        66   G     69πθ     Modulo 2π
        67   A      2 L
        68   T     58 θ
  66 →  69   J     56 θ
        70   H        D     Place result in multiplier register
        71   A     71 θ     Call in P31 to print result
        72   F    102 F
 P31 →  73   A        F
        74   S        L     Test whether n = 47
        75   G     44 θ
        76   F     18 θ
```

The final allocation of storage space can now be made. The first order of the master routine will be placed in 200 and the following values will be given to the preset parameters:

```
H   P 300 D
N   P  96 H
M   P  48 N
Δ   P  48 M
L   P 280 F
```

Make-up of program tape

$\boxed{R30}$	Placed in locations 1014–1023	
space		
PZT 56 *K*		
$\boxed{R33}$		
space		
PZT 102 *KGK*		
T 45 *K*	Set *H*- and *N*-parameters for *P*31, so that numbers will be printed to 6 decimals in 6 columns	
P 512 *F*		
P 30 *F*		
$\boxed{P31}$		
space		
PZT 200 *KGK*		
T 45 *K*	Set parameters used by master routine	
P 300 *D*		
P 96 *H*		
P 48 *N*		
P 48 *M*		
P 280 *F*		
$\boxed{\text{master routine}}$		
space		
PZT 280 *K*		
$\boxed{L\text{-sequence}}$		
E 200 *KPF*		
space	*R*33 ignores blank tape automatically; *PZ* is therefore not necessary	
+99785892	cos (π/48)	Read in by *R*33 for constructing table of sines
+06540313	sin (π/48)	
+1*M*		

The data are supplied on separate tapes and are punched in simple sequences of 47 numbers, the last being terminated by M.

Notes: (1) The method used here is crude. A more sophisticated program would (a) check the numbers read with the aid of a check sum punched on the data tape, (b) use a faster and more complicated routine, (c) allow the number of coefficients (48) to be varied, and (d) include checks on the accuracy of calculation.

(2) There is little scaling required in this problem, but it has been assumed that max $|y_n| < 10$.

Example 5. Evaluation of a definite integral. *Statement of problem.* The following integral, which occurs in the theory of the ionization of an exponential atmosphere by solar radiation, is to be tabulated to seven decimal places as a function of χ and x, for $\chi = 90°\ (—1°)\ 50°$ and $x = 200(20)280$. The table is to be printed in five columns, each giving the values of the integral for fixed x.

$$\int_0^\chi \exp\left\{-x\frac{\sin\chi - \sin\lambda}{\sin\lambda}\right\} \operatorname{cosec}^2\lambda\, d\lambda.$$

Method. The integral is written in the following form, in which all quantities to be handled in the machine are numerically less than unity.

$$+8\int_{\chi/2}^{0} \exp\,(2^5 U)\cdot\frac{\operatorname{cosec}^2 2t}{4}\,dt,$$

where

$$U = \frac{\chi}{2^5}\cdot\left[\frac{\frac{1}{2}\sin 2t - \frac{1}{2}\sin\chi}{\frac{1}{2}\sin 2t}\right] \qquad \text{and} \qquad t = (\chi - \lambda)/2.$$

Although apparently $|U|$ and $\frac{1}{4}\operatorname{cosec}^2 2t$ may exceed 1, the integrand is negligibly small at these points and the program does not evaluate it.

The integration is performed by applying Gauss' 6-point formula (subroutine $Q3$) to a series of strips of equal width, starting with $t = 0$ and continuing until the integrand becomes so small that further contributions would be insignificant. The strip width is referred to as the interval of integration and denoted by $2h$. Since the optimum interval is difficult to estimate in advance, the program is arranged to perform a sequence of trial integrations starting with an interval known to be too large, and halving it after each integration. The process is terminated when two successive results, differing by less than $2^{12}\cdot 10^{-8}$, are obtained. Since the error depends on the 12th power of the interval, the last result to be obtained probably does not differ from the true value of the integral by more than about 10^{-8}.

To simplify the construction of the master routine, use is made of a separate subroutine, *B*, designed to evaluate the integral for a specified value of the interval $2h$. Adjustment of the interval is performed by the master routine. An auxiliary subroutine, *A*, for *Q*3 is also required. The following library subroutines are used: *E*6, *P*31, *T*7, *D*11.

Allocation of storage space

Routine	Location of first order	Number of locations occupied	Type
*R*9	56	15	Special
*D*11	72	30	Closed *B*
*E*6	102*	38	Closed *B*
*P*31	140*	61	Closed *A*
*T*7	202*	36	Closed *A*
*Q*3	238*	48	Closed *A*
A (aux)	300	40	Closed *A*
B	350	29	Closed *B*
master routine	400	36	
N-sequence	440	9	
H-sequence	450	20	

N-sequence

N 0	*P* 200 *F*	Initial value of $x \cdot 2^{-15}$	
1	(*P* 200 *F*)	Current value of $x \cdot 2^{-15}$	
2	*P* 20 *F*	Increment of $x \cdot 2^{-15}$	
3	*P* 300 *F*	Final value of $x \cdot 2^{-15}$	
4	*Q* *F*	2^{-4}; starting value of h (radians)	
5	*P* 90 *F*	Current value of $\chi \cdot 2^{-15}$	Where χ is expressed in degrees
6	*P* 50 *F*	Final value of $\chi \cdot 2^{-15}$	
7	*R* *F*	$4 \cdot 2^{-4}$	
8	*S* *F*	$12 \cdot 2^{-4}$	

* First order must be in an even-numbered storage location.

Locations used by Q3 and subroutine B for temporary storage (H-sequence—long locations)

$0\ H$	midpoint of interval
$2\ H$	h
$4\ H$	Used by $Q3$
$6\ H$	$\frac{1}{2}\sin\chi$
$8\ H$	Used by B
$10\ H$	$2^5\pi/180$ (conversion constant read by $R9$ as $2^{39}\pi/180$)

Locations used by master routine and subroutine A

$6\ D$	$\frac{1}{2}$ cosec $2t$ (subroutine A)
$8\ D$	$\frac{1}{2}(\sin 2t - \sin\chi)$ (subroutine A)
$10\ D$	Used in master routine
$12\ D$	Indicator; see below

When $Q3$ is called in by subroutine B, $12D$ contains $-2h$, and this quantity remains there when the auxiliary subroutine, A, is called in by $Q3$. If $2^5U \geq -24$, $C(12D)$ is unaltered by A, but if $2^5U < -24$, so that $\exp(2^5U)$ is small enough to make the integrand negligibly small, $C(12D)$ is replaced by zero. When control finally returns to subroutine B, the sign of $C(12D)$ is tested. If the sign is negative, a further step of integration is performed; otherwise, the integration is terminated.

Master routine

```
                 G       K
start →   0      A       N   ]  Set x = 200
          1      T     1 N   ]
   29 →   2      A     4 N   ]  Set initial value of h = 2^-4
          3      T     2 H   ]
          4      B     4 θ   ]  Call in subroutine B to evaluate integral
          5      F   350 F   ]
aux, 21 → 6      T    10 D      Store result in 10D
          7      A     2 H   ]
          8      R       D   ]  Halve interval
          9      T     2 H   ]
         10      B    10 θ   ]  Call in subroutine B to evaluate integral
         11      F   350 F   ]
   B →   12      U     4 D      Store new value in 4D
         13      S    10 D      Subtract previous value
         14      G    17 θ   ]  Form negative absolute value of differ-
         15      T       D   ]    ence
         16      S       D   ]
```

(*continued*)

```
 14 → 17 | A   2 F     | Add comparison value $2^{-15} \doteqdot 2^{12} \cdot 10^{-8}$
      18 | E  22 $\pi\theta$ | Jump if sufficiently accurate
      19 | T     F     | Clear accumulator
      20 | A   4 D     ] Store new value for comparison and
      21 | F   6 $\theta$     ]   repeat
 18 → 22 | H   4 D     |
      23 | A  23 $\theta$     ] Call in $P31$ to print result
      24 | F 140 F     ]
P31 → 25 | A   1 N     ]
      26 | A   2 N     ] Increase $x$
      27 | U   1 N     ]
      28 | S   3 N     ] Jump to repeat until final value is
      29 | G   2$\pi\theta$    ]   reached
      30 | A   5 N     ]
      31 | S   2 F     ] Reduce $\chi$
      32 | U   5 N     ]
      33 | S   6 N     ] Jump to repeat until final value reached
      34 | E    $\pi\theta$    ]
      35 | Z     F     | Stop
```

Subroutine B. Evaluates integral, using $Q3$, with given interval $2h$.

```
         | G     K     |
Master → 0 | K  29 $\theta$     | Store $b$ for link order
       1 | T   8 H     |                          ]
       2 | H   5 N     | $\chi \cdot 2^{-15}$               ]
       3 | V  10 H     | $(\pi/180) \cdot 2^5$      ] Convert $\chi$ from degrees to
       4 | L 128 F     |                          ]   radians
       5 | Y     F     |                          ]
       6 | U   4 D     | $\chi/2$                    ]
       7 | S   2 H     | $h$ ] Obtain midpoint of first interval
 26 →  8 | T     H     |   ]
       9 | A   9 $\theta$     | Call in $T7$ to form $\frac{1}{2} \sin \chi$
      10 | F 202 F     |
 T7 → 11 | A   4 D     ] Store $\frac{1}{2} \sin \chi$ in $6H$
      12 | T   6 H     ]
      13 | T   8 H     | Clear $8H$
      14 | S   2 H     ] Put $-2h$ in $12D$. Subroutine $A$ will re-
      15 | S   2 H     ]   place $C(12D)$ by zero when the inte-
      16 | T  12 D     ]   grand becomes small
      17 | A  17 $\theta$     ] Call in $Q3$ to integrate over one strip
      18 | F 238 F     ]
```

(*continued*)

```
Q3 → 19 | A       D  ]
     20 | A     8 H  ] Add contribution to sum
     21 | T     8 H  ]
     22 | A    12 D  ] Test C(12D) to find out if integrand is
     23 | E   27πθ   ]   small
     24 | A       H  ] Reduce argument by 2h before evalu-
     25 | T       H  ]   ating next strip
     26 | F     8 θ
        |----------
23 → 27 | A     8 H  ] Add total integral to accumulator
     28 | L     2 F  ]
     29 | (P      F)
     30 | FS    2 F    Link order
        |----------
```

Auxiliary subroutine, A

```
        | T       Z
Q3 →  0 | A     3 F  ] Plant link order
      1 | T    39 θ  ]
      2 | A       D  ] t to 4D
      3 | T     4 D  ]
      4 | A     4 θ  ] Call in T7, ½ sin 2t to 4D
      5 | F   202 F  ]
        |----------
T7 →  6 | A     4 D    ½ sin 2t
      7 | S     6 H    ½ sin χ
      8 | T     8 D    ½ (sin 2t − sin χ) to 8D
      9 | A     7 N    ¼
     10 | B    10 θ  ] Call in D11, form ½/sin 2t
     11 | F    72 F  ]
        |----------
D11 → 12 | H     8 D   ½ (sin 2t − sin χ)
     13 | T     6 D    ½/sin 2t
     14 | V     6 D
     15 | Y       F
     16 | T       D    1  sin 2t − sin χ
                       -  ------------
                       4     sin 2t
     17 | H       D
     18 | V     1 N    x · 2^-15
     19 | L    16 F  ] Multiply by 2^12, giving U
     20 | L    16 F  ]
     21 | Y       F
     22 | U       D
     23 | A     8 N  ]
     24 | E   27πθ   ] Test whether U < −12 · 2^-4; if so,
     25 | T    12 D  ]   make C(12D) = 0
     26 | T    12 D  ]
```

(*continued*)

24 → 27	*H* *D*	
28	*B* 28 *θ*	Call in *E*6
29	*F* 102 *F*	
30	*T* *D*	Exp 2^5U to 0*D*
*E*3 → 31	*H* 6 *D*	
32	*V* 6 *D*	
33	*Y* *F*	
34	*T* 4 *D*	$\frac{1}{4}/\sin^2 2t$
35	*H* 4 *D*	
36	*V* *D*	Exp 2^5U
37	*Y* *F*	
38	*T* *D*	
39	(*P* *F*)	Becomes link order

Make-up of program tape

$\boxed{R30}$	Placed in locations 1014–1023
space	
P *Z*	
$\boxed{R9}$	Placed in locations 56–70
space	
PZT 72 *K*	
$\boxed{D11}$	
space	
PZT 102 *KGK*	
T 45 *K*	
P 5 *F*	*H*-parameter for subroutine *E*6
$\boxed{E6}$	
space	
PZT 140 *KGK*	
T 45 *K*	
P 1664 *F*	*H*-parameter for digit layout: five digits, space, two digits
P 25 *F*	*N*-parameter for 5 columns
$\boxed{P31}$	
space	
PZT 202 *K*	
$\boxed{T7}$	
space	

(*continued*)

```
PZT 238 K
  G     K
  T  45 K
  P 450 D        H-parameter; location of arguments for Q3
  P 300 F        N-parameter; location of auxiliary subroutine A
Q3
space

PZT 300 KGK
  T  46 K
  P 440 F        N-parameter; location of constants for auxiliary
                   subroutine, and master routine

auxiliary
subroutine A
space

PZT 350 K
subroutine B
space

PZT 400 K
master
routine
space

PZE 69 KT 10 H
9595049034π      = 2^39π/180 (conversion constant)
  T 440 K
N-sequence
space

PZZKPF           Stops machine until Reset button is pressed
E 400 KPT        Transfers control to 400, i.e., to first order of
                   master routine
```

Notes: (1) Since only one long constant, $2^5\pi/180$, is needed, the use of a subroutine to read fractions is avoided by using *R*9 to read it as the integer $2^{39}\pi/180$.

(2) The scale factors used for x and χ are such that increments can be expressed exactly, and rounding-off errors do not accumulate.

(3) The whole program consists of 335 orders, of which 107 have been specially written; the remaining 228 are provided by library subroutines.

(4) In integrations of this kind it is impossible to estimate the optimum interval by any theoretical argument. Experience has shown that unless

some form of automatic step adjustment is used, a good deal of time is likely to be wasted in arriving, by trial and error, at optimum intervals for the various values assumed by the argument. The penalty paid for using, as a safety measure, an interval which is much smaller than the optimum is that the time taken to perform the calculation is much longer than it need be. However, the method of automatic step adjustment used in this example is somewhat crude, since the optimum interval is determined afresh for each value of x and χ, information available from previous integrations being ignored.

CHAPTER 8

AUTOMATIC PROGRAMMING

8–1 Introduction. The subject matter of this chapter might be described as "programming for programming's sake." It concerns various methods whereby a machine can be made to help with the task of drawing up its own program. Naturally, a machine can help only with those tasks which can be reduced to precise rules. These at present include everything concerned with assembling the various subroutines and other sections of the program, and with providing the necessary cross referencing; they exclude anything that can be properly called Numerical Analysis, although automatic methods of writing programs for evaluating quantities given by explicit formulas have been developed, and we shall say something about them in Section 8–5.

There has been much controversy about the value of the methods discussed in this chapter. From the time when its possibilities were first fully appreciated, automatic programming has always had its enthusiasts. On the other hand, there have been those who have felt that its advantages were more apparent than real. Some went so far as to assert that programmers should be compelled to write orders in a form as near as possible to that which they take inside the machine, and that attempts to make the machine assist with the clerical tasks of programming lead only to a wasteful dissipation of effort. Our experience with the EDSAC led us from the beginning to reject this extreme view. It has always been open to workers with that machine to include at the beginning of their program tapes a short routine which would replace the standard initial input routine by a simple input routine designed to read programs punched in direct binary form. No one, however, has ever done this, and in fact the facilities provided by the initial input routine have always been taken for granted and used by everybody. The controversy may now be said to have died down and it is generally agreed that the work of programmers can be much facilitated by the use of such techniques as are described in this chapter. However, there still remains scope for discussion as to how elaborate the system used in conjunction with any particular installation should be.

Initial input routines designed to provide the relatively elaborate facilities with which we are concerned in this chapter will generally be referred to as *conversion routines.* The first comprehensive conversion routine, for reading programs in which the orders are punched in forms very different from the internal binary form they ultimately take inside the machine, was developed under the direction of C. W. Adams at M. I. T. Equally extensive schemes following somewhat different lines have been developed under the direction of Dr. Grace Hopper at the Sperry Rand Corporation. Readers who are interested in carrying the study of the subject further than is done in this chapter should consult the reports of these organizations, some of which are listed in the bibliography.

8–2 Conversion versus interpretation. A number of schemes designed to simplify programming have been based on the use of interpretive subroutines (see Section 2–22). Such schemes can provide the programmer with an order code which is entirely different from the basic order code of the machine; for example, he can be provided with a three-address code even though the machine itself may have a single-address code. Although such schemes have enjoyed a certain vogue, they suffer from two severe disadvantages. In the first place, the time taken to interpret orders is appreciable, with the result that the over-all effective speed of the machine is reduced very considerably. In the second place, the interpretive subroutine, or a large part of it, must remain permanently in the high-speed store during the execution of the program, with the result that the amount of high-speed storage space available to the programmer is reduced. In view of these disadvantages, it would appear that interpretive methods of facilitating programming are of real value only when applied to relatively simple problems in which the total running time is short. Interpretive methods, however, have an application in making one machine simulate another, for example, when it is desired to develop subroutines and programs for a new machine in advance of that machine being completed.

Conversion routines, as that term is used in this book, do not suffer from the disadvantages mentioned above, since the program, as written, is converted to a sequence of basic machine orders once and for all when it is first read into the machine. Once converted, the program runs like any other program, and the conversion routine is no longer needed. If desired, the converted program can be taken out of the machine and put back on subsequent occasions without further conversion.

8–3 Assembly of a program. A simple illustration will now be given of the way in which a machine can be made to help with the clerical tasks involved in drawing up a program. A program is composed of a master routine and a number of subroutines and, in the ordinary way, the programmer must decide where these are to go in the store and provide the necessary cross referencing between them; for example, he must insert the correct addresses in those orders in the master routine which call in the subroutines. Use of terminal code letters enables him to defer until a later stage the decision as to where the various subroutines shall go, but he still has, ultimately, to make the decision himself.

The assembly subroutine given below allows the assembly of the master routine and the subroutines to be performed automatically by the machine. The assembly subroutine must be in the store when the program tape is read. The master routine is given the identification number 0 and the subroutines the numbers 1, 2, 3, etc. In what follows the word "routine" will be used to indicate either the master routine or a subroutine.

In front of each routine on the program tape are punched control combinations which call in the assembly subroutine. Following these control combinations, and immediately in front of the routines themselves, are labels in the form *T* ϕ, *T* 1 ϕ, *T* 2 ϕ, etc., giving the number of the routine. ϕ is a terminal code letter (not normally used by programmers) to which storage location 44 corresponds, and the assembly subroutine establishes a *directory* composed of *F*-orders in storage locations 0 ϕ, 1 ϕ, 2 ϕ, etc. This directory remains in the store during the subsequent execution of the program. When one of the routines, say the third, is called in, control is sent first to storage location 3 ϕ, and thence (by the *F*-order planted by the assembly subroutine) to the first order of the routine. Thus orders calling in the routines must be punched *F* ϕ, *F* 1 ϕ, *F* 2 ϕ, etc.

Many subroutines in the EDSAC library must be placed in the store with their first orders in even-numbered locations, and a practical assembly subroutine designed for use with the EDSAC would need to take account of this fact. For purposes of illustration, however, we shall assume that subroutines can be placed in the store with their first orders in any location, odd or even. An assembly subroutine constructed on this understanding is given below; it is entered with the label of the next routine to be read from the tape in the accumulator.

```
      T    Z
 0    T   5 θ     Plant C(Acc) (label) in 5θ
 1    A  42 F   ] Reinstate Transfer Order destroyed when assembly
 2    A   8 θ   ]   subroutine was entered
 3    U  22 F   ]
 4    M   7 θ   ] Form and plant in directory an order F n F, where
 5   (Z     F)  ]   n is address specified in Transfer Order
 6    F  34 F     Return control to initial input routine
    ------------
 7  ‖ F     F
 8  ‖ T     F
```

The following is an example of how a program tape would be punched for use with the assembly subroutine just described. It is assumed that the program consists of a master routine and two subroutines. The assembly subroutine is placed in the store with its first order in location 100, and the directory starts in location 120. The program itself goes into the store from location 200 onwards. Note that the control combination *E* 100 *K T n* ϕ sends control to 100 with *T n* ϕ in the accumulator.

```
P       Z
T  100  K
G       K
T   44  K
P  120  F
assembly subroutine
space

P       Z
T  200  K              Program to start at 200
G       K              Call in assembly subroutine
E  100  K
T       ϕ              Label for master routine
Master routine
space

P       Z
G       K
E  100  K              Call in assembly subroutine
T    1  ϕ              Label for subroutine No. 1
subroutine no. 1
space

P       Z
G       K
E  100  K              Call in assembly subroutine
T    2  ϕ              Label for subroutine No. 2
subroutine no. 2
E   25  K              Send control to master routine via directory
E       ϕ                (See Appendix 4)
P       F
```

It should be noted that although the directory must remain in the store during the execution of the program, the assembly subroutine itself may be overwritten by the last subroutine to go into the store.

8–4 Floating addresses. In addition to cross references between the master routine and subroutines, these routines themselves contain many internal cross references to orders and pseudo-orders. In a machine which takes its orders sequentially from the store, orders and pseudo-orders are referred to by the addresses of the storage locations in which they stand. The various jump orders, orders which modify other orders, and orders which refer to pseudo-orders as operands, make a subroutine into a very closely knit entity, and it is not normally possible to insert extra orders into it, or to make any modification, however trivial, without some renumbering becoming necessary. Not only is this renumbering a nuisance, but it is also a source of mistakes whenever changes are made in a program. In the system now described the programmer refers to orders and pseudo-orders in the program, not by the addresses of the locations in which they stand, but by labels arbitrarily assigned to them. These labels are referred to as *floating addresses*, and may be written in orders instead of absolute addresses. When the program comes to be read into the machine, the initial input routine or conversion routine replaces the floating addresses by absolute addresses. This is a process which cannot generally be completed until the whole program has been scanned, since some orders punched near the beginning of the program may refer to other orders or pseudo-orders punched near the end, and the latter will not have been allocated addresses when the former are first read.

One method of proceeding is to arrange that the program shall be read twice. During the first reading the orders are not placed in the store, but are counted, and a list is formed of the locations into which the various labelled orders and pseudo-orders will eventually go. During the second reading the orders are placed in the store, and floating addresses, where they occur, are replaced by absolute addresses from the list previously formed. Reading a program tape twice would be a tiresome procedure, but the same effect can be achieved by first reading the unconverted program into an auxiliary store, and then transferring it to the main store as required. An alternative procedure is to keep a record, during input, of those orders which must have absolute addresses inserted into them, and to carry out the necessary substitutions after the input of the program. A third alternative is for absolute addresses to be inserted into orders to replace floating addresses as soon as the absolute addresses are determined, the reading of the program being halted for this purpose. If the absolute address is known when an order containing a floating address is read, the substitution takes place at once. A conversion routine constructed along these latter principles is given below.

It is possible to establish, with the aid of sufficiently complex conversion routines, very convenient and elegant systems of programming. These may provide not only for the use of floating addresses, but also for the automatic incorporation of subroutines. They may also provide powerful facilities for error diagnosis; this may well be of some importance since, if a program is written in a form very different from that which it takes inside the machine, many of the ordinary methods used for error diagnosis may fail.

We shall content ourselves in this chapter with illustrating one of the ways in which a conversion routine for providing floating address facilities can be constructed. The conversion routine given is.intended for purposes of illustration, rather than for practical use. Readers who desire further information on the subject should consult the papers and reports given in the bibliography.

The conversion routine given below works in conjunction with the initial input routine of the EDSAC. When read into the store it causes a jump order to be planted in storage location 32, and this causes control to be transferred to the conversion routine when certain symbols are read. Floating addresses are indicated by having an asterisk punched after them. This symbol, which does not appear in the ordinary input code of the EDSAC, is here used to denote a row of blank tape. When floating addresses are used as labels they are prefixed by the letter *L*.

The following example shows how a program for use in conjunction with the conversion routine would be punched. The program is designed to illustrate the use of floating addresses; it is not the most efficient that could be devised.

Example. Form and print $\sum_{j=0}^{99} a_j$ where $a_j = C(400 + j)F$, given a print routine which prints $C(R)$ and starts at 50*.

```
      T 200   K
      T       F     Clear 0F initially
      S    5*     ] Set count in 4F
L3*   T    4  F   ]
      A    1*     ] Increase address in A-order
      A    2  F   ] Note: C(2F) = P 1 F.
      T    1*     ]
      A       F   ]
L1*  (A  399  F)  ] Add contribution to sum in 0F
      T       F   ]
      A    4  F   ]
      A    2  F   ] Count
      G    3*     ]
      H       F
L6*   A    6*     ] Print
      F   50*     ]
      Z       F     Stop
L5*  ||P 100  F
```

The conversion routine is given below. As each label is read, a record of the corresponding absolute location is made in a list whose first entry is in $0H$. Whenever a floating address is encountered in an order, the list is consulted and, if the corresponding absolute value has already been placed there, this absolute value is substituted for the floating address. If no entry has been made in the list the order is placed in the store with a zero address, and the number of the location in which it is placed is temporarily recorded in the list. Subsequently, when the corresponding label is encountered, the existence of this temporary record enables the absolute address to be inserted in the order; at the same time the temporary entry in the list is replaced by the absolute address. It may happen that the same floating address occurs in a number of orders which are read before the corresponding absolute address has been determined. The first of these is treated in the way just described, being placed temporarily in the store with a zero address and a record of where it has gone being inserted in the list. When the second order with the same floating address is encountered, this is also placed in the store, but use is made of its address section to hold the quantity which was previously recorded in the list. The number of the location into which this second order has gone is in its turn placed temporarily in the list. The same procedure is followed as each successive order with the same floating address is encountered, so that, when the corresponding label is read and the absolute address determined, there exists a "chain" of references which enables the absolute address to be incorporated in all the orders which require it. At the outset it is assumed that the storage locations to be occupied by the directory are clear.

When a floating address, punched as m^*, has been read, the conversion routine is entered at order 33 with m in storage location 0. If the floating address occurs in a label, storage location 40 contains L. If the floating address occurs in an order, this location contains the function letter of that order; this function letter cannot be L, since a floating address cannot occur in a left shift order.

Conversion routine

```
      T       Z
 0    H    29 θ  ┐
 1    C       F  │  Form m (floating address with-
 2    A    26 θ  │     out π digit)
 3    T     4 θ  ┘
 4   (P       F)    Becomes B  m  F
 5    A    40 F  ┐
 6    A    12 θ  │  Test and jump if not label
 7    F    30πθ  ┘
 8    AS      H  ┐  Extract table entry i, say,
 9    T     1 F  ┘     from m H and place in 1 F
```

(*continued*)

```
                        10    A      22 F     P n F to 0 F, where n is abso-
                        11    S       8 F       lute address
                        12    U         F
                        13    A      39 θ     C F
                        14    TS        H          C n F to m H
                        15    A       1 F
                        16    F      25 θ
                   25 → 17    A      26 θ     B F
                        18    T      19 θ
                        19    (P        F)    Becomes B i F
                        20    HS        F
                        21    C      42 θ     Insert absolute address in next
                        22    A         F       order of "chain"
                        23    TS        F
                        24    C      29 θ     Use former address in order to
                   16 → 25    F     17πθ        find next order in chain
                        26    B         F     Clear B-register
                        27    H       8 F     Restore C(R)
                        28    F      34 F     Return to initial input routine
                        29    || P 1023 F
                    7 → 30    MS        H     Jump if C n F, i.e., if absolute
                        31    G      37 θ       address already inserted in
                                                table
                        32    T       1 F     Plant former table entry in
                        33    A      22 F       1 F and new table entry in
                        34    S       8 F       m H
                        35    TS        H
                        36    A       1 F
                   31 → 37    M      41 F     Clear function digits
                        38    H      43 F     2^-16  add π-digit if present
Enter when              39    C         F            in order
blank tape              40    H       8 F     Restore C(R) and return to ini-
read at end             41    F      21 F       tial input routine
of floating             42    || V 1024 D     ≡ 11111000000000001
address or            → 43    E        πθ
when nega-              44    A      22 F     Return to initial input routine
tive control            45    F      33 F       if G K or similar negative
combination                                     control combination is read
such as G K                   T      32 K     Modify initial input routine by
is read                32F    F     43πθ        placing F 43 πθ in 32 F
                              T      46 Z
```

The following table shows how conversion of the program for the example given on p. 131 would proceed. The second and third columns refer to words which will, if not overwritten, form part of the converted program, while the fourth and fifth columns similarly refer to entries in the list of absolute addresses.

Characters read from tape	Words placed in store			
	word	location	word	location
T *F*	*T* *F*	200		
S 5*	*S* *F*	201	*P* 201 *F*	5*H*
L 3*			*C* 202 *F*	3*H*
T 4 *F*	*T* 4 *F*	202		
A 1*	*A* *F*	203	*P* 203 *F*	1*H*
A 2 *F*	*A* 2 *F*	204		
T 1*	*T* 203 *F*	205	*P* 205 *F*	1*H*
A · *F*	*A* *F*	206		
L 1*	*T* 207 *F*	205	*C* 207 *F*	1*H*
	A 207 *F*	203		
A 199 *F*	*A* 199 *F*	207		
T *F*	*T* *F*	208		
A 4 *F*	*A* 4 *F*	209		
A 2 *F*	*A* 2 *F*	210		
G 3*	*G* 202 *F*	211		
H *F*	*H* *F*	212		
L 6*			*C* 213 *F*	6*H*
A 6*	*A* 213 *F*	213		
F 50*	*F* *F*	214	*P* 214 *F*	50*H*
Z *F*	*Z* *F*	215		
L 5*	*S* 216 *F*	201	*C* 216 *F*	5*H*
P 100 *F*	*P* 100 *F*	216		

It is assumed that an absolute value will be assigned to the floating address 50* at a later stage when the print routine is read into the machine.

Conversion routines can be designed to provide a number of facilities additional to those provided by the simple one just given. One obvious extension is to provide for the explicit setting of floating addresses, as well as for their implicit setting by means of labels; for example, it might be possible to punch 5* = 50 to indicate that the absolute address corresponding to 5* is 50. Another very useful facility is the automatic listing of constants required in the program. Suppose, for example, that at some stage in the program it were required to add 0.59574 to the number in the accumulator; the programmer might punch A.59574 on the input tape, and the conversion routine would cause the constant 0.59574 to be placed in the first unused location of a sequence of locations set aside for storing such constants, and would cause an A-order, containing the address of that location, to be placed in the converted program. Thus, if the sequence of locations were to start at 0N and the constant were to go into 6N, the order A 6 N would go into the program. There must be some means by which the conversion routine can distinguish a constant from an ordinary address. In the above example the presence of a decimal point enables this to be done, but if necessary an extra symbol could be introduced for the purpose.

A comprehensive conversion routine should contain facilities for reading and converting numbers, so that the use of an input subroutine can be avoided, so far as the input of constants along with the program is concerned. In the case of a binary machine, it is convenient if powers of 2 and 10 can be punched after the numbers, and treated by the conversion routine as multiplying factors. A number might then be punched in the following form: $.59574\ 10^3\ 2^{-10}$. The punching of indices may present difficulties because of the limitation of the number of distinct characters which can be punched on the tape; a device we have used successfully in these circumstances is to arrange for 10 and 2 to appear as suffixes, so that the above number would be punched as $.59574_{10}3_{2}-10$. A further refinement is to provide for the reading of numbers punched as fractions, for example in the form 136/137.

If the library of subroutines is recorded on magnetic tape, the conversion routine can be designed to incorporate open or closed subroutines as required in a program. Very short open subroutines containing perhaps no more than a few orders in all may be held in the high-speed store along with the conversion routine itself, and copied into the program when required. In this way, operations not included among the basic operations performed by the machine, for example the formation of a modulus or the multiplication and division of complex numbers, can be handled by the programmer as easily as if they were so included. This method of extending the order code of a machine is, in principle, as flexible as the use of interpretive subroutines, and since it does not bring about as great a reduction in effective machine speed, its adoption is to be preferred wherever possible. However, a serious restriction on its use is imposed by the fact that the converted program may contain a very large number of orders. This is particularly so in the case of floating-point operation, where the use of conversion rather than interpretation would otherwise be of great value.

In the case of the majority of fixed point machines there does not seem to be any reasonable alternative to the use of interpretive subroutines, or some equivalent procedure, for providing floating-point facilities when these are required.

It is desirable that the rules for punching a program should be as straightforward and as easily memorized as possible. In practice, this almost certainly means that a program as punched contains more characters than are absolutely necessary. This redundancy may be put to good use if the conversion routine is designed to examine the program for any forbidden sequences of characters, and to cause the machine to "report" if any occur. The "report" may simply take the form of a machine stoppage with the lighting of an indicator lamp, or it may include the printing of diagnostic information for the assistance of the programmer.

8–5 Formula recognition. We have seen that by the use of a more or less complicated conversion routine it is possible to make a computer recognize special symbols punched on tape or cards, and to react in a manner arbitrarily laid down. From this it is but a step to suggest that the machine might accept formulas written in ordinary mathematical notation, and punched on a specially designed keyboard perforator. This would appear at first sight to be a very significant development, promising to reduce greatly the labor of programming. A number of schemes of formula recognition have been described or proposed, but on examination they are found to be of more limited utility than might have been hoped.

The difficulty about making machines accept problems stated in mathematical language is that all such statements do not lend themselves to ready translation into a program. A program contains an explicit definition of a *process* which the machine must carry out; a mathematical statement of a problem also may contain a definition of a process, but it is more likely to consist of a series of statements about the relationships between certain quantities, some known and some unknown. Even if it can be shown that these statements are sufficient to fix the values of the unknowns, it is quite possible, and indeed likely, that the mathematical statement of the problem will afford no clue as to how a process for evaluating the unknowns may be established. The filling of this gap is the object of the branch of mathematics known as numerical analysis. Therefore the best that one could expect a general purpose formula-recognition routine to do, would be to accept a statement of the problem after it had been examined, and if necessary transformed, by a numerical analyst.

Consider, for example, the problem of finding the root of the equation $x^3 - 3x + 1 = 0$ which lies between 0 and 1. This is a case in which the statement of the problem gives no guide as to how a solution might be obtained. It would, of course, be possible to solve the problem on a digital computer if a program specially designed for solving polynomial equations were available. As it stands, however, the equation is not of suitable form to be presented to a general purpose formula-recognition routine.

For the purpose of calculating the root in question, we can replace the original equation by

$$x_2 = x_1 - (x_1^3 - 3x_1 + 1)/3(x_1^2 - 1).$$

This is understood to imply an iterative procedure in which a value is assumed for x_1 (say, $x_1 = \frac{1}{2}$), and the value of x_2 is calculated from the formula. The next step is to take the value obtained for x_2 as a new value for x_1, and to evaluate a new value for x_2. If $|x_2 - x_1|$ is less than an assigned quantity, the process is terminated; otherwise it is repeated.

Any of the systems of formula recognition which have been developed would enable a program for calculating x_2, given x_1 to be formed automatically; the quantities x_1 and x_2 would be stored in locations assigned by the routine. It will be seen, however, that there is a good deal more to the complete program than a sequence of orders for calculating x_2 in terms of x_1 and, in fact, the construction of the iterative loop with means for emerging from it is something which must be left to the programmer; all that the formula-recognition routine can do is to enable him to give the necessary instructions to the machine in a semi-algebraic language.

While the authors do not attach the same importance to formula-recognition systems as to the other systems which have been discussed in this chapter, they believe that they may often be of use in enabling people who do not wish to spend much time studying the subject to write programs for simple problems. Their utility in this respect, in any particular computing center, depends partly on how straightforward and easily grasped the standard methods of programming used in that center are. An important consideration is whether the basic order code of the machine includes orders for floating-point arithmetic; if it does not, then a formula-recognition routine which makes good the deficiency may be, for this reason alone, of overwhelming attraction to programmers. However, as already pointed out, in such cases the use of interpretive methods will usually be necessary, and there will, in consequence, be a serious price to be paid by way of reduction in effective machine speed. Even in more favorable cases, experienced programmers will be able to obtain greater efficiency by using more conventional methods of programming.

PART TWO

SPECIFICATIONS OF EDSAC LIBRARY SUBROUTINES

Each subroutine is distinguished by a letter denoting its category, and a serial number within that category. The categories are as follows.

Category	*Subject*
A	Floating-point arithmetic
B	Arithmetical operations on complex numbers
C	Error diagnosis
D	Division
E	Exponentials
F	General subroutines relating to functions
G	Differential equations
L	Logarithms
M	Miscellaneous
N	Double-length arithmetic
P	Print and layout
Q	Quadrature
R	Read (i.e., Input)
S	*n*th root
T	Trigonometric functions
X	Complete programs
Z	Post-mortem routines

In the specifications on succeeding pages the following information is given in abbreviated form immediately beneath the title of each subroutine:

1. Type of subroutine, i.e., whether open, closed *A*, closed *B*, interpretive, or special.

2. Restriction on address of first order. If the word "even" appears it denotes that the first order must be placed in an even location. If no note appears it indicates that the location may be either odd or even.

3. Total number of storage locations occupied by the subroutine.

4. Addresses of any storage locations needed as temporary storage by the subroutine (other than 0*D*, which is used by the majority of subroutines).

5. Approximate operating time (not possible to state in all cases).

A. Subroutines to carry out floating-point arithmetic.

*A*9 *Input of a sequence of numbers in floating decimal form during input of orders* (*used with A*11).

Special; even; 31 storage locations.

The numbers are punched on a separate data tape in the following form: character representing exponent; sign; numerical part (the decimal point being after the first digit). For example,

$$\begin{array}{llll} 512 & \text{would be punched as } W + 512 \text{ or } 2 + 512 & = 10^2(5.12), \\ -.0012 & \text{“ \quad “ \quad “ \quad “ } B - 12 & = 10^{-3}(1.2). \end{array}$$

The first number in the sequence is preceded by $Z\ T\ X$. After the subroutine, $T\ m\ D$ is punched, followed by the data tape which is copied *in the reverse direction.* The numbers are then placed in the store in floating decimal form in storage locations mD, $(m - 2)D$, etc., so that mD is the location of the number originally punched *last.*

Accuracy: The numerical part of each number is represented by 23 binary digits—equivalent to almost 7 decimal digits.

Note: *R*9 must be in the store when *A*9 is read.

*A*10 *Print single floating decimal number* (*used with A*11).

Closed *A*; even; 63 storage locations; uses 4*D*.

Prints the signed exponent, followed by the signed numerical part, of the number stored in floating decimal form in 0*D*. Each number is printed as: negative sign, or space; exponent (2 figures); 2 spaces; negative sign, or space; integral part (1 figure); space; 6 decimal figures.

Accuracy: The number is rounded off to 7 figures (including integral part).

Notes: 1. The last order on the tape is the digit layout parameter for the numerical part (as in *P*30), and may be altered if required.

2. Normally, before the number is printed, carriage return and line feed will occur. They may be omitted by entering the subroutine at its third order. One space only is printed after each number. Not more than four numbers may be printed on one line.

*A*11 *Arithmetical operations on real numbers expressed in floating decimal form.*

Interpretive; even; 128 storage locations; uses 0*H* and 0*N* for floating accumulator; time, see Note.

Operations: $A11$ carries out the operations specified individually by interpretive orders according to the following code:

Order	*Operation*
$A\ m\ F/D^*$	Add to the number in the floating decimal accumulator the number represented by $C(m)$.
$B\ m\ F/D^*$	Subtract from the number in the floating decimal accumulator the number represented by $C(m)$.
$V\ m\ F/D^*$	Multiply the number in the floating decimal accumulator by the number represented by $C(m)$.
$T\ m\ F/D$	Transfer the number in the floating decimal accumulator to m, and clear the accumulator.
$E\ m\ F$	Transfer control to m with accumulator clear.

$^*m \leq 511$

Representation of numbers: Each number is expressed in the form $a \cdot 10^p$, where a is the numerical part and p the exponent (an integer). In the store the number is represented by the long or short number $a \cdot 2^{-11} + p \cdot 2^{-6}$. The routine uses positions $0H$ (long) and $0N$ (short) as a "floating decimal accumulator," or "f.d.a.," in which the above numbers would appear as $-a \cdot 2^{-11}$ in $0H$ and $p \cdot 2^{-14}$ in $0N$.

Range of values: In the f.d.a. $|p| < 16000$ approx., and $-2048 \leq a < 2048$. In the store $-63 \leq p < 63$ and $|a| \leq 10$, but when a number is transferred to the store from the f.d.a. it is always represented in such a way that either $1 < |a| \leq 10$, or $a = 0$ and $p = -63$.

Capacity of registers: If an interpretive T-order is encountered, and the number in the f.d.a. exceeds 10^{63}, the machine will normally come to a dynamic stop. If this is undesirable, the preset L-parameter $E\ 56\ \theta$ may be replaced by any E-order transferring control to a suitable point in the store in the event of capacity being exceeded (the accumulator not being empty).

If the number in the f.d.a. is less than or equal to 10^{-63}, an interpretive T-order will place the representation of zero in the store (0.10^{-63}).

It is possible to exceed the capacity of the *numerical* part of the f.d.a. without the number actually represented by the f.d.a. exceeding the range of possible values. The rules for avoiding this are as follows:

After an interpretive T-order, $|a| = 0$; an interpretive A- or B-order may increase $|a|$ by ten, and an interpretive V-order may multiply $|a|$ by ten, in the worst cases; hence the sequence of interpretive orders should be such as to ensure that a can never reach 2048.

Accuracy: When using long numbers, a has between 23 and 27 binary digits, that is, 7 or 8 decimal digits. When using short numbers, a has between 5 and 9 binary digits, that is, about 2 decimal digits.

Preset parameters:

45	*H*	*P* *s* *D*	Location of numerical part of f.d.a.
46	*N*	*P* *t* *F*	Location of exponent of f.d.a.
47	*M*	*P* 103 θ	Set and used internally by *A*11
48	Δ	*P* θ	
49	*L*	*E* 56 θ	

Notes: 1. Times of operation are

Order	*Time in seconds*
A	0.066
B	0.066
T	0.05 + m(0.015), where m is the number of decimal shifts necessary to convert the number to the form required in the store.

2. See Part 3 for detailed program.

*A*30 *Operations on real numbers expressed in floating decimal form.*

Interpretive; even; 298 storage locations.

This subroutine provides much more comprehensive facilities than *A*11. A short description will be found in Part 1, Section 5–15. The full details are beyond the scope of this book.

B. Subroutines to perform arithmetical operations on complex numbers.

*B*2 *Complex operations other than division.*

Interpretive; even; 53 storage locations; uses 0*H* to 10*H*; time approx. 0.075 sec per operation.

When called in, *B*2 carries out the interpretive orders occurring in the program immediately after the orders calling in *B*2. These interpretive orders are in a special code; they specify operations on complex numbers. The address n in an interpretive order specifies the complex number $C(nD) + i \cdot C(n + 2)D$. The operations which may be carried out correspond to *A*, *S*, *T*, *U*, *V*, *N*, *Y* (round-off), and also right or left shift of one or two places.

Preset parameters:

45	H	$P\ h\ D$	(h must be even)
46	N	Parameter is used internally by $B2$.	

Notes: 1. There is no operation corresponding to an H-order. The role of "complex multiplier register" is undertaken by storage locations $0H$ and $2H$, which may be filled by the interpretive orders $T\ \ H$ or $U\ \ H$.

2. Shifts of up to six places must be effected by a series of single or double shifts. A shift of n places, where $7 \leq n \leq 14$, may be obtained by a pair of interpretive orders such as

$$L\ 2^{n-5}\ F, \qquad L\ 2^{n-5}\ -2\ F.$$

3. Interpretive orders $T\ \ D$ and $U\ \ D$ must not be used, since they would destroy the content of $2D$.

4. Exit from $B2$ is made by an interpretive E-order immediately following an interpretive T-order; control is transferred to the address specified in the E-order.

5. Care should be taken to ensure that storage locations $4H$ to $10H$, which are used as accumulators for the real and imaginary parts of the complex numbers, are cleared before $B2$ is called in for the first time.

6. See Part 3 for detailed program.

$B4$ *Division of complex numbers.*

Closed B; 62 storage locations; uses $4D$, $6D$, and B-register; time approx. $(0.016m + 0.120)$ sec, where modulus of divisor lies between 2^{-m} and 2^{-m-1}.

Forms $\dfrac{C(4H) + iC(6H)}{C(H) + iC(2H)}$ and places the real and imaginary parts of the result in $4H$ and $6H$.

Accuracy: The maximum error in the real or imaginary part of the quotient is 2^{-33}.

Preset parameter: 45 | H | $P\ h\ D$ (h must be even)

Notes: 1. If the modulus of either the real or imaginary part of the quotient is >1, the subroutine will stop on a ϕ-order.

2. The accumulator should be clear on entering, and will be clear on leaving the subroutine.

3. Uses an iterative process similar to that used in $D6$.

C. Error-diagnosis subroutines.

C27 Check-point subroutine.

Special; 30 + 4*r* storage locations.

This is a simple check-point subroutine which causes extra indicating symbols to be punched whenever control reaches certain specified check points in a program. The *r* check points are specified by the addresses in a series of *r* pseudo-orders (each terminated by *S*) which are punched immediately after *C*27 on the input tape. When the program is carried out, each time control reaches a check point the function letter of the corresponding pseudo-order will be punched out.

Notes: 1. *C*27 plants an *F*-order in each check point. These *F*-orders must not be altered in any way, either during input or during the operation of the program.

2. *C*27 does not automatically punch carriage return or line feed symbols. If these are required the pseudo-orders

$$\begin{array}{ccc}\theta & n & S\\ \Delta & n+1 & S\end{array}$$

should be included. Carriage return and line feed will then be punched whenever control reaches *n* and *n* + 1.

3. The first 28 orders of *C*27 may be overwritten after all the pseudo-orders have been read in.

4. To save storage space and printing time, *C*27 may, if desired, be written over the print subroutine of the program. In this case, when all the pseudo-orders have been read in, a dummy print routine may be written over the first few orders of *C*27. This can be of the following form:

```
      blank tape
        P       Z
        T       Z
 0      A     3 F
 1      T     3 θ
 2      O     4 θ
 3     (Z       F)  Link order
     ______________
 4    || K 4096 F
```

Each time the program directs control to its print subroutine the symbol for *P* will then be punched, instead of a number.

5. The program should be started, at the order in storage location *m*, by the usual control combination, *E m K P F*.

6. *C*27 modifies the initial input routine so that the letter *S* can be used for terminating the pseudo-orders specifying the check points. This is entirely different from the standard use of *S* by *R*30, in connection with the *B*-register. *C*27 must therefore be placed in the store *after* any part of the program which uses the normal *S* facility.

7. See Part 3 for detailed program.

*C*30 *Numerical check-point subroutine.*

Special; 57 storage locations; even.

Punches *C*(Acc), as a long signed decimal, at a specified check point in a program.

Preset parameter: 45 |*H*| *P h F* (*h* is location of check point)

Notes: 1. *C*30 plants an *F*-order at the check point. This *F*-order must not be altered in any way, either during input or during the operation of the program.

2. Symbols for carriage return and line feed are punched before *C*(Acc).

3. *C*(Acc) is punched before the machine obeys the order which was originally at the check point.

4. The less significant half of the accumulator is cleared at the check point.

5. Since the accumulator is cleared during the operation of *C*30, it is possible for a ϕ-order following the check point *not* to indicate an overflow, even if it should have done so in the original program without *C*30.

6. The program should be started, at the order in location *m*, by the usual control combination, *E m K P F*.

[Two further check-point subroutines, *C*28 and *C*29, exist in the EDSAC library. They provide elaborate facilities for the optional printing of varied information, in several different forms, at specified check points.]

*C*31 *Trace of function letters, with delayed start.*

Special; 66 storage locations.

May be applied to a program to punch out a trace of the progress of part of the program. The program works at full speed until the order in *h* is obeyed. *C*31 then takes control, obeys the rest of the program order by order (at much reduced speed), and punches out the function letter of each order as it is executed.

Preset parameter: 45 $|H|$ $P\ h\ F$ (trace starts at order in h)

Notes: 1. h must be so chosen that the accumulator is always empty before $C(h)$ is obeyed, and that $C(h)$ is not used or altered in any way by the program before checking begins. $C31$ must not be read into the machine before that part of the original program which includes $C(h)$.

2. Carriage return and line feed symbols are punched immediately after the function letter of any order which transfers control.

3. The symbol for a full stop is punched instead of ϕ.

4. On obeying Y, E, F, G, J-orders the least significant digit of the accumulator (2^{-69}) is converted into 0.

5. The program should be started, at the order in location m, by the usual control combination, $E\ m\ K\ P\ F$.

6. A letter shift symbol is punched during input. Any numerical information punched by the print subroutine of a program will, therefore, be printed as the corresponding letter-shift characters.

7. See Part 3 for detailed program.

D. Division subroutines.

D6 Division.

Closed A; 36 storage locations; uses $6D$ and $8D$; time $=$ $(0.01m + 0.12)$ sec, where $2^{-m-1} \leq |C(4D)| < 2^{-m}$.

Forms $C(0D)/C(4D)$, where $C(4D) \neq 0$ and $\neq -1$, and places result in $0D$.

Accuracy: Maximum error is $\pm 2^{-35}$, (quotient) $\pm 2^{-34}$.

Notes: 1. Uses the iterative process

$$a_{n+1} = a_n - c_{n+1}a_n + c_{n+1},$$
$$c_{n+1} = -a_n b + (b - 1),$$

where b is the shifted divisor, $1-a_n \rightarrow 1/b$, $c_n \rightarrow 0$; a_n and c_n are negative, $a_0 = 2b - 2\sqrt{2} + 1$; c_n is therefore negative until the process is completed.

2. $C(4D)$ is disturbed by the operation of $D6$.

3. If $C(4D) = 0$ or -1 the program will go into a loop.

D7 Division

Closed A; 26 storage locations; time $=$ $(0.012m + 0.105)$ sec, where $2^{-m-1} \leq |C(4D)| < 2^{-m}$.

Forms $C(0D)/C(4D)$, where $C(4D) \neq 0$ and $\neq -1$, and places result in $0D$.

Accuracy: Maximum error is $3(1 + |k|) \cdot 2^{-34}$, where k = quotient.

Notes: 1. Uses the repetitive process

$$a_{n+1} = -a_n c_n + a_n \quad (a_0 = \text{dividend}, \; c_0 + 1 = \text{divisor})$$

$$c_{n+1} = -{c_n}^2;$$

stop when $c_n = 0$.

2. The less significant half of the accumulator is not cleared at the end of the operation, that is, $0 \leq C(\text{Acc}) < 2^{-34}$.

3. At the end of the process $-2^{-34} \geq C(4D) > -2^{-17}$.

4. If $C(4D) = 0$ or -1 the program will go into a loop.

[Slightly less accurate than *D*6.]

*D*11 *Division, with double-length dividend held in the accumulator.*

Closed *B*; 30 storage locations; time $(0.135 + 0.016n)$ sec, where n is the number of times that the divisor must be doubled before it exceeds $\frac{1}{2}$ in magnitude.

Replaces both $C(\text{Acc})$ and $C(0D)$ by $C(\text{Acc})/C(4D)$, where $C(\text{Acc})$ is a double-length number held in the accumulator.

Accuracy: Maximum error is $3(1 + |k|) \cdot 2^{-34}$, where k is the quotient.

Notes: 1. If $C(4D)$ is initially -2^{-p}, where p is an integer, the quotient will be the shifted, truncated numerator, and the time taken will be $(0.044 + 0.016p)$ sec.

2. If $C(4D)$ is initially zero the program will go into a loop.

3. $C(4D)$ is disturbed by the operation of *D*11.

4. The capacity of the accumulator may legitimately be exceeded during the operation of *D*11. A subsequent *Y m D* order may therefore transfer control unexpectedly.

5. See Part 3 for detailed program.

E. Exponential subroutines.

*E*4 *Exponential function.*

Closed *A*; 36 storage locations; even; time 0.13 sec.

Forms exp (x), where $x = C(R)$, and places result in $0D$. $-1 \leq x \leq 0$. *R*9 must be in the store when *E*4 is being read.

Accuracy: Maximum error is $2^{-34} + 2^{-35} \sum_{r=0}^{7} |x|^r$.

*E*6 *Exponential function, large range.*

Closed *B*; 38 storage locations; even; time $(0.09 + 0.015p)$ sec.

Forms exp $(2^p y)$, where $y\ (<0) = C(R)$ and $p \geq 1$. The result is left in the accumulator, which must be clear on entry to *E*6.

Preset parameter: 45 | *H* | *P p F* (*p* is binary exponent of argument)

Accuracy: Maximum error is less than $(2^{p-1} + 1)2^{-34}$; this occurs when the result is nearly equal to unity. The error diminishes rapidly for smaller values of the result, and for small values it is less than 3.2^{-35}.

Notes: 1. *R*9 must be in the store when *E*6 is read.
2. See Part 3 for detailed program.

F. General subroutines relating to functions.

*F*2 *Inverse interpolation, or solution of* $f(x) = 0$ *(second-order process).*

Closed *A*; 58 storage locations; uses $4D$ and $4H$ to $8H$.

Places in $0H$ a solution of $f(x) = 0$, where $f(x)$ is defined by an auxiliary subroutine. Two trial values, x_1 and x_2, must be placed in $0H$ and $2H$ before *F*2 is called in; they must be such that $f(x_1)$ and $f(x_2)$ have opposite signs. The solution will lie between x_1 and x_2.

Preset parameters:	45	*H*	*P h D*	Defines locations of trial values and of working space
	46	*N*	*P n F*	First order of auxiliary subroutine is in *n*.
	47	*M*	Parameter is used internally by *F*2	

Notes: 1. The auxiliary subroutine must be of closed *A* type, and should place $f[C(0H)]$ in $0D$, leaving $C(0H)$ unaltered. It may use $4D$, but not $2H$ through $8H$.

2. If $f(x_1)$ and $f(x_2)$ have the same sign, $F2$ will place x_2 in $0H$ and leave 2^{-35} in the accumulator.

3. If x is not required to an accuracy better than 2^{m-33}, where $m \leq 10$, order 52 of $F2$ may be replaced by $R\ 2^m\ F$. This will save time, but the less significant half of the accumulator may not be empty on exit from $F2$.

4. See Part 3 for detailed program.

F7 Interpolation in a table, using Neville's process.

Closed B; $46 + 2n$ storage locations; even; uses $4D$ and $0H$, $2H, \ldots, (2n - 2)H$; time approx. $(0.01n^2 + .003n + .005)$ sec.

Given, in consecutive long-storage locations, a table of values of a function at interval 2^{-m} of the argument, with the entry corresponding to zero argument specified by a program parameter, $F7$ calculates the value of $f[C(\text{Acc})]$ and places it in the accumulator. The number of entries over which interpolation is made is specified by a preset parameter.

Preset parameters:	45	H	P	h	D	hD defines beginning of working space
	46	N	P	$2n$	F	$F7$ uses n function values
	47	M	P	2^{15-m}	F	The table is at interval 2^{-m}

Program parameter:	p	B	p	F	Orders calling in $F7$
	$p + 1$	F	s	F	
	$p + 2$	$\|\|P$	$2a$	F	$2aD$ is location corresponding to zero argument

Accuracy: Maximum rounding off error is $\pm 2^{-36}n(n + 1)$; rms value is $2^{-35} \cdot n$, where interpolation is made over n entries. Truncation error, $R_n = (x - c)(x - c - 2^{-m}) \ldots (x - c - (n - 1)2^{-m})f^n(k)/n!$, where c is the argument corresponding to the first tabular value used, and $c \leq k \leq c + (n - 1) \cdot 2^{-m}$.

Notes: 1. If the capacity of the accumulator is exceeded during $F7$ the machine will stop on a ϕ-order.

2. The quantity ($=q$, say) in 47θ causes the first function value used to correspond to argument y, where $(x - q) \geq y > (x - q - 2^{-m})$. q is initially set (by an interlude) to $(n - 2) \cdot 2^{-m-1}$, which causes the subroutine to use function values as nearly as possible symmetrical about x.

3. The subroutine fails unless $3 \leq m \leq 11$.

4. Care must be taken to avoid values of the argument that would cause the subroutine to use function values outside the table.

5. The argument for which the value of the function is found can also be thought of as $(a - b)2^{-m} + x$, where $P\ 2a\ F$ is the program parameter and the location corresponding to zero argument is $2bD$.

6. See Part 3 for detailed program.

G. Subroutines for the integration of ordinary differential equations.

G4 Integration of $y'' = f(x,y)$ by sixth order process.

Closed A; 47 storage locations; uses $4D$, $10D$, and $0H$ to $22H$. Time = $(0.175 + 3t)$ sec, where t = time of auxiliary subroutine.

Each time $G4$ is called in it advances the integration by one step. At the beginning of the integration the following initial values must be planted:

$0H$	$y_1 + \delta^2 y_{\frac{1}{2}}''(h^2/240)$	$10H$	x_1
$2H$	$\delta y_{\frac{1}{2}}$	$12H$	h
$4H$	y_1''	$14H$	h^2
$6H$	$\delta y_{\frac{1}{2}}''$	$16H$	1/12 (=0.083)
$8H$	$\delta^2 y_0''$	$18H$	$h^2/240$

An auxiliary subroutine, of closed A type and starting in $0N$, must be provided to calculate $f[C(10H), C(20H)]$ and place it in $22H$. On exit from $G4$, corresponding values of x and y will be found in $10H$ and $20H$, respectively.

Preset parameters:	45	H	$P\ n\ D$	nD defines beginning of working space
	46	N	$P\ m\ F$	Auxiliary subroutine begins in m

Accuracy: Truncation error may be taken approximately as

$$(y_0^{(v)} - y_n^{(v)})h^5/240,$$

over range y_0 to y_n.

Notes: 1. $\delta^2 y_{\frac{1}{2}}''(h^2/240)$ is a small quantity, so an approximate value will suffice.

2. The auxiliary subroutine may use $0D$ and $4D$, but not $10D$.

$G12$ *Integration of one or more simultaneous differential equations by Runge-Kutta-Gill process.*

Closed B; 49 storage locations; even; uses $4D$; time $= (0.09 + 0.14n + 4t)$ sec, where n is number of variables and t is time of auxiliary subroutine.

The variables y are stored in n consecutive long-storage locations, the first of which is aD. Each time $G12$ is called in it will advance the values of these variables by one step. An auxiliary subroutine must be provided to calculate all the derivatives from given values of the variables. It should be of closed B type, and have its first order in d. The quantities $2^m hy'$, calculated by the auxiliary subroutine, should be placed in n consecutive long-storage locations, the first of which is bD. A further set of n consecutive storage locations, the first of which is cD, must be provided to hold the quantities $2^m q$; at the beginning of a range these must be cleared.

For a detailed description of the process see Part 1, Section 5–12.

Preset parameters:

45	H	P	a	D	aD is location of first variable, y
46	N	P	b	D	bD is location of first $2^m hy'$
47	M	P	c	D	cD is location of first $2^m q$
48	Δ	P	$2n$	F	n is number of variables
49	L	P	2^{m-2}	F	(or P D if $m = 1$)
50	X	P	d	F	d is location of first order of auxiliary subroutine

Accuracy: The truncation error in one step is of the order h^5. For a small set of well-behaved equations its magnitude is roughly $10^{-2}h^5$. Rounding-off errors accumulate at a rate corresponding to the keeping of $34 + m$ binary digits.

Notes: 1. m should be chosen so that the largest $2^m hy'$ is just within the capacity of the accumulator, with the proviso that $1 \leq m \leq 11$.

2. If a variable exceeds the capacity of the accumulator, the machine will stop on a ϕ-order.

3. The auxiliary subroutine may use $0D$ and $4D$.

4. The accumulator must be clear when $G12$ is entered, and also when control is returned to $G12$ from the auxiliary subroutine; it will be clear on returning to the master routine.

5. If the independent variable is required it may be treated as a dependent variable with the corresponding $2^m hy' = 2^m h$. The latter quantity may be set once and for all at the beginning of the range; it will not be disturbed.

6. See Part 3 for detailed program.

*G*13 *Integration of one or more simultaneous differential equations by Runge-Kutta process.*

Closed *B*; 47 storage locations; even; time = $(0.07 + 0.08n + 4t)$ sec, where n is number of variables and t is time of auxiliary subroutine.

Similar to *G*12 but uses the faster, orthodox Runge-Kutta process, which needs a further n consecutive long-storage locations, the first of which is *fD*, to hold the variables within a step. At the beginning of a range the initial values of the variables must be set both in these locations and in the set starting in *aD*.

Preset parameters:

45	*H*		Same as for *G*12
·			
·			
·			
50	*X*		
51	*G*	*P f D*	*fD* is location of first variable, *y*

Accuracy: See *G*12.

Notes: 1. m should be chosen so that the largest $2^m hy'$ is just within the capacity of the accumulator, with the proviso that $1 \leq m \leq 10$.

2. See also Notes 2, 4, and 5 of *G*12.

*G*14 *Location of zero of variable, to specified accuracy, using G*12 *or G*13.

Closed *B*; 33 storage locations; even; uses 4*D*; time—see Note 3.

*G*14 uses *G*12 or *G*13 to advance the integration of a set of ordinary differential equations to a point at which the modulus of a specified variable, y_r, is less than or equal to a specified quantity of the form $k \cdot 10^{-s-1}$. *G*14 also uses *D*11.

Preset parameters:

45	*H*	*P*	*h*	*D*	Location of variable y_r which is to vanish
46	*N*	*P*	*p*	*D*	Location of $2^m hy'_r$
47	*M*	*P*	*q*	*D*	Location of $j \cdot 2^m h$
48	Δ	*P*	*g*	*F*	Location of first order of *G*12 or *G*13
49	*L*	*P*	2^{m-2}	*F*	m as in *G*12 or *G*13
50	*X*	*P*	*e*	*F*	Location of first order of *D*11
51	*G*	Σ	*s*	*F*	Accuracy required (see below)

Accuracy: The accuracy required is specified by the pseudo-order $\Sigma\ s\ F$ (where Σ stands for any function letter) in the G-parameter position. $G14$ transfers control back to the master routine when $y_r \leq k \cdot 10^{-s-1}$, where k is the decimal equivalent of the function letter Σ and $0 \leq k, s \leq 9$.

Notes: 1. The auxiliary subroutine provided for $G12$ or $G13$ must be so designed that the quantity h is determined by the content of storage location q. This number will be altered by $G14$ and must therefore be reset before the integration is continued. $G14$ never increases the modulus of $C(q)$, however, so that if, on entering $G14$, this is sufficiently small for accurate integration, it will remain so.

2. When $G14$ is entered, an approximately correct value of $2^m h y_r'$ must already be in place in pD. This will be the case if the integration has already proceeded for at least one step; if not, the auxiliary subroutine must be called in once, independently of $G12/G13$, before entering $G14$.

3. Until the specified variable comes within one step of the zero, $G14$ will add 0.05 sec to the time taken by $G12$ or $G13$ for each step of the integration. Thereafter it will also add the time taken by $D11$. The number of steps taken in this last part of the integration will be about 4 or 5 if $k = 0$, and less otherwise.

4. Integration proceeds in the direction of decreasing modulus of y_r. If a minimum of this modulus is encountered which has a value greater than $k \cdot 10^{-s-1}$, $G14$ will cycle indefinitely.

5. The approximate zero found may be on either side of the true zero.

6. The accumulator should be clear on entering $G14$ and will be clear on leaving it.

L. Subroutines for evaluating logarithms.

L4 Logarithm.

Closed B; 54 storage locations; even; time $(0.12 + 0.003n)$ sec, where $2^{-n-1} < C(\text{Acc}) \leq 2^{-n}$.

Replaces $C(\text{Acc})$ by $2^{-5} \log_e C(\text{Acc})$.

Accuracy: $\pm 2^{-33}$.

Notes: 1. $R9$ must be in the store when $L4$ is read.

2. See Part 3 for detailed program.

M. Miscellaneous subroutines.

*M*18 *Store repetitive pattern of orders.*

Special; 19 orders

This subroutine is intended for use on those occasions when it is desired, for the sake of greater speed, to use a single, long sequence of orders for a particular calculation, instead of a short sequence repeated, with modification, a number of times. *M*18 allows input time to be saved by rendering it unnecessary for the entire sequence to be punched on the tape. Orders for the first two cycles are punched, and go into the store immediately before *M*18. *M*18 is called into action as soon as it is read, and plants the remaining orders of the sequence, forming their addresses by linear extrapolation.

Parameters: The following must be punched immediately after *M*18:

$O\ c\ F$	$c =$ number of orders in set
$T\ t\ F$	Last order of last set to be copied into location t
$E\ \ Z$	
$T\ f\ F$	First order of first set to be copied into location f.

Notes: 1. Since *M*18 proceeds by linear extrapolation, different orders may be increased by different steps. In particular, if desired, an order may appear alternately as A and S, with regularly increasing address.

2. The copying process may be stopped in the middle of a set, if required, by punching the second parameter appropriately.

3. At the end of the process *M*18 transfers control back to the initial input routine. The Transfer Order must then be restored by a suitable control combination punched on the input tape.

4. *M*18 may be used more than once during the input of a program.

*M*20 *Set parameter value, by means of telephone dial, during input of orders.*

Special; uses no storage space.

If *M*20 is included at the appropriate point on the input tape, the H-parameter may be set to $d \cdot 2^{-15}$ by dialing an integer d. As soon as the first few rows of *M*20 have been read the machine stops on a Z-order. Exactly three decimal digits should then be dialed to specify d.

Notes: 1. A preset parameter other than H may be set by changing the control combination $T\ 45\ K$ near the end of the *M*20 tape.

2. If it is desired to dial more, or less, than three digits the central section of *M*20 (marked on the program sheet) should be repeated an appropriate number of times, or omitted, as the case may be.

3. See Part 3 for detailed program.

M30 Sideways addition (Gillies-Miller method).

Open; 20 + 15 storage locations; time, see Note 3.

Counts the 1's in $C(R)$ and places the sum n in $0F$ as $P\ n\ F$.

Preset parameter: 45	H	$P\ h\ F$	$0H$ is the location of the first of a set of 7 collating constants

Notes: 1. The accumulator must be clear at the start of $M30$, and will be clear at the end.

2. Times of operation are

$$0.035 \text{ sec if } C(R) < 0, \qquad 0.033 \text{ sec if } C(R) \geq 0.$$

3. See Part 3 for detailed program.

[If $C(R)$ is known to be always positive, a faster subroutine may be written by omitting the third and fourth orders.]

M31 Serial correlation

Closed B; $30 + 2m$* storage locations; 1st location even; time $(0.006ml + 0.014m + 0.018l + 0.01)$ sec.

Adds into 2Δ, 4Δ, . . . , $2l\Delta$, in the form of long integers, the sums

$$4 \sum_{i=1}^{m} x_i y_{i+j} \quad (j = 0, 1, \ldots, (l-1)),$$

where the sequence of short integers x_i is stored in locations $0H$, $1H$, . . . , and the sequence y_i in locations $0N$, $1N$, ...

$$-4 \sum_{j=0}^{l-1} \sum_{i=1}^{m} x_i y_{i+j}$$

is computed independently and added into 0Δ. This quantity is intended to be used as a check sum.

Preset parameters:

45	H	$P\ h\ F$	1st set of numbers in $0H$, $1H$, . . .
46	N	$P\ n\ F$	2nd set of numbers in $0N$, $1N$, . . .
47	M	$P\ m\ F$	m is number of products
48	Δ	$P\ d\ D$	Check sum to 0Δ, products to 2Δ, 4Δ, . . .
49	L	$P\ l\ F$	l is number of correlations used by
50	X	$P\ x\ F$	subroutine.

Notes: *1. In order to save computing time, this subroutine uses a sequence of m pairs of H- and V-orders, instead of a repetitive loop, to compute the sum of products. These orders are planted, during input, in storage locations 24θ, 25θ, . . . , $(24 + 2m - 1)\theta$, by means of an interlude.

2. See Part 3 for detailed program.

N. Operations on double-length numbers.

N2 Arithmetical operations on real numbers, each occupying two long-storage locations.

Interpretive; 43 storage locations; time, see Note 1.

$N2$ carries out the operations specified individually by interpretive orders according to the code given below. The operations are carried out on real numbers each occupying two consecutive long-storage locations. The value of the number stored in $C(mD)$ and $C(m+2)D$ is

$$2^n[C(mD) + 2^{-33} \cdot C(m+2)D],$$

where n can have any integral value such that $0 \leq n \leq 13$,

$$-1 < C(mD) < 1, \qquad \text{and} \qquad 0 \leq C(m+2)D \leq \tfrac{1}{2}.$$

The subroutine uses the adjacent long-storage locations $0H$ and $2H$ as a double-length accumulator, or d.l.a., in which numbers are stored in the same way.

Interpretive code:

$A\ m\ D$ Add to C(d.l.a.) the number represented by $C(mD)$ and $C(m+2)D$

$S\ m\ D$ Subtract from C(d.l.a.) the number represented by $C(mD)$ and $C(m+2)D$

$G\ m\ D$ Multiply C(d.l.a.) by the number represented by $C(mD)$ and $C(m+2)D$

$\theta\ m\ D$ Multiply C(d.l.a.) by -1 and by the number represented by $C(mD)$ and $C(m+2)D$

$T\ m\ D$ Transfer C(d.l.a.) to mD and $(m+2)D$ and clear the d.l.a.

$U\ m\ D$ Transfer C(d.l.a.) to mD and $(m+2)D$ and do not clear the d.l.a.

$E\ m\ F$ Transfer control to the order in $(m+2)F$, with the content of the less significant half of the d.l.a. in the accumulator.

Preset parameters:

45	H	$P \quad h \quad D$	hD and $(h + 2)D$ form the d.l.a.
		$\{H \qquad F$	If $n = 0$
46	N	$L \ 2^{n-2} \ F$	If $0 < n \leq 12$
		$L \qquad F$	If $n = 13$

Accuracy: The operations are carried out to 67 binary places (about 20 decimal places).

Notes: 1. Times of operation are as follows:

Order	*Time (sec)*
A	0.040
S	0.036
G	0.056
θ	0.056
T	0.036
U	0.036
E	0.012

2. C(d.l.a.) may be doubled or squared by adding it to itself or multiplying it by itself.

3. The interpretive orders can be modified, by the addition of the content of the B-register, in the same way as ordinary EDSAC orders.

*N*3 *Input of one double-length signed decimal fraction (used with N*2*).* Closed A; 35 storage locations; even.

*N*3 reads one fraction of up to 20 decimal places and places it in 0Δ and 2Δ, using the same representation for double-length numbers as *N*2, n being zero. Any number of digits may be punched, but a code letter V must be punched after the 10th decimal place, and a G or a θ after the 20th—G for a positive number and θ for a negative number.

Preset parameters:

45	H	$P \ h \ D$	hD and $(h + 2)D$ form d.l.a.	used by *N*2
46	N	$H \quad F$	$n = 0$	
47	M	$P \ m \ F$	m is location of first order of *N*2	
48	Δ	$P \ d \ D$	dD and $(d + 2)D$ are locations into which number read is placed	

Notes: 1. The number is read starting from its more significant end.

2. $2H$ must be clear when $N3$ is entered. This will be so if $2H$ was last used by $N2$ and control was transferred from $N2$ by an E-order following a T-order.

3. The B-register must be clear during the operation of $N3$.

P. Print subroutines.

Subroutines in this category should strictly be termed *punch* subroutines, since the output medium of the EDSAC is punched paper tape. For convenience, however, they will be described in terms of the final printed records which are obtained when the output tapes are run through a tape reader connected to a teleprinter.

*P*30–47 *Standard set of print subroutines.*

Closed A; use $4F$.

Number	Description	Locations used	Preset parameters
*P*30	Print $C(R)$ as signed fraction, with column layout	48 (even)	*H*
*P*31	Print $C(R)$ as signed fraction, with page layout	61 (even)	*H,N*
*P*32	Print $C(R)$ as positive fraction, with column layout	43 (even)	*H*
*P*33	Print $C(R)$ as positive fraction, with page layout.	56 (even)	*H,N*
*P*34	Print $2^{34}C(R)$ as long signed integer, with column layout	50 (even)	—
*P*35	Print $2^{34}C(R)$ as long positive integer, with column layout	43 (even)	—
*P*36	Print $2^{16}C(R)$ as short signed integer, with column layout	40	—
*P*37	Print $2^{16}C(R)$ as short positive integer, with column layout	34	—
*P*38	Print $C(R)$ as signed fraction, with no layout	44 (even)	*H*
*P*39	Print $C(R)$ as positive fraction, with no layout	39 (even)	*H*
*P*40	Print $2^{34}C(R)$ as long signed integer, with no layout	46 (even)	—
*P*41	Print $2^{34}C(R)$ as long positive integer, with no layout	39 (even)	—

(*continued*)

Number	Description	Locations used	Preset parameters
*P*42	Print $2^{16}C(R)$ as short signed integer, with no layout	36	—
*P*43	Print $2^{16}C(R)$ as short positive integer, with no layout	30	—
*P*44	Print $2^{34}C(R)$ as long signed integer, with page layout	63 (even)	*H*
*P*45	Print $2^{34}C(R)$ as long positive integer, with page layout	56 (even)	*H*
*P*46	Print $2^{16}C(R)$ as short signed integer, with page layout	53	*H*
*P*47	Print $2^{16}C(R)$ as short positive integer, with page layout	48	*H*

Preset parameters: 45 *H*] See Notes 2(iii) and 3(i).
46 *N*]

Accuracy: Fractions are automatically rounded off before being punched, the round-off number being calculated from the digit layout constant (see Note 3) by an interlude. If the digit layout constant is such that more than ten digits are punched, the round-off number is zero.

Notes: 1. The figure shift symbol is punched by an interlude while the subroutine is being read.

2. *Layout.* (i) Subroutines giving *no* layout print the number on the same line as, and immediately following, the last character to be printed. Layout symbols (for spaces, carriage return, and line feed) must be provided separately by the program, although the subroutines automatically print one space at the end of each number.

(ii) Subroutines giving *column* layout print the numbers in a single column at the left-hand margin of the paper. That is, each time the subroutine is called in it punches out a single number preceded by a carriage return and line feed. Note, however, that, if desired, the carriage return and line feed may be omitted by entering the subroutine at its third order. Thus a subroutine giving column layout can be used to give special types of page layout when desired.

(iii) Subroutines giving *page* layout print the numbers in n columns, with two spaces between columns, and in blocks of five lines, with one space between blocks. The value of n is determined by the *N-parameter, P* 5n *F,* in subroutines *P*31 and *P*33, and by the *H*-

parameter in subroutines *P*44 through *P*47. The two spaces between columns may be supplemented, if necessary, by providing suitable output orders in the program after leaving the subroutine. The number of lines in the block may be altered, if desired, by changing the last pseudo-order on the subroutine tape. This is normally *P* 5 *F* and should be changed to *P b F*, where *b* is the desired number of lines. The *N*-parameter must then be *P n · b F*. If it is desired to begin a new block at a specific point in a program, the last location occupied by the subroutine should be cleared. (See Example 5 of Chapter 7.)

3. *Digit layout.* (i) In the case of subroutines which print *fractions*, the digit layout (that is, the layout of the digits of a single number and of the spaces separating them) is determined by the *H-parameter P x F*, where *x* may be obtained in the following way. Imagine the printed characters, including both digits and spaces (only single spaces being permissible) to be laid out in the squares below, starting with the most significant digit in the left-hand square. *x* is then the result of adding the numbers *below* each space and the number *above* the right-hand digit.

8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1	
□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
24576	12288	6144	3072	1536	768	384	192	96	48	24	12	6		

For example: (a) To print ten digit members with a space after the fifth digit, $x = 1536 + 16 = 1552$; (b) to print ten digit numbers with spaces after the third, sixth, and ninth digits,

$$x = 6144 + 384 + 24 + 4 = 6556.$$

(ii) In the case of subroutines which print *integers*, the digit layout has been standardised so that a short integer is printed as a single group of five digits, while a long integer is printed as two groups of five digits separated by a single space, e.g., 12345 67890.

4. In the case of subroutines which print signed numbers, negative numbers are preceded by a minus sign and positive numbers by a space.

5. No provision is made for the suppression of nonsignificant zeros by subroutines which print integers.

6. See Part 3 for detailed programs of *P*31 and *P*40.

*P*50 *Print $C(R)$ as signed fraction, with column layout and variable digit layout.*

Closed *A*; 50 storage locations; even.

Prints $C(R)$ as a signed fraction rounded off to a preset number of decimal places specified by the *H*-parameter. Numbers are printed in column layout, but with a variable digit layout controlled by a program parameter.

Preset parameter:	45	*H*	$P\ h\ F$	Numbers are rounded off to *h* digits by adding $\frac{1}{2} \cdot 10^{-h}$

Program parameter:	m	$A\ m\ F$	Orders calling in *P*50
	$m + 1$	$F\ p\ F$	
	$m + 2$	$P\ x\ F$	x is digit layout parameter (see Note 3).

Notes: 1. The figure shift symbol is punched by an interlude while *P*50 is being read.

2. Details of the column layout will be found in Note 2(ii) of subroutines *P*30–47.

3. The digit layout parameter x is calculated from the rules given in Note 3(i) of *P*30–47.

*P*56 *Print C*(Acc) *as signed fraction, with digit and page layouts controlled by adjustable parameters within the subroutine.*

Closed *B*; 67 storage locations; uses 4*D*.

*P*56 will print C(Acc) correctly rounded off to d places of decimals. A positive number will be preceded by a space and a negative one by a minus sign. A space will be printed after the ith digit and after every successive sth digit other than the last. The numbers will be spaced across the page in x columns, and an extra line feed will occur after every y lines. Each number will be preceded by one, two, or three spaces according as *P*56 is entered at order 2, 1, or 0.

Accuracy: The round-off constant is correct to four significant figures.

Parameters: x, y, d, i, and s are initially set so that *P*56 prints five columns of ten figure numbers with five lines to a block. A space is printed after the fifth digit of each number. The parameters are, in fact, a set of *B*-orders, each of which is marked with an asterisk on the detailed program sheet. These orders are also marked on the tape, and may therefore be changed if the parameters need to be changed before input. During the program they may be changed by using *K*-orders to plant the desired *B*-orders within the subroutine. The locations and details of the *B*-orders concerned are as follows.

9θ	$B\ y\ F$	$y =$ number of lines to a block
12θ	$B\ x\ F$	$x =$ number of columns
18θ	$B\ d\ F$	$d =$ number of digits
27θ	$B\ i\ F$	$i =$ number of initial digits (before first space)
36θ	$B\ s\ F$	$s =$ number of digits between spaces

If $B\ \ F$ is planted in 3θ the next number to be printed will begin a new line and will set the column count to its initial value. Similarly, if $B\ \ F$ is planted in 6θ the next number to be printed will begin a new block and will set the line count to its initial value.

Note: See Part 3 for detailed program.

Q. Quadrature subroutines.

Q3 Quadrature, using Gauss' six-point formula.

Closed *A*; even; 48 storage locations; uses $(m + 4)D$; time = $(0.24 + 6t)$ sec, where t is time of auxiliary subroutine.

Places in $0D$ an approximation I to the integral

$$\int_{a-h}^{a+h} f(x)\,dx,$$

where $a = C(0H)$, $h = C(2H)$, and $f(x)$ is computed by an auxiliary, closed *A* subroutine, whose first order is in $0N$, and which places $f[C(0D)]$ in $0D$.

Preset parameters:

45	H	$P\ m\ D$	m defines locations of a and h
46	N	$P\ n\ F$	n is location of first order of auxiliary subroutine.

Accuracy: The truncation error of the formula used is approximately

$$10^{-15} \cdot f^{12}(\theta) \cdot (2h)^{13}.$$

The rounding-off error in the worst case is

$$2^{-35}[1 + 10h + 10h\,\{\max f'(x)\}]; \qquad (a - h) \leq \theta \leq a + h.$$

Notes: 1. $R9$ must be in the store when $Q3$ is read.

2. The mean value of the integral (that is, $I/2h$) is placed in $(m + 4)D$.

3. See Part 3 for detailed program.

$Q19$ *Gauss-type quadrature, with exponential weighting function (uses six-point formula).*

Closed A; even; 56 storage locations; uses $2H$; time $=$ $(0.2 + 6t)$ sec, where $t =$ time of auxiliary subroutine.

Places in $0D$ an approximation I to the integral $\int_a^\infty F(x) \cdot dx$, where $F(x) \sim f(x) \cdot e^{-bx}$, $f(x)$ having polynomial behavior, $1/b = C(0H)$, and the quantity $F[a + 2^4 C(0D)/b]$ is computed, and placed in $0D$, by an auxiliary closed A subroutine whose first order is in $0N$.

Preset parameters:			
45	H	$P\ h\ D$	hD is location of $1/b$
46	N	$P\ n\ F$	n is location of first order of auxiliary subroutine.

Accuracy: 1. The rounding-off error is not more than $2^{-3} \cdot (1 + 48/b)$.

2. The truncation error of the formula used is

$$0.053e^{-ab}/b^2[f^{(12)}(\theta)].$$

Notes: 1. $R9$ must be in the store when $Q19$ is read.

2. The modulus of the quantity $b/8 \cdot \int_a^\infty |F(x)| \cdot dx$ must be less than 1.

$Q20$ *Quadrature by Gregory's method.*

Closed B (see Note 1); even; 44 storage locations; uses $4D$, $6D$; time $=$ $(0.185 + 0.006n)$ sec.

Places in the accumulator an approximation I to the integral

$$\int_c^{c+(n-1)h} f(x)\, dx,$$

where $f(x)$ is specified by a table of long numbers, $f_r = f(c + rh)$, being stored in $2rH$; $h = C(0N)$. The formula used is

$$I = h\left[\sum_{r=0}^{n-1} f_r + \sum_{r=0}^{6} d_r(f_r + f_{n-r-1})\right].$$

Preset parameters:	45	*H*	*P a D*	*aD* is location of first entry in table
	46	*N*	*P b D*	*bD* is location of *h*

Accuracy: Rounding-off error is less than $(nh + 14)2^{-35}$. For details of truncation error, see Milne-Thomson, p. 191 (Bibliography p. 236). Truncation error is zero for polynomials of seventh degree or less.

Notes: 1. The accumulator must contain *B 2n F* when *Q*20 is called in.

2. n must be ≥ 7.

3. If the capacity of the accumulator is exceeded during the operation of *Q*20 the machine may stop on a ϕ-order. However, since the final answer is built up in the accumulator and left there without being transferred to the store, it is possible for capacity to be exceeded without an indication being given.

R. Input subroutines.

*R*2 *Input of positive integers during input of orders.*

Special; 15 storage locations (temporarily).

*R*2 is to be followed immediately on the input tape by the parameter *T m D* and a sequence of positive integers . Control is automatically transferred from the initial input routine to *R*2 which reads the numbers from the tape, multiplies them by 2^{-34}, and places them in sequence in mD, $(m + 2)D$, $(m + 4)D$, etc. Each integer is terminated by the letter *F*, with the exception of the last of the sequence, which must be terminated by π *T Z*. When this control combination is read it will return control to the initial orders. They will then read in the remainder of the program, part of which will be written over the orders of *R*2.

Parameter: *T m D* must be punched immediately after *R*2.

Note: See Part 3 for detailed program.

*R*9 *Input of positive integers during input of orders* (*standard form for regular use*).

Special; 15 storage locations.

The actual orders of this subroutine are identical with those of *R*2, but *R*9 is intended always to be placed in locations 56 to 70 inclusive, and to remain there throughout the input of a whole program, being used any number of times. Each time it is used it will read a sequence of positive decimal integers and place them in consecutive long storage locations.

Notes: 1. The subroutine tape commences with *P K T* 56 *K*, so that it may be copied immediately at the head of a program tape. It does *not* have *E* 13 *Z* at the end, so that it is not automatically obeyed after being read.

2. $R9$ is called in by the control combination E 69 K T m D. This is followed by the integers, each terminated by F except the last, which is terminated by π to return control to the initial orders. After this must be punched a control combination to restore the transfer order, e.g., T Z. The integers will be placed in mD, $(m + 2)D$, $(m + 4)D$, etc.

3. Negative integers may be read if 2^{35} is added to each before punching.

$R23$ *Input of one short or long positive integer by means of telephone dial.*

Closed A; 17 storage locations.

When $R23$ is called in the machine stops on a Z-order. The required integer (≥ 0) may then be dialed, most significant digit first, and terminated by pressing the Reset button. If a short integer (that is, one less than 65536) is dialed it will be placed in $0F$; a long integer will be placed in $0D$.

$R29$ *Input of one positive integer punched in output code.*

Closed A; 33 storage locations.

Notes: 1. $R29$ is used to read back into the store results which have previously been punched onto an output tape.

2. The symbol F is treated as if it were the digit 0; M is treated as a decimal digit of value 15. These anomalies arise through the extreme compactness of the subroutine; they do not give trouble in practice since the symbols F and M are unlikely to be encountered on a normal output tape.

3. When $R29$ is called in it ignores all symbols until a decimal digit is encountered. It then reads digits until any symbol which is not a digit (except, of course, F or M) is encountered; this terminates the integer. Thus integers to be read by $R29$ must consist of continuous sequences of digits and must not include spaces or other layout symbols.

4. See Part 3 for detailed program.

$R30$ *Extension of the initial input routine to read code letter S.*

Special; 10 storage locations.

The initial input routine wired into the EDSAC (see Appendix 3) does not recognize the code letter S. $R30$ provides an extension to the initial input routine so that this code letter is treated during input in the way described in Part *I*, Section 1–13. The orders of $R30$ may be placed anywhere in the store, although it is recommended that they should be placed in locations 1014 through 1023. The library tape begins with PZT 1014 K to facilitate this. Every program using the S facility must begin by placing a copy of $R30$ in the store.

Notes: 1. The orders in $27F$ and $28F$ are modified as the orders of $R30$ read in.

2. An order with zero address must *not* be terminated by code letter S.

3. See Part 3 for detailed program.

$R33$–36 *Standard set of subroutines to read sequences of long numbers.*

Number	Type	Description	Locations used	Working space
$R33$	Special	Input of a sequence of signed long decimal fractions. Called in by: n B m D mD is to be $n + 1$ A $n + 1$ F location of first number read $n + 2$ F s F s is location of first order of $R33$	46 (even)	$4D$, $6F$
$R34$	Special	Input of a sequence of signed long decimal fractions during input of orders. $R34$ is followed on the input tape by T m D and the sequence of numbers, which will then be read into mD, $(m + 2)D$, etc.	47 (even)	
$R35$	Special	Input of a sequence of signed long integers. Called in by: n B m D mD is to be $n + 1$ A $n + 1$ F location of first number read $n + 2$ F s F s is location of first order of $R33$	30	
$R36$	Special	Input of a sequence of signed long integers during input of orders. $R36$ is followed on the input tape by T m D and the sequence of numbers, which will then be read into mD, $(m + 2)D$, etc.	28	

Accuracy: Decimal fractions having less than eleven digits are converted to binary form with a maximum error of $(0.55) \cdot 2^{-34}$.

Notes: 1. Each number is punched as sign followed by any number of decimal digits. The last number of a sequence is terminated by the character M.

2. R33–36 will recognize only symbols corresponding to decimal digits, $+$, $-$, M, and also J, which is equivalent to the decimal number 10. All other symbols, occurring in any position on the input tape, will be ignored. The correct reading of numbers will thus not be upset by the occurrence of layout symbols on the tape. Note, however, that $C(m - 2)D$ will be destroyed unless a sign is punched in front of the first decimal digit to be read.

3. If desired, R34 or R36 may be used more than once during input of orders. After the first time they should be called in by transferring control to the first order, with $T\ \ m\ \ D$ in the accumulator.

R37 *Input of one signed decimal fraction.*

Closed B; even; 34 storage locations; uses $4D$, $6F$.

Reads one signed decimal fraction and places it in the accumulator. The fraction should be punched without a decimal point (which is assumed to be at the left-hand end), and with a plus or minus sign after the last digit. For example,

$$+0.8571 \text{ should be punched as } 8571+$$

and

$$-\tfrac{1}{2} \quad \text{should be punched as } 5-.$$

Any number of decimal digits may be punched, although digits after the eleventh will have no effect.

Accuracy: See R33–36.

Notes: 1. Symbols for J, π, S, Z, K, "erase," F, θ, and D, and also blank tape, will be ignored.

2. Symbols for M, Δ, L, X, G, A, B, C, and V will cause errors if they are read by R37.

3. ϕ will have the same effect as $+$.

4. If R37 is called in without the accumulator being clear, the operation of the subroutine will not be impaired, although the initial content of the accumulator will be discarded.

5. See Part 3 for detailed program.

S. Subroutines for evaluating fractional powers.

S3 Cube root.

Closed A; 25 storage locations; uses $4D$, $8D$; time, approx. 1 sec.

Forms cube root of $C(6D)$ and places result in $0D$. $C(6D)$ may be positive or negative and is not changed by the action of the subroutine.

Accuracy: $\pm 2^{-34}$.

Note: See Part 3 for detailed program.

[This subroutine is mainly included as an example of a digit-by-digit process. Such processes are rather slow, but are very simply programmed.]

S10 Square root of double-length number held in the accumulator.

Closed B (see Note 1); 37 storage locations; uses $4D$; time $=$ $(0.2 + 0.01n)$ sec, if $C(\text{Acc})$ is in the range 2^{-2n}, 2^{-2n-2}.

Replaces $C(\text{Acc})$ by $\sqrt{C(\text{Acc})}$, wheıe $C(\text{Acc})$ is a double-length number.

Accuracy: 32–33 significant binary digits. If the answer is small many of these digits may be in the less significant half of the accumulator.

Notes: 1. This is a normal closed B subroutine but, in order to deal correctly and quickly with the case $C(\text{Acc}) = 0$, it should be called in by the orders

$$\begin{array}{llll} p & B & p & F \\ p+1 & F & m & D \end{array} \qquad (m \text{ is location of first order of } S10).$$

$S10$ will then be by-passed if $C(\text{Acc}) = 0$, thus giving the correct result with no waste of time.

2. If $C(\text{Acc}) < 0$ the machine will come to a dynamic stop on the third order of $S10$.

3. The answer, left in the accumulator, is not rounded off.

S11 Reciprocal square root of double-length number held in the accumulator:

Closed B; 37 storage locations; uses $4D$; time $=$ $(0.2 + 0.01n)$ sec, if $C(\text{Acc})$ is in the range 2^{-2n}, 2^{-2n-2}, where $0 \leq n \leq 33$.

Forms, and places in $4D$, the quantity $C(0D)/\sqrt{C(\text{Acc})}$, where $C(\text{Acc})$ is a double-length number held in the accumulator.

Accuracy: Maximum error $\simeq 2^{-33}$.

Notes: 1. If $C(\text{Acc}) < 0$, the machine will come to a dynamic stop on the third order of $S11$.

2. If $[C(0D)]^2 > C(\text{Acc})$, the machine will stop on a ϕ-order in storage location 16θ or 27θ.

3. If $C(\text{Acc}) = 0$, the program will go into a loop.

4. See Part 3 for detailed program.

T. Subroutines for calculating trigonometric functions.

T7 Sine.

Closed A; even; 36 storage locations; time $= 0.081$ sec.

Forms and places in $4D$ the quantity $\frac{1}{2} \sin [2 \cdot C(4D)]$, where $|2 \cdot C(4D)| \leq \pi/2$. $R9$ must be in the store when $T7$ is read.

Accuracy: Maximum error is $\pm 2^{-33}$.

Note: See Part 3 for detailed program.

T9 Tangent.

Closed A; even; 46 storage locations; time $= 0.155$ sec.

Forms and places in $4D$ the quantity $\tan C(4D)$, where $-\pi/4 < C(4D) < \pi/4$. $R9$ must be in the store when $T9$ is read.

Accuracy: Maximum error is 2^{-33}.

[This subroutine is based on the use of a Chebyshev series.]

T11 Cosine.

Closed B; even; 44 storage locations; uses $4D$; time $= 0.12$ sec.

Replaces $C(\text{Acc})$ by $\frac{1}{2} \cos \pi[C(\text{Acc})]$.

Accuracy: Maximum error has modulus $< 2^{-34}$.

Notes: 1. The accuracy of the result is not affected if $C(\text{Acc})$ differs from the required argument by a multiple of 2. Thus the function for a large angle can be evaluated by deliberately exceeding the capacity of the accumulator. For example, to calculate $\frac{1}{2} \cos 2^n \pi x$, where $x = C(0D)$, we may use

$$\begin{array}{llll} 0\theta & A & & D \\ 1 & L & 2^{n-2} & F \end{array}$$
$$\left.\begin{array}{llll} 2 & B & 2 & \theta \\ 3 & F & \dots & \end{array}\right] \text{Orders calling in } T11.$$

2. If it is required to calculate $\frac{1}{2} \sin \pi x$, $(x - \frac{1}{2})$ should be placed in the accumulator before $T11$ is called in.

3. See Part 3 for detailed program.

Z. Post-mortem routines.

Z5 Standard set of post-mortem routines.

Routine	Form of post-mortem information punched	Storage locations*	Location of first order
*Z*1	Short fractions	47	odd or even
*Z*2	Long fractions	47	odd or even
*Z*3	Short integers	57	even
*Z*4	Long integers	58	even
*Z*5	Orders	64	odd or even

*Other than those required for the initial input routine.

Any of the above routines may be used to punch on the output tape, in appropriate form, the contents of a sequence of storage locations, the first of which is specified by dialing a two-figure integer. When sufficient information has been punched the machine must be stopped manually (see Note 9). The method of use is as follows:

Either (i) Place the post-mortem tape in the reader, beginning on the initial blanks, and press the Start button. When the tape stops, dial *three* digits to specify the location into which the first order of the post-mortem routine is to be placed. When the tape stops again, dial *two* digits to specify $n/10$, where n is the first location for which post-mortem information is required. Note that n must be a multiple of ten.

or (ii) Place the tape in the reader, beginning on the blanks preceding the second section; press the Start button. The orders of the post-mortem routine will then be placed in a sequence of storage locations beginning at 800. When the tape stops, dial two digits to specify $n/10$ as described in (i).

Notes: 1. Post-mortem information cannot be obtained for parts of a program occupying storage locations 0 to 45 inclusive.

2. Long, or short, decimal fractions are printed (unrounded) as eleven, or five, digits, preceded by the sign.

3. Integers, up to $(2^{34} - 1)$, are printed accurately; the first significant digit is preceded by the sign, nonsignificant zeros being completely omitted.

4. The quantities 0 and -1, whether treated as integers or as fractions, are punched as $+$ and $-$, respectively.

5. An order is punched as function letter followed directly by address (in which nonsignificant zeros are suppressed), and terminated by

a space if the order refers to a short location, or an asterisk if it refers to a long location.

6. The information punched is preceded by the number of the first storage location from which it is taken.

7. A space is punched after every item of information other than an order terminated by an asterisk.

8. Carriage return and line feed symbols are punched after every ten short items, or five long items.

*Z*7 *Universal post-mortem routine.*

172 storage locations; even.

This routine may be used to punch post-mortem information in any of the five forms described below. The method of use is similar to that described for *Z*1 through *Z*5, but the first location from which post-mortem information is to be punched is not dialed, but is punched on a supplementary tape. This supplementary tape must be inserted in the reader when the machine stops at the end of the *Z*7 tape.

The supplementary tape must be specially punched for each use of *Z*7. It contains a number of items, each specifying a series of locations and the form in which the contents of those locations are to be punched. An item consists of $aXb\Sigma$, where a and b are absolute addresses specifying the beginning and end of the sequence of storage locations from which post-mortem information is required, and Σ is a code letter specifying the form in which that information is to be printed, thus:

Code letter	*Printed form of post-mortem information*
S	short fractions
π	long fractions
K	short integer
Z	long integer
J	order

After an item is read, the addresses a and b will be punched, separated by a space and followed by carriage return and line feed symbols. The post-mortem information will then follow, with carriage return and line feed symbols after every ten short numbers or orders, or every five long numbers. At the end of the information specified by the item, an extra line feed will be punched. The next item will then be read and the process repeated.

Z8 Comparison post mortem routine.

66 storage locations; even.

This may be used, after a program has stopped, to punch out those orders which have changed during the course of the program. For use, the *Z8* tape should be placed in the tape reader and the Start button pressed. When the first few rows of tape have been read the machine will stop. At this point a three-figure integer must be dialed to specify the location into which the first order of *Z8* is to be placed. When the rest of the tape has been read the machine will stop. The original program tape should then be placed in the reader, and the Reset button pressed. The orders of the program will then be read in again and compared with those in the store. Each time there is a discrepancy, the number of the storage location concerned will be punched together with the order in the store, which will then be replaced by the order from the tape.

Notes: 1. When a program has been completely restored, under the control of *Z8*, it may be run again, if so desired.

2. If required, a post-mortem may be obtained for part of a program, provided that this part is self-contained (e.g., provided that all preset parameters are correct).

3. Orders referring to a long-storage location will be terminated by an asterisk when printed out.

4. Since the address of the Transfer Order (in location 22) is used to initiate each comparison, control combinations may cause useless information to be punched. For example,

$$\begin{array}{lll} T & n & \pi Z \\ \alpha & m & F \end{array} \qquad \text{(where } \alpha \text{ is any function letter)}$$

will compare $C(n\pi)$ with $m\ F$. Thus, unless $C(nF) = 0$, $C(n + 1)F$ will be punched out.

Also, $E\ n\ \ K\ P\ F$ or $Z\ n\ \ K\ P\ F$

will punch out $C(nF)$ unless $C(nF) = 0$,

but $Z\ \ K\ P\ F$

will not cause anything to be punched.

5. Interludes will cause many orders to be punched.

PART THREE

PROGRAMS OF SELECTED EDSAC LIBRARY SUBROUTINES

The following notation is used on all library program sheets.

Entry points: If control may arrive at an order by being transferred there by a jump order the location of the latter (relative to the first order of the subroutine) is shown on the extreme left, with an arrow pointing to the location of the order to which control is transferred, e.g.,

16 → 23 T 6 θ.

Unconditional transfers of control: A horizontal line is drawn underneath every jump order which is intended to produce a transfer of control each time it is encountered.

Variable orders: Orders and pseudo-orders which are to be changed during the course of the calculation are shown in brackets.

Pseudo-orders: A double vertical line is drawn on the left of the contents of all storage locations which are intended never to be obeyed as orders.

In the program sheets which follow, the code letter S is freely used, in orders involving the B-register, in the way described in Part 1. To use a subroutine in which this code letter occurs it would be necessary to have subroutine R30 in the store at the time the subroutine was read. In practice, in the subroutines actually used on the EDSAC, this necessity has been avoided by subtracting the address from 1024 in the case of an order which would be terminated by code letter S, and by adding 1024 to the address in an order in which the B-digit must be 1. For example,

TS 2 F would be written as T 1026 F
B 2 S " " " " B 1022 F
BS 2 S " " " " B 2046 F.

The content of the B-register is sometimes denoted by b.

A11 Arithmetical operations on real numbers expressed in floating decimal form.

For representation of numbers see Part 2. The number in the floating decimal accumulator (f.d.a.) is here referred to as $y \cdot 10^q$, and the operand as $x \cdot 10^p$.

Parameters:

```
Preset:  H | P s D |                  [numerical part]
         N | P t F |  Location of     [exponent      ]  of f.d.a.
```

Preset by subroutine:

```
         M | P 103 θ |
         Δ | P     θ |
         L | E  56 θ |  Dynamic stop order
```

```
           E    69 K  |        ]  Call in R9 to read following
           T   9πM    |        ]    constants
  9πM    1 71798 69183 F  |  1
 11πM      17179 86918 F  |  10^{-1}
 13πM       1717 98692 F  |  10^{-2}
 15πM        171 79869 F  |  10^{-3}
 17πM         17 17987 F  |  10^{-4}
 19πM          1 71799 F  |  10^{-5}
 21πM            17180 F  |  10^{-6}
 23πM             1718 π  |  10^{-7}
           T         Z
      0    A    46 θ   ]
94 →  1    A     2 F   ]  Form and plant order 3
      2    T     3 θ   ]
      3   (H       F)  ]  Place interpretive order in multiplier register
      4    C     9 M   ]    and in accumulator
      5    E    20 θ      Jump if interpretive E- or T-order
      6    R   256 F   ]  Form jump order specifying an address de-
      7    A       M   ]    pending on function of interpretive order
      8    T    19 θ   ]
      9    C     1 M   ]  Form and plant A-order speci-   ]
     10    T    11 θ   ]    fying same address as inter-  ]
                       ]    pretive order                 ]
     11   (A       F)     Select operand                  ]  Unpack
     12    U    22 θ      Store more significant half     ]  operand for
     13    L    32 F   ]  Remove exponent, p              ]  interpretive
     14    R    32 F   ]                                  ]  A-, B-, and
     15    U       D      x · 2^{-10} to 0D               ]  V-orders
     16    S    22 θ   ]                                  ]
     17    R    64 F   ]  -2^{-14}p to 22θ                ]
     18    T    22 θ   ]                                  ]
```

```
          19   (E      F)   *
  5 →     20    U   22 θ    Interpretive order to 22θ and 64θ
          21    T   64 θ
          22   (E      F)   **
          23    A      H
          24    G   30 θ    If number in f.d.a. is positive,
          25    T      H      change sign and set order 63
          26    A   23 θ      to A H
          27    T   63 θ
          28    S      H
          29    E   61 θ    Test if f.d.a. contains zero
  24 →    30    A    2 M    Form (10 − y) · $2^{-11}$
          31    E   49 θ
          32    H  11πM     1/10 to multiplier register
  39 →    33    T      H    (10 − y) · $2^{-11}$      Cycle to
          34    A    5 M                              multiply
          35    A      N    Adjust exponent           y by neg-
          36    T      N                              ative
          37    V      H                              power of
          38    A    4 M    Add 9 · $2^{-11}$         10 if nec-
          39    G   33 θ                              essary
          40    E   49 θ
  50 →    41    T      H
          42    S    5 M                              Cycle to
          43    A      N    Adjust exponent           multiply
          44    T      N                              y by pos-    Interpretive
          45    A      H                              itive        T-orders
          46    L    1 F    Multiply by ten           power of     only
          47    A      H                              10 if nec-
          48    L      D                              essary
  31,40 → 49    S    4 M    Subtract 9 · $2^{-11}$
          50    E   41 θ
          51    S    3 M    Subtract $2^{-11}$
          52    Y      F
          53    T      H    Final value of −y · $2^{-11}$ to 0H
          54    A      N    q · $2^{-14}$
          55    S    6 M
  56 →    56    P      L    Dynamic stop if
                              q ≥ 63                  Examine
          57    A    7 M                              exponent
          58    E   61 θ    Jump if q ≥ −63
          59    T      H    Set y = 0 if
          60    T      H      q < −63
  29,58 → 61    S    6 M    Re-form q · $2^{-14}$
          62    L   64 F    q · $2^{-6}$
          63   (S      H)   Add y · $2^{-11}$ †
          64   (T      D)   To store ††
```

	65	A	28	θ	Reset 63θ †		Interpretive T-orders only
	66	T	63	θ			
	67	S	6	M	Set $q = -63$		
	68	T		N			
	69	E	91	θ			
19 →	70	S		D	Change sign of x if to be added		Interpretive A- and B-orders only
	71	T		D			
19 →	72	H		D	Subtrahend to multiplier register		
	73	A		N	Form $2^{-14}(q - p)$		
	74	A	22	θ			
	75	E	84	θ	Jump if $q \geq p$	Larger number to f.d.a.	
19 →	76	E	95	θ	†††		
	77	T	64	θ	$-2^{-14}(q - p)$ to 64		
	78	H		H	Interchange numerical parts		
	79	A		D			
	80	T		H			
	81	S	22	θ	Larger exponent to $0N$		
	82	T		N			
	83	S	64	θ	$2^{-14}(q - p)$ to accumulator		
75 →	84	S	3	M	Jump if smaller number is negligible		
	85	E	92	θ			
	86	A	8	M	Divide numerical part of smaller number by appropriate power of ten		
	87	T	88	θ			
	88	(V		D)			
	89	A		H	Combine numbers		
101,102 →	90	Y		F			
69 →	91	T		H	Sum of difference to $0H$		
85 →	92	T	64	θ	Clear accumulator		
	93	A	3	θ	Prepare to change order 3		
	94	G	1	θ			
76 →	95	S	22	θ	Add exponents	Interpretive V-orders only	
	96	A		N			
	97	T		N			
	98	H		D	Multiply numerical parts		
	99	V		H			
	100	L	512	F			
	101	E	90	θ			
	102	G	90	θ			
M	0	E	78	θ			
	1	A	1023	D			
	2	P	160	F	$= 10 \cdot 2^{-11}$		
	3	P	16	F	$= 2^{-11}$		
	4	P	144	F	$= 9 \cdot 2^{-11}$		
	5	P	2	F	$= 2^{-14}$		
	6	P	126	F	$= 63 \cdot 2^{-14}$		
	7	P	252	F	$= 126 \cdot 2^{-14}$		
	8	V	25π	M			
		T	128	Z			

* Order 19 transfers control to 70θ in the case of an interpretive A-order,
to 72θ " " " " " " B-order,
and to 76θ " " " " " " V-order.

** 22θ contains the interpretive order itself, if the function is E or T (in the latter case the order plays no part in the calculation), otherwise 22θ is used to store $-p \cdot 2^{-14}$.

† Order 63 is always $S\ H$, unless an interpretive T-order is being obeyed and y is negative, in which case it becomes $A\ H$.

†† 64θ holds the interpretive order itself, if the function is E or T (in the former case control does not reach 64θ); in the case of interpretive A- and B-orders, 64θ is used to hold $-|q - p| \cdot 2^{-14}$. For all interpretive orders other than E, 64θ is used as a "dump" for order 92 to clear the accumulator.

††† If control reaches order 76 from order 75, C(Acc) must be <0, and control therefore proceeds to 77. If 76 is reached from 19, on the other hand, C(Acc) must be zero, and control will be switched to 95.

B2 Complex operations other than division.

Performs operations (including multiplication) on complex numbers. Uses as "multiplier register" 0H (real part) and 2H (imaginary part).

	46F	P	47 θ	N parameter
		T	50πZ	These tape entries do not go into the store as part of the program, but merely serve to clear 50$\pi\theta$ to ensure that the "sandwich digit" is zero when the constants PD and PF, appearing at the end of the subroutine, are planted there
		P	F	
		T	Z	
	0	A	N	Form $A\ n + 2\ F$
39 →	1	T	2 θ	
	2	(A	F)	Select interpretive order and plant in 16θ and 26θ
	3	U	16 θ	
	4	U	26 θ	
	5	A	N	Increase address in interpretive order by 2 and plant in 30θ
	6	U	30 θ	
	7	G	40 θ	Jump
	8	T	F	Clear accumulator
	9	A	17 θ	Place ineffective order in 18, 26 for operations other than V or N
	10	U	26 θ	
46 →	11	T	18 θ	
	12	A	4 H	Multiply by 2^{-34} to "unpack" real accumulator
	13	H	3πN	
	14	V	6 H	
	15	H	H	Operation on real part
	16	(V	F)	
	17	H	2 H	
	18	(N	F)	

```
        19   U    4 H
        20   S    4 H
        21   L      F     "Pack" real accumulator
        22   L      F
        23   L   64 F
        24   T    6 H
        25   A    8 H
        26  (V      F)
        27   H    3πN     "Unpack" imaginary accumulator and per-
        28   V   10 H       form operations
        29   H      H
        30  (V      F)
        31   U    8 H
        32   S    8 H
        33   L      F     "Pack" imaginary accumulator
        34   L      F
        35   L   64 F
        36   T   10 H
        37   A    2 F
        38   A    2 θ
        39   G    1 θ
   T → 40   A    1 N     Jump if interpretive V-order
        41   E   45 θ
        42   A    2 N     Jump if not interpretive N-order
        43   E    8 θ
        44   A    5 N
  41 → 45   S    5 N
        46   G   11 θ     Jump
  N      0   P    2 F
         1   Q      F
         2   I      F
         3   P      D
         4   P      F
         5   J      F
```

C27 Check-point subroutine.

```
             G      K
             O   40 L     Letter shift
             V      F
             O   40 K     Carriage return
             W      F
             T   27 K     Modify initial input routine to allow S to
             F   28 θ       be used to terminate a pseudo-order
             F      θ       specifying a check point
             T      Z
```

```
28F →  0 │  A    19 θ F │
       1 │  U    14 θ   │ Form orders referring to check point
       2 │  M     8 F   │
       3 │  T    21 θ   │
       4 │  A    22 F   │
       5 │  U    13 θ   │ Form T-orders to plant
       6 │  A     2 F   │
       7 │  U    15 θ   │ Required order and increase C(22) by
       8 │  A     2 F   │   P 4 F
       9 │  U    18 θ   │
      10 │  A     2 F   │
      11 │  U    22 F   │
      12 │  S    19 F   │ Plant output order
      13 │ (T       F)  │
      14 │ (A       F)  │ Plant order to be replaced
      15 │ (T       F)  │
      16 │  A    14 θ   │
      17 │  M    26 θ   │ Plant F-order for return
      18 │ (T       F)  │
      19 │  A    13 θ   │
      20 │  M    25 θ   │ Plant blocking order
      21 │ (T       F)  │
      22 │  A    40 F   │
      23 │  A    27 θ   │
      24 │  F    22 F   │ Return to initial input routine
      ───┼──────────────┤
      25 │  F       F   │
      26 │  F     1 F   │
      27 │  K  4096 F   │
27F → 28 │  A    43 F   │ π facility of initial input routine
      29 │  F     8 F   │
      ───┼──────────────┤
      30 │ (          ) │ Becomes output order      The contents of
      31 │ (          ) │ Becomes order from n      these 4 storage loca-
      32 │ (          ) │ Becomes F n + 1 F         tions refer to the
      33 │ (          ) │ Becomes symbol to be      first check point, in
         │              │   printed                 storage location n.
      ..  │              │ and so on, for further check points.
      ..  │              │
```

*C*31 *Trace of function letters, with delayed start*

```
         │  T       Z   │
       0 │ (A       H)  │ 0θ later stores C(0H), stores C(Acc), and
       1 │  T       θ   │   is used as working space
       2 │  A    11 θ   │ Plant blocking order in 0H
       3 │  T       H   │
```

```
             4 |  O      8 θ  |  Carriage return               ┐
             5 |  O      9 θ  |  Line feed                     │
             6 |  O     10 θ  |  Letter shift                  │
             7 |  E     25 F  |  Return to initial input rou-  │Over-written
               |--------------|    tine after interlude        │after interlude
             8 || W        F  |                                │
             9 || I        F  |                                │
            10 || V        F  |                                │
            11 || F     37 θ  |                                ┘
               |--------------|
               |  E        Z  | ┐
               |  P        F  | │ Enter interlude
               |  T      3 Z  | ┘
             3 || Q        F  |
             4 || W        F  |
             5 || I        F  |
             6 || O        F  |
             7 || S        F  |
             8 || K   4096 F  |
             9 || P     10 θ  |
     38 → 10 |  O      4 θ  |  Carriage return ┌Enter from 38 if C(0H)
          11 |  O      5 θ  |  Line feed       └causes a jump
          12 |  U        θ  | ┐
          13 |  S        θ  | ┘ Store C(Acc)
          14 |  A      2 θ  | ┐
          15 |  A      2 θ  | │ Change the select order if a jump occurs
          16 |  A     62 θ  | ┘
     43 → 17 |  U     19 θ  | ┐
          18 |  S     19 θ  | ┘ Plant new select order
          19 | (G   2047 H) |  The select order (initially A(−1)H)
          20 |  U     38 θ  |
          21 |  E     25 θ  | ┐
          22 |  A      5 θ  | │
          23 |  A      5 θ  | │
          24 |  E     26 θ  | │ Change sign digit and print function letter
     21 → 25 |  A      8 θ  | │   without exceeding capacity
     24 → 26 |  U      1 θ  | │
          27 |  G      1 θ  | ┘
          28 |  E     44 θ  |
          29 |  A      6 θ  |
          30 |  E     63 θ  |  Jump if function is U,I,O,J,π,S,Z,K, or
             |              |    "erase"
          31 |  A      3 θ  |
          32 |  E     52 θ  |  Jump if function is Y
          33 |  A      4 θ  |
          34 |  G     50 θ  |  Jump if function is P,Q,W, or E
45,49,61 → 35 |  U      1 θ  | ┐
          36 |  S      1 θ  | ┘ Clear accumulator
```

```
Enter     → 37 |  A       θ |                              ]Restore C(Acc)
from        38 | (T       H)|  Later stores current order  |(and C(0H)
blocking       |            |                              ]initially)
order       39 |  U       θ | ]
            40 |  S       θ | ] Store C(Acc)
            41 |  A    19 θ | ]
            42 |  A     2 F | ] Increase address in select order
            43 |  F    17 θ |
               |------------|
       28 → 44 |  S     7 θ |
       64 → 45 |  E    35 θ |  Jump if function is A,B,C,V,π,S,Z,K, or
               |            |    "erase"
            46 |  A     3 θ |
            47 |  A    52 θ |  Jump if function is G or J
            48 |  A     6 θ |
            49 |  E    35 θ |  Jump if function is
               |            |    X,L,Δ,N,M,H,φ,D,θ,U,I, or O
       34 → 50 |  A     3 θ |
       51 → 51 |  G    51 θ |  Dynamic stop if function is P,Q,W, or space
    32,47 → 52 |  R       D | ]
            53 |  U     2 θ | |
            54 |  S     2 θ | |
            55 |  S     2 θ | |
            56 |  S     2 θ | |Change address to 10θ if jump order
            57 |  A     9 θ | |
            58 |  A    38 θ | |
            59 |  U    38 θ | |
            60 |  L       D | ]
            61 |  F    35 θ |
               |------------|
            62 | || A     F |
       30 → 63 |  A    62 θ |
            64 |  F    45 θ |
               |------------|
```

*D*11 *Division, with double-length dividend held in the accumulator.*

Replaces $C(\text{Acc})$ and $C(0D)$ by $C(\text{Acc})/C(4D)$, where $C(\text{Acc})$ is a double-length number held in the accumulator.

Repetitive process:

$$a_{n+1} = -a_n c_n + a_n, \qquad (a_0 = \text{dividend}, (c_0 + 1) = \text{divisor});$$
$$c_{n+1} = -c_n{}^2;$$
Stop when $c_n = 0$.

```
  |  G     K |
0 |  F   8 θ |  Jump into shift cycle
  |----------|
1 | || I   F |  ½
```

```
12 →  2 | A   4 D |
      3 | S   1 θ |  Double divisor in 4D
      4 | U   4 D |
      5 | S   4 D |
      6 | A     D |
      7 | L     D |  Double dividend in 0D and accumulator
 0 →  8 | U     D |
      9 | S     D |
     10 | A   4 D |
     11 | A   1 θ |  Jump unless divisor ≥ 1/2 or < − 1/2
     12 | E   2 θ |
     13 | A   1 θ |  Add 1/2
     14 | G  19 θ |  Test sign of (shifted divisor —1)
     15 | T   4 D |
     16 | S     D |  Change signs of shifted dividend and divisor
     17 | T     D |
     18 | S   4 D |
14
27 → 19 | T   4 D |  c_n
     20 | H   4 D |
     21 | N     D |
     22 | A     D |  a_n+1                        Repetitive cycle
     23 | Y     F |
     24 | T     D |
     25 | N   4 D |
     26 | Y     F |  c_n+1
     27 | G  19 θ |  Jump until c_n+1 = 0
     28 | A     D |
     29 | FS  2 F |  Link order
```

E6 Exponential function, large range.

Forms exp $(2^p \cdot y)$, where $y\ (<0) = C(R)$ and $p \geq 1$.

```
       | E        69   K      |  Call in R9 to read in following coeffi-
       | T        20   πθ     |    cients in power series
 20πθ  |          2 60054  F  |
 22πθ  |         31 23906  F  |
 24πθ  |        235 85378  F  |
 26πθ  |       1430 07273  F  |
 28πθ  |       7157 73946  F  |
 30πθ  |      28633 01149  F  |
 32πθ  |      85899 33588  F  |
 34πθ  |     171798 69147  F  |
 36πθ  |     171798 69184  π  |
       | T             Z      |
    0  | K        18   θ      |  Copy b for link order
    1  | B        16   S      |
       2 | A    20 πθ |  Last coefficient in power series
10 →   3 | BS    4 F  | ┐
       4 | T       D  | │
       5 | V       D  | │
       6 | AS   34 πθ | │  Sum power series two terms at a time
       7 | T       D  | │
       8 | V       D  | │
       9 | AS   36 πθ | │
      10 | J     3 θ  | ┘
      11 | B       H  | ┐
17 →  12 | BS    1 S  | │
      13 | T       D  | │
      14 | H       D  | │  Square result p times
      15 | V       D  | │
      16 | Y       F  | │
      17 | J    12 θ  | ┘
      18 | (B      F) |
      19 | FS    2 F  |    Link order
```

F2 Inverse interpolation, or solution of $f(x) = 0$ (second-order process).

Places in $0H$ a solution of $f(x) = 0$, where $f(x)$ is defined by an auxiliary, closed A subroutine. Working space is allocated as follows.

$0H$ (long)	x_c
$2H$	x_a
$4H$	x_b
$6H$	$-f(x_a) = -f_a$ (say)
$8H$	$-f(x_b)$ or $-2^{-m}f(x_b) = -F_b$ (say).

x_a and x_b are two values of x such that f_a and F_b have opposite signs. x_c is a value obtained by linear inverse interpolation between x_a and x_b. The auxiliary subroutine places f_c in $0D$. If the sign of f_c is opposite to that of f_a, then F_b is replaced by f_a, f_a by f_c, x_b by x_a, and x_a by x_c. If the sign of f_c is the same as that of f_a, then f_a is replaced by f_c, x_a is replaced by x_c, and F_b is halved. This last operation prevents x_b remaining unaltered for many cycles since, if it did, the process would become a first-order one, or fail to converge altogether. At the start, x_a and x_c are the given trial values x_1 and x_2, and $6H$ is cleared. This is to ensure that, initially, f_a and f_c are treated as of opposite sign. The first two function values to be calculated are $f(x_1)$ and $f(x_2)$. The process terminates either when $f_c = 0$ or when $|x_a - x_b| \leq 2^{-34}$.

```
47F | Δ   F |    M-parameter
    | T   Z |
  0 | A 3 F | ┐  Plant link order
  1 | T 7 θ | ┘
```

```
        2 | T  6 H | Clear 6H
  56 →  3 | A  3 θ | Call in auxiliary subroutine
        4 | G    N |
 Aux →  5 | H    D |
        6 | N    D |                       Test whether f_c = 0
  54 →  7 | (Z   F)| Link order
        8 | T  4 D | Clear accumulator
        9 | V  6 H | Test relative sign of f_a and f_c
       10 | G 17 θ |
       11 | T  4 D | Clear accumulator
       12 | A  2 H | x_a to x_b
       13 | T  4 H |
       14 | A  6 H | f_a to F_b                Opposite signs
       15 | T  8 H |
       16 | E 21 θ |
  10 → 17 | T  4 D | Clear accumulator
       18 | A  8 H |                          Same sign
       19 | R    D | Halve F_b
       20 | T  8 H |
  16 → 21 | S    D | f_c to f_a
       22 | U  6 H |
       23 | S  8 H |
       24 | E 29 θ |
       25 | T  4 D |
       26 | S    D |
       27 | T    D |
       28 | S  4 D |
  24 → 29 | A 57 θ |        f_c
  38 → 30 | T  4 D | Form ─────── in 0D by division process similar
       31 | Y    F |      f_a − f_c
       32 | H  4 D |   to that used in D7 and D11
       33 | N    D |
       34 | A    D |
       35 | T    D |
       36 | N  4 D |
       37 | Y    F |
       38 | G 30 θ |
       39 | H    D |
       40 | A    H | Plant new x_a
       41 | U  2 H |
       42 | S  4 H | (x_a − x_b) to 0D
       43 | T    D |
       44 | A    H |
       45 | V    D | New x_c
       46 | Y    F |
       47 | T    H |
        48 | A       D  |
        49 | G  52   θ  |
        50 | T       D  |
        51 | S       D  |  Test for |x_a − x_b| ≤ 2^-34
  49 →  52 | R       D  |
        53 | Y       F  |
        54 | E   7   θ  |
        55 | T       D  |  Clear accumulator
        56 | E   3   θ  |  Repeat
           |------------|
        57 | || Δ    M  |  = −1
```

F7 Interpolation in a table, using Neville's process.

Replaces C(Acc) by $f[C(\text{Acc})]$. The address of the storage location containing $f(0)$ is specified by a program parameter.

For a description of the interpolation process, see Milne's *Numerical Calculus*, p. 72.

```
           | T        Z  |
         0 | A   2044 N  |  B (2n − 4) F
         1 | TS   48πθ   |
  19 →   2 | BS     2 S  |
         3 | A     41 F  |
         4 | A     42 θ  |
         5 | U       πθ  |
         6 | U     41 F  |
         7 | A     43 θ  |
  16 →   8 | T        D  |                        These orders place
         9 | H        D  |                        r − 1/r in locations
        10 | N       πθ  |                        [46 + 2(n − r)]πθ, where
        11 | A       πθ  |  Division cycle as     r = 1, 2, . . . , (n − 1)
        12 | Y        F  |    used in D7
        13 | T       πθ  |    and D11
        14 | N        D  |
        15 | Y        F  |
        16 | G      8 θ  |
        17 | A       πθ  |
        18 | TS   48πθ   |
        19 | J      2 θ  |
        20 | A     47 F  |
        21 | R      2 F  |
        22 | A     44 θ  |  R 2^(12−m) F to 6θ
        23 | T      6 θ  |
        24 | H     47 F  |
        25 | V     46 F  |
        26 | N     46 θ  |  (n − 2)2^(−m−1) to 47θ
        27 | L        F  |
        28 | T     47 θ  |
        29 | S     47 F  ]  $-2^{-m}$ to 46θ
        30 | U     46 θ  ]
        31 | A      2 F  ]
  33 →  32 | L        D  ]
        33 | G     32 θ  ]  L $2^{m-2}$ F to 33θ
        34 | R      2 F  ]
        35 | A     27 θ  ]
        36 | T     33 θ  ]
        37 | S     46 F  ]
        38 | M     45 θ  ]  B (1026 — 2n) F to 14θ
        39 | T     14 θ  ]
        40 | H      8 F     Restore C(R)
        41 | F     34 F     Return to initial input routine after interlude
           |-------------
        42 || P  1024 F
        43 || K  5120 F
        44 || R       F
        45 || B     2 F
        46 || P     4 F
           | E        Z  ]  Enter interlude
           | P        F  ]
```

The above interlude calculates the constants and forms, and plants some of the orders required. It is followed immediately on the tape by:—

```
           | T        Z
         0 | K     42 θ     Plant link order
         1 | H     46 θ     $-2^{-m}$ to multiplier register
         2 | U      4 D
         3 | S     47 θ
         4 | ϕ        D
         5 | C        D
           | T      7 Z     (6θ contains R $2^{12-m}$ F)
         7 | AS     2 F     Pick up program parameter
         8 | A     45 θ
         9 | U     15 θ
        10 | T     25 θ
        11 | C        D
        12 | S      4 D
        13 | T      4 D
           | T     15 Z     (14θ contains B(1026–2n)F)
        15 | (Z       F)    Becomes A-order to pick out function value
           |                  from table
  41 →  16 | BS     2 F
        17 | K     40 θ
        18 | ϕ        H
        19 | S     40 θ  ]
        20 | M     44 θ  ]  Set order in 37θ
        21 | T     37 θ  ]
        22 | A      4 D  ]
        23 | S     46 θ  ]  Modify the argument
        24 | φ      4 D  ]
        25 | (Z       F)    Becomes A-order
        26 | B   1022 N
  39 →  27 | BS     2 S  ]
        28 | US     2 H  ]
        29 | SS       H  ]
        30 | φ        D  ]
        31 | H        D  ]
        32 | N      4 D  ]  Linear interpolation cycle
           | T     34 Z  ]  (33θ contains L $2^{m-2}$ F)
        34 | Y        F  ]
        35 | U        D  ]
        36 | H        D  ]
        37 | (Z       F) ]
        38 |  AS    2 H  ]
        39 | J     27 θ  ]
        40 | (Z       F)
        41 | J     16 θ
        42 | (Z       F)
        43 | FS     3 F     Link order
           |------------
        44 || N  1072πθ
        45 || A  1022πN
        46 ||              $-2^{-m}$            ]  Planted by interlude
        47 ||              $(n - 2)2^{-m-1}$    ]
```

*G*12 *Solution of one or more simultaneous differential equations by Runge-Kutta-Gill process*

```
           | T      6πZ  ]
           | P        F  ]  Clear 6D, 8D, and 10D to ensure that sand-
           | T      8πZ  ]    wich digits are zero when numbers are
           | P        F  ]    later planted as pairs of orders.
           | T     12πZ  ]
           | P        F  ]
         0 | K     47 θ     Copy b for link order
         1 | B      8 F     Set stage count
   46 →  2 | BS     2 S     Reduce stage count by 2
         3 | K     16 θ     Copy stage count in 16θ
         4 | B     14 θ  ]  Call in auxiliary subroutine so that it will
         5 | F        X  ]    return control to 16θ
           |------------
```

```
             6 | ‖(US    42  θ) | ] After interlude this long location contains
             7 | ‖ J      1  F  | ]   the sum (F  42  θ, R  D) +
                                |                           (N  1024  N, Y  F)
             8 | ‖ H    682  D  | ]
             9 | ‖ T    682  D  | ] −2/3
 Enter →    10 |  (A      6  θ) | ]                                        ┐
 inter-     11 |   F     14  θ  | ] Becomes +1/√2                          │
 lude          |----------------|                                        │
            12 | ‖ E    407  D  | ]                                        │
            13 | ‖ φ   1405  D  | ] −1/2                                   │
            14 |   T      6  θ  | ]                                        │ Interlude
            15 |   T     14πθ   | ] Becomes 0                              │
            16 |   S     12πθ   | ]                                        │
            17 |   T     10πθ   | ] +1/√2 to 10πθ                          │
            18 |   E     34  F  |   Return to initial input routine      ┘
               |----------------|
               |   E     10  Z  | ]
               |   P         N  | ] Enter interlude with C(Acc) = P  N
               |   T     17  Z  |
 Aux →      16 |  (           ) |   Stage count (B-order) planted by order in 3θ
            17 |   K     45  θ  |   Copy stage count
            18 |   HS     8πθ   |   Set stage multiplier
            19 |   S      6πθ   | ]
            20 |   U      6πθ   | ] Binary switch
            21 |   G     25πθ   | ]
            22 |   S     40πθ   | ] N  1024  N, Y  F, or         ] Orders planted in
            23 |   U     40πθ   | ]   F  42  θ, R  D to 40πθ     ] stages 2 and 4
            24 |   T     28πθ   |   Y  F or R  D to 28θ          ]
 21 →       25 |   B         Δ  |   Set 2n
 44 →       26 |   BS     2  S  |   Reduce by 2
            27 |   AS        N  |
            28 |  (R         D) |   Becomes Y  F during stages 2 and 3
            29 |   SS        M  |   −q_n
            30 |   U         D  |   t_n
            31 |   V         D  |
            32 |   R         L  |
            33 |   U      4  D  |   r_n
            34 |   AS        H  |   y_n
            35 |   φS        H  |   y_n
            36 |   A      4  D  | ]
            37 |   L         D  | ]
            38 |   A      4  D  | ] 3r_n
            39 |   L         L  | ]
            40 |   F     42  θ  |   Become [N  1024  N] during stages 2 and 3
            41 |   R         D  |          [Y        F]
 40 →       42 |   S         D  |   t_n
            43 |   φS        M  |   q_n
            44 |   J     26  θ  |   Cycle
            45 |  (B         F) |   Stage count
```

```
      46   J      2 θ    Cycle
      47   (B       F)
      48   FS     2 F    Link order
```

L4 Logarithm.

```
              G        K
         0    K     25 θ    Store b for link order
         1    B        F    Clear B-register
  4 →    2    BS     4 F  ⎤ Shift left until x (=C(Acc)) > 1, counting
         3    L        D  ⎥    in units of 4 in the B-register
         4    E      2 θ  ⎦
         5    A     27 θ  ⎤ y to 0D, where y = (2^p x − 3/2), and p =
         6    T        D  ⎦    number of places shifted
         7    K     50 θ    Record B 4p F in 50θ
         8    H    28πθ     2^-5 log_e 2 to multiplier register
         9    N     50 θ  ⎤
        10    V      1 θ  ⎥
        11    L        F  ⎥ (1/32) (log_e 3/2 + p · log_e ½)
        12    A    30πθ   ⎥
        13    T    50πθ   ⎦
        14    H        D    y to multiplier register
        15    A    52πθ   ⎤
        16    B     20 S  ⎥
 24 →   17    BS     4 F  ⎥
        18    T        D  ⎥
        19    V        D  ⎥ Evaluate power series
        20    AS   48πθ   ⎥
        21    T        D  ⎥
        22    V        D  ⎥
        23    AS   50πθ   ⎥
        24    J     17 θ  ⎦
        25    (B       F)
        26    FS     2 F    Link order
        ------------------
        27    I        F    ½
              E     69 K  ⎤ Call in R9 to read following coefficients
              T    28πθ   ⎦
      28πθ        3721 30559 F    2^-5 log_e 2
      30πθ        2176 82422 F    log_e 3/2
      32πθ          20 13466 F    a_9
      34πθ   3 43571 94535 F    a_8
      36πθ          43 67037 F    a_7
      38πθ   3 43518 74548 F    a_6
      40πθ         141 53462 F    a_5
      42πθ   3 43332 26603 F    a_4
      44πθ         530 23671 F    a_3
```

```
46πθ  | 3 42404 33717 F |  a_2
48πθ  |    3579 13950 F |  a_1
50πθ  |               F |  Later contains 1/32 (log_e 3/2 + p log_e 1/2)
52πθ  | 3 43585 24506 π |  a_10
      |       T    54 Z |
```

M18 Store repetitive pattern of orders

```
            |   G      K   |
15 →  0     |   U    9 θ   | ┐
      1     |   S   17 θ   | │
      2     |   U    6 θ   | │ Form and plant variable orders
      3     |   S   17 θ   | │
      4     |   M   16 θ   | │
      5     |   T    8 θ   | ┘
      6     |  (Z      F)  |   Becomes A f − c + m F
      7     |   L      D   |
      8     |  (Z      F)  |   Becomes S f − 2c + m F
      9     |  (Z      F)  |   Becomes T f + m F
     10     |   A    9 θ   | ┐ Return to initial input routine when last order
     11     |   S   18 θ   | │   copied into t
     12     |   E   25 F   | ┘
     13     |   A   18 θ   | ┐
     14     |   A    2 F   | │ Increase order in 9θ
     15     |   E      θ   | ┘
            |--------------|
     16     || S       F   |
     17     || O     c F   |   c = length of cycle
     18     || T     t F   |   Last order copied into t             ┐ Punched
            |   E      Z   |   Enter M18 with T t F in Acc          │  by
            |   T    f F   |   First order will be copied into f    ┘  user
```

M20 Set parameter value, by means of telephone dial, during input of orders.

This subroutine consists largely of control combinations. It requires no storage space, but uses 22F, 42F, and 43F, normally occupied by orders of the initial input routine, as working space.

```
  Tape entry                          Notes

  P          Z  |
  Z          K  | ┐ Stop machine; when digit r is dialed set Transfer
  M   2037   F  | ┘   Order to T(r − 10)F
  G          K  |   Copy address (r − 10) into 42
  P     10   K  |   Set C(22) = P 10 F
  P          Z  |   Add C(22) to 42 if (r − 10) < 0; if (r − 10) = 0
                |     leave unaltered
  T     43   K  | ┐
  P          θ  | ┘ Transfer C(42) to 43
```

```
          Tape entry                    Notes
         [Z        K  ]
         |M   2037 F  |
central  |G        K  |  Repeat for second digit dialed
section  |P     10 K  |
         |P        Z  ]
         |T     43 K  ]  Multiply C(43) by 10, add C(42), and place sum in 43
         [P π    0 θ  ]
          Z        K  ]
          M   2037 F  |
          G        K  |  Repeat for final digit dialed
          P     10 K  |
          P        Z  ]
          T     45 K  ]  Multiply C(43) by 10, add C(42), and place sum in 45
          P π    0 K  ]
          I     43 K  ]  Reset C(43) to P D, Transfer Order to T 46 F, and
          B      2 F  |    resume normal action of initial input routine
                 Q    ]
```

Notes: 1. If it is desired to dial more, or less, than three digits, the central section should be repeated an appropriate number of times, or omitted.

2. A preset parameter other than H may be set by suitably altering the control combination T 45 K.

M30 Sideways addition by Gillies-Miller method.

Counts the 1's in $C(R)$ and places the count, as $P\ n\ F$, in $0F$. The method is best explained by the example on page 190, which is given in full. The numbers in parentheses at the right of the example refer to the following notes:

Notes: 1. Consider $C(R)$ as a series of triads of binary digits (the left-hand triad being incomplete.)

2. Collate with $C(\pi H)$ to examine the left-hand digit of each triad.

3. Shift one place to the right; the middle digit of each triad in the accumulator may then be 0 or 1, while the other digits are all zero.

4. Collate with $C(2\pi H)$ to examine the middle digit of each triad of $C(R)$, including the sign digit; add the result to C(Acc).

5. Examine the sign digit; if it is a 1, subtract $C(14H)$ which adds one to the middle digit of the next triad to the right of the sign digit and makes C(Acc) positive.

6. Shift one place to the right; the count in each triad will then be ≤ 2 in most cases, although it may be 3 in the next triad to the right of the sign digit.

7. Collate with $C(4\pi H)$ to examine the right-hand digit of each triad of $C(R)$; add the result to C(Acc).

8. Place C(Acc) in the multiplier register and clear the accumulator. Consider the digits of $C(R)$ in groups of six, the left-hand group being incomplete.

```
C(R)       10  011  011  111  101  001  100  000  011  111  010  001      (1)
          ==============================================================
C(πH)      00  100  100  100  100  100  100  100  100  100  100  100      (2)
          --------------------------------------------------------------
C(Acc)     00  000  000  100  100  000  100  000  000  100  000  000
C(Acc)     00  000  000  010  010  000  010  000  000  010  000  000      (3)
C(2πH)     10  010  010  010  010  010  010  010  010  010  010  010      (4)
          --------------------------------------------------------------
C(Acc)     10  010  010  100  010  000  010  000  010  100  010  000

C(14H)     01  110  000  000  000  000  000  000  000  000  000  000      (5)
          --------------------------------------------------------------
C(Acc)     00  100  010  100  010  000  010  000  010  100  010  000
C(Acc)     00  010  001  010  001  000  001  000  001  010  001  000      (6)

C(4πH)     01  001  001  001  001  001  001  001  001  001  001  001      (7)
          --------------------------------------------------------------
C(Acc)     00  011  010  011  010  001  001  000  010  011  001  001

C(R)          00011    010011    010001    001000    010011    001001     (8)
            ============================================================
C(8πH)        01000    111000    111000    111000    111000    111000     (9)
            ------------------------------------------------------------
C(Acc)        00000    010000    010000    001000    010000    001000
C(Acc)        00000    000010    000010    000001    000010    000001    (10)
C(6πH)        00111    000111    000111    000111    000111    000111    (11)
            ------------------------------------------------------------
C(Acc)        00011    000101    000011    000001    000101    000010    (12)
 → C(R)        (a)       (b)       (c)       (d)       (e)       (f)

C(10πH)       00010    000010    000010    000010    000010    000010    (13)
            ------------------------------------------------------------
C(Acc)           00    000011    001000    001011    001100    010001
                  010011    010001    001100    001011    001000    000011   00
                 |______|

C(Acc)
 → C(R)       00110    010000    010110    011000    100010    100110    (14)
                                                               |______|
C(12πH)       00000    000000    000000    000000    000000    111110    (15)
            ------------------------------------------------------------
C(Acc)        00000    000000    000000    000000    000000    100110
                                           |____________________________|
                                                 P       19       F
```

9. Collate with $C(8\pi H)$ to examine the first three digits of each group.

10. Shift three places to the right to align with the last three digital positions in each group.

11. Collate with $C(6\pi H)$ to examine the last three digits in each group of $C(R)$; add the result to $C(\text{Acc})$, whose digits were aligned correctly in step 10.

12. Place $C(\text{Acc})$ in the multiplier register and clear the accumulator. The sum of each group of six digits is now ≤ 6. Call these sums a, b, c, d, e, and f.

13. Multiply by $C(10\pi H)$. The groups of six digits in the accumulator now represent the quantities

$$0,\ 2a,\ 2(a + b),\ \cdots,\ 2(a + b + c + d + e + f),\ \cdots,\ 2(a + b),\ 2a.$$

None of the quantities in brackets exceeds 36, and the group of digits containing the desired sum; $=2\Sigma a = 2(a + b + c + d + e + f)$, is scaled by 2^{-38}.

14. Shift four places to the left to give $2^{-33} \cdot \Sigma a$ as the six least significant digits of the more significant half of the accumulator. Place $C(\text{Acc})$ in the multiplier register and clear the accumulator.

15. Collate with $C(12\pi H)$ to give $2^{-33} \cdot \Sigma a$ in the accumulator. Plant this sum in $0F$ as the pseudo-order P 19 F.

The program is as follows (the six long, and one short, collating constants are placed in $0H$ through $14H$ by an interlude, details of which will be omitted).

		G	K	
	0	C	πH	Add left-hand digits of triads and shift one place to right
	1	R	D	
	2	C	$2\pi H$	Add sign digit and middle digits of triads
	3	E	5 θ	Jump if sign digit is 0
	4	S	14 H	Subtract $C(14H)$ if sign digit is 1
3 →	5	R	D	Shift one place to right
	6	C	$4\pi H$	Add right-hand digits of triads
	7	T	D	Place $C(\text{Acc})$ in multiplier register
	8	H	D	
	9	C	$8\pi D$	Add first three digits of each group of six
	10	R	2 F	Align groups of digits
	11	C	$6\pi H$	Add last three digits of each group
	12	T	D	Place $C(\text{Acc})$ in multiplier register
	13	H	D	
	14	V	$10\pi H$	Cross multiply
	15	L	4 F	
	16	T	D	Required sum in the form $n \cdot 2^{-33}$ is now represented by the six digits from 2^{-28} to 2^{-33}
	17	H	D	
	18	C	$12\pi H$	Add required six digits
	19	T	D	Place P n F in $0F$

M31 Serial correlation.

```
                T      24πZ
                P         F     Clear 24πθ
                T         Z
Enter →   0     A       4 θ     Plant B 2m F
          1     U       4 θ
          2     M      42 F
          3     T      50 F     Set X = P(2m + θ)
          4     (B        M)    Becomes B 2m F
          5     A      26πθ
   9 →    6     BS      2 S
          7     US     24πθ     Plant sequence of H- and V-orders
          8     A      24πθ       in 24πθ, . . .
          9     J       6 θ
         10     T         F     Clear accumulator
         11     A      47 F
         12     A      22 θ     Plant AS(m + n − 1)F in 4θ
         13     T       4 θ
         14     A      23 θ
         15     A      49 F     Plant SS(l + n − 1)F in 13θ
         16     T      13 θ
         17     A      21 θ
         18     A      49 F     Plant B 2l F in 18θ
         19     T      18 θ
         20     F      34 F     Return to initial input routine
                ------------
         21     B         L
         22     A    1023 N
         23     S    1023 N
         24     P       1 F
         25     P       1 F
         26     H         H
         27     V    1024 N
                E         Z     Enter interlude with P m F in
                P         M       accumulator
                T         Z
          0     K      28 X     Plant link order
          1     B         L
          2     K      21 θ
   5 →    3     BS      1 S
                T       5 Z     4θ contains AS(m + n − 1)F
                                  planted by interlude
          5     J       3 θ
          6     T         F
          7     B         M
  17 →    8     BS      1 S
          9     H    1024 H
         10     N         F     Add contribution to check sum
         11     A         Δ
         12     T         Δ
```

```
                       Order                Notes

                 T         14 Z      13θ contains SS(l + n − 1)F
           14    A       1023 N        planted by interlude
           15    A            F
           16    T            F
           17    J          8 θ      Compute check sum
                 T         19 Z      18θ contains B 2l F planted by
                                       interlude
   27X →   19    BS         2 S
           20    K         24 X
           21    (P           F)     Becomes count
           22    BS         1 S
           23    K         21 θ
                 E         25 K   ┐ Set initial input routine to read re-
                 T         25 X   │   maining orders into 25X, . . . ,
                                  ┘   29X.
          24θ    H   (m − 1) H    ┐
           ..    VS  (m − 1) N    │
           ..          ..         │
           ..          ..         │ Sequence of H- and V-orders planted
           ..          ..         │   by interlude
                 H          1 H   │
                 VS         1 N   │
                 H            H   │
  (24 + 2m − 1)θ VS           N   ┘
          24X    (             )     Becomes count
          25X    AS         2 Δ
          26X    TS         2 Δ      Compute products
          27X    J         19 θ
          28X    (P          F )
          29X    FS         2 F      Link order
```

P31 *Print signed fraction with page layout.*

```
                T    54πZ
                I       F
                T      πZ   ┐
                C   819 F   │
                T       Z   │ −1/10 to πθ
             ‖  L  1229 F   │
             ‖  T     2 Z   ┘
          2     H      πθ     −1/10 to multiplier register  ┐ Interlude
  9,11 →  3     T       F                                   │ to calculate
          4     N    54πθ   ┐                               │ round-off
          5     Y       F   │ Round off cycle               │ quantity
          6     T    54πθ   ┘                               ┘
          7 │ A      F  ┐
          8 │ L      D  │
          9 │ E    3 θ  │ Examine digit layout constant  │ Interlude
         10 │ L      D  │                                │ to calculate
         11 │ G    3 θ  ┘                                │ round-off
         12 │ H    8 F    Restore multiplier for initial │ quantity
            │               input routine                │
         13 │ O   11 θ    Figure shift                   │
         14 │ E   25 F    Return to initial input routine┘
            │ E    2 Z  ┐ Enter interlude with digit lay-
            │ P      H  ┘   out constant in accumulator
            │ T      Z
          0 │ A    3 F  ┐ Plant link order
          1 │ T   48 θ  ┘
          2 │ A   59 θ  ┐
          3 │ G   11 θ  │
          4 │ O   49 θ  │
          5 │ O   50 θ  │
          6 │ S    2 F  │
          7 │ E   10 θ  │ Page layout: n columns, and blocks of 5 rows
          8 │ O   50 θ  │
          9 │ A   60 θ  │
    7 →  10 │ S   58 θ  │
    3 →  11 │ A   60 θ  │
         12 │ T   59 θ  ┘
         13 │ N   34 θ  ┐ C(R) to accumulator
         14 │ L    1 F  ┘
         15 │ G   24 θ    Test sign; jump if negative
         16 │ O   51 θ    Print space if positive
         17 │ E   27 θ                                   ┐
            ─────────────                                │
         18 ││ Δ      F                                  │
         19 ││ Y      F                                  │
         20 ││ θ      F                                  │ Table of
         21 ││ F      F                                  │ characters
         22 ││ J      F                                  │
         23 ││ φ      F                                  │
   15 →  24 │ T      D  ┐                                │
         25 │ O   52 θ  │ Change sign and print minus    │
         26 │ S      D  ┘                                ┘
   17 →  27 │ A   54πθ    Add round-off quantity
         28 │ T      D
         29 │ H   56 θ    10 · 2^-16 to multiplier register
         30 │ A   57 θ  ┐ Set digit layout constant
43,46 →  31 │ T    4 F  ┘
         32 │ V      D  ┐
         33 │ U      F  │ Convert and print digit
         34 │ A      F  ┘
```

```
35  |  A    53  θ  |
36  |  U    37  θ  |
37  | (O         F)|  Convert and print digit
38  |  L         F |
39  |  L      2  F |
40  |  T         D |
41  |  A      4  F |
42  |  L         D |
43  |  E    31  θ  |  Digit layout
44  |  O    51  θ  |
45  |  L         D |
46  |  G    31  θ  |
47  |  O    51  θ  |  Final space
48  | (E         F)|  Link order
    |--------------|
49  || W         F |  Carriage return
50  || I         F |  Line feed
51  || C         F |  Space
52  || M         F |  Minus sign
53  || O    17  θ  |  Base order
    || T    56  Z  |  (Round-off quantity is planted in 54πθ)
56  || P     5  F  |  $10 \cdot 2^{-16}$
57  || P         H |  Digit layout constant
58  || P         N |  P 5n F, where n is number of columns
59  ||(P         F)|  Column counter
60  || P     5  F  |  Block counter
```

P40 Print long signed integer with no layout.

Prints $2^{34} \cdot C(R)$.

```
        |  G          K  |
        |  O    40    K  |  Figure shift
        |  G          F  |
        |  T     6πZ     |  Plant more significant half of $-2^{33}/10^{10}$
        |  θ   524    D  |
        |  T          Z  |
     0  |  A     3    F  |  Plant link order
     1  |  T    32    θ  |
     2  |  N     6πθ     |  Multiply number by $2^{33}/10^{10}$
     3  |  G     8    θ  |  Test sign; jump if negative
     4  |  O    42    θ  |  Print space if positive
     5  |  E    11    θ  |
        |----------------|
     6  || P   610    D  |  $-2^{33}/10^{10}$
     7  || T     8    Z  |
3 →  8  |  O    43    θ  |  Print minus sign if negative
     9  |  T     4    F  |  Clear accumulator
    10  |  V     6πθ     |  Multiply number by $-2^{33}/10^{10}$
```

```
  5 → 11 | Y      F |
      12 | L      D |  Round off, shift and place in 0D
      13 | T      D |
      14 | H   44 θ |  10 · 2^-16 to multiplier register
      15 | A   45 θ |  Set digit layout constant
28,31 → 16 | T    4 F |
      17 | V      D |
      18 | U      F |
      19 | A      F |
      20 | A   40 θ |
      21 | U   22 θ |  Convert and punch digit
      22 | (O     F)|
      23 | L      F |
      24 | L    2 F |
      25 | T      D |
      26 | A    4 F |
      27 | L      D |
      28 | E   16 θ |  Digit layout
      29 | O   42 θ |
      30 | L      D |
      31 | G   16 θ |
      32 | (E     F)|  Link order
      ----------------
      33 || Δ     F |
      34 || Y     F |
      35 || θ     F |
      36 || F     F |
      37 || J     F |                  Table of characters
      38 || φ     F |
      39 || T     F |
      40 || O  32 θ |  Base order
      41 || S     F |
      42 || C     F |  Space
      43 || M     F |  Minus sign
      44 || P  5  F |  10 · 2^-16
      45 || P 1552 F|  Digit layout constant
```

P56 Print C(Acc), as signed fraction, with digit and page layout control by adjustable parameters within the subroutine.

The adjustable parameters are all *B*-orders within the subroutine. They are marked on the subroutine tape and are indicated below by asterisks.

```
                   |  G      K  |
Entry   [ →  0     |  O   31 θ  |  Space
points  [ →  1     |  O   31 θ  |  Space
        [ →  2     |  K   52 θ  |  Copy b for link order
        3 | (B      F) | Column count                   ┐
        4 |  J   14 θ  |                                │
        5 |  O   64 θ  | Carriage return                │
        6 | (B      F) | Line count                     │
        7 |  J   10 θ  |                                │
        8 |  O   65 θ  | Line feed                      │
      * 9 |  B    5 F  |                                │ Page layout
  7 →  10 |  BS   1 S  | ┐ Decrease line count          │
       11 |  K    6 θ  | ┘                              │
     * 12 |  B    5 F  |                                │
       13 |  O   65 θ  | Line feed                      │
  4 →  14 |  BS   1 S  | ┐ Decrease column count        │
       15 |  K    3 θ  | ┘                              ┘
       16 |  G   60 θ  | Jump if number is negative
       17 |  O   31 θ  | Space
 63 *→ 18 |  B   10 F  |
       19 |  K   47 θ  |
       20 |  H   57 θ  | —9/10
       21 |  T      D  |
       22 |  M   65 θ  |
 26 →  23 |  U    4 D  |
       24 |  V    4 D  |
       25 |  BS   1 S  |
       26 |  J   23 θ  |
     * 27 |  B    5 F  | ┐
       28 |  A      D  | │ Set to print five (or i) initial digits
       29 |  H   32 θ  | │
       30 |  F   34 θ  | ┘
          |------------|
       31 | ||C     F  | Space
       32 | ||P   5 F  | $10 \cdot 2^{-16}$
 51 →  33 | (B      F) | Count digits between spaces    ┐
 30 →  34 |  J   37 θ  |                                │ Print space
       35 |  O   31 θ  |                                │ every 5
     * 36 |  B    5 F  |                                │ (or every s)
 34 →  37 |  BS   1 S  | ┐ Decrease count of digits be- │ digits
       38 |  K   33 θ  | ┘   tween spaces               │
       39 |  T      D  | Plant fraction in 0D           ┘
       40 |  V      D  | Select digit, p, to be printed
       41 |  U      F  | ┐ Double, without shifting, so that $2^{-16}$ digit
       42 |  A      F  | ┘   is zero
       43 |  A   66 θ  | Add base order of table of characters
       44 |  U   50 θ  | Plant O(p + 53)θ in 50θ
       45 |  L      F  | ┐ Multiply by $2^{16}$
       46 |  L    2 F  | ┘
       47 | (B      F) | Digit count  ┐
       48 |  BS   1 S  |              │ Test for last digit
       49 |  K   47 θ  |              ┘
```

```
           50   (O         F)   Print digit      ] Test for last digit
           51    J     33  θ                     ]
           52   (B         F)   Link order
           53    E   1026  D                                          ]
           54    Δ         F                                          ]
           55    Y         F                                          ]
           56    θ         F                                          ] Table of
           57    F   1229  F                                          ] characters
           58    J         F                                          ] 0 to 9
           59    φ         F                                          ]
   16 →    60    T         D                     ] Change sign        ]
           61    O     22  θ    Minus sign       ] and print          ]
           62    S         D                     ] minus sign         ]
           63    F     18  θ
           64    W         F    Carriage return
           65    I         F    Line feed
           66    O     53  θ    Base order
                 O     40  K  ] Punch figure shift symbol during input of
                 G         F  ]   orders
                 T     67  Z    Reset Transfer Order
```

Q3 Quadrature using Gauss' six-point formula.

Computes $\int_{a-h}^{a+h} f(x)dx$ by the approximation

$$2h \sum_{i=1}^{3} d_i[f(a + b_i h) + f(a - b_i h)],$$

where d_i and b_i are constants. This is equivalent to fitting a curve of the eleventh degree.

$$a = C(0H),\ h = C(2H).$$

```
                 T          Z
            0    A      3   F   ] Plant link order
            1    T     30   θ   ]
            2    T      4   H     Clear 4H
            3    S     31   θ   ]
   24 →     4    A     32   θ   ] Plant orders
            5    U     16   θ   ]
            6    A     33   θ   ]
   22 →     7    T     10   θ
            8    A          H     +a
            9    H      2   H
           10   (V          F)    a ± b_i · h
           11    Y          F
           12    T          D     x to 0D
           13    A     13   θ   ] Call in auxiliary subroutine to calculate f(x)
           14    G          N   ]
```

```
Aux → 15  | H       D  |  f(x) to multiplier register
      16  | (V      F) |  d_i · f(x)
      17  | Y       F  |
      18  | A     4 H  |     3
      19  | T     4 H  |     Σ d_i[f(a + b_i h) + f(a − b_i h)] to 4H
      20  | A    10 θ  |  ]  i=1
      21  | A    34 θ  |  ]  Test if V or N in 10θ
      22  | G     7 θ  |  ]
      23  | S    35 θ  |
      24  | G     4 θ  |
      25  | H     4 H  |
      26  | V     2 H  |
      27  | L       D  |
      28  | Y       F  |
      29  | T       D  |
      30  | (E      F) |  Link order
          |------------|
      31  || P    6 F  |
      32  || V   42πθ  |
      33  || M    6 F  |
      34  || O      F  |
      35  || I   46πθ  |
          |  E   69 K  |  ]
          |  T   36πθ  |  ]  Call in R9 to read in following coefficients
    36πθ  |    14716  66184 F | d_1
    38πθ  |    30989  18315 F | d_2
    40πθ  |    40193  50093 F | d_3
    42πθ  |  1 60197  04270 F | b_1
    44πθ  |  1 13594  90762 F | b_2
    46πθ  |    40994  46400 π | b_3
          |  T   48 Z  |
```

*R*2 *Input of positive integers during input of orders.*

```
          | G     K   |
 9 →  0   | T 20  F   |                                        ]
      1   | V     D   | ] Multiply partial sum by ten (multiplier ]
      2   | L  8  F   | ]   register contains 10/32, set by initial ]
          |           | ]   input routine)                       ]
      3   | A 40  D   |   Add new digit                          ]
14 →  4*  | U     D   | ] New partial sum to 0D and ]            ]  Digit
      5   | (T    D)  | ]   to final destination of ]            ]  cycle
          |           | ]   number                  ]  Number    ]
      6   | I 40  F   | ]                           ]  cycle     ]
      7   | A 40  F   | ] Read next symbol          ]            ]
      8   | S 39  F   |   Subtract 11 · 2^-16       ]            ]
      9   | G     θ   |   Jump if symbol is not F or π ]         ]
```

* When obeyed for the first time in each number cycle this order clears 0*D*.

```
           10 │ S    2 F │ Subtract 2 · 2^-16                 ┐
           11 │ G   23 F │ Return to initial input rou-       │
              │          │   tine if symbol is π              │ Number
           12 │ A    5 θ │┐ Increase address of order in      │ cycle
Enter →    13 │ T    5 θ │┘   5θ by 2                         │
           14 │ E    4 θ │                                    ┘
              │──────────│
              │ E   13 Z │
              │----------│                     ┐ Enter subroutine with
              │ T   m  D │ Punched by user     ┘ T  m  D in accumulator
```

R29 Input of one positive integer, of any length up to ten decimal digits, punched in output code.

Note: The *B*-register must contain zero whenever this subroutine is called in.

```
               │ G  1025 K │ Place C(22) − 1 in 42F
       (θ − 1) │ F     2 θ │
               │───────────│
             0 ││W  1024 F │
             1 ││P     4 F │ (4)
(θ − 1) →    2 │ A     3 F │ (3)
             3 │ T    27 θ │      Plant link order
             4 │ L     6 F │ (6)  No effect
     20 →    5 │ T  1024 D │
     31 →    6 │ I    11 θ │      ┐ Read symbol
             7 │ A    11 θ │      ┘
             8 │ L     1 D │ (1)  ┐
             9 │ A    16 θ │      │ Form and plant H-order
            10 │ T    11 θ │      ┘
            11 │ (Z      F)│        Becomes H-order
            12 │ C       θ │        Test for digit
            13 │ F    24πθ │ (+)    Jump if not digit
            14 │ C    23 θ │ (Space)  Digit to accumulator
            15 │ R  1280 F │ (Erase)            ┐
            16 │ H       θ │ (Blank)            │ Multiply previous
            17 │ R       F │                    │ digit by 10 and
            18 │ V  1024 D │ (Carriage return)  │ add new digit
            19 │ L    16 F │ (0)                ┘
            20 │ F     5 θ │
               │───────────│
            21 ││P     7 F │ (7)
            22 ││P     2 F │ (2)
            23 ││P    15 F │
     13 →   24 │ E    29πθ │ (Line feed)
               │───────────│
            25 ││P     8 F │ (8)
            26 ││P     5 F │ (5)
     30 →   27 │ (E      F)│ Link
               │───────────│
            28 ││P     9 F │ (9)
     24 →   29 │ S  1024 D │
            30 │ G    27πθ │
            31 │ E     6πθ │
               │───────────│
```

*R*30 *Extension of the initial input routine to read code letter S.*

		G	πK		
$28F(S) \rightarrow$	0θ	S	F	Complement address	Test whether initial or final S
	1	F	$6\ \theta$	Becomes $F\ 6\ \pi\theta$	
	2	A	$2\ F$	Add $2^{-5} \cdot 2^{-18}$ to function digits of order, which are stored in less significant half of $40D$	
	3	R	$64\ F$		
	4	A	$40\ D$		
	5	F	$37\ F$		
$1 \rightarrow$	6	M	$41\ F$	Remove surplus digits of complement	
	7	F	$21\ F$		
$27F \rightarrow$	8	A	$43\ F$	These orders add 2^{-16} to an order containing code letter π	
	9	F	$8\ F$		
		T	$27\ K$		Modify orders of initial input routine
	$27F$	F	$7\pi\theta$	Becomes $F\ 8\ \theta$	
	$28F$	E	θ	Becomes $E\ \ \pi\theta$	

*R*37 *Input of one signed decimal fraction.*

		G	K	
		T	$28\pi Z$	Plant pseudo-orders, with sandwich digit zero, in $28\pi\theta$, $30\pi\theta$ and $32\pi\theta$ (i.e., in 29θ, 31θ and 33θ)
		$\| C$	$819\ F$	
		T	$30\pi Z$	
		$\| \theta$	$524\ D$	
		T	$32\pi Z$	
		$\| \Delta$	$1125\ F$	
		T	Z	
	0	T	D	Clear accumulator
	1	S	$32\pi\theta$	$8 \cdot 10^9 \cdot 2^{-34}$ to $4D$
	2	T	$4\ D$	
	3	F	$11\ \theta$	
$15 \rightarrow$	4	H	$4\ D$	Replace $C(4D)$ by $\frac{1}{10}\ C(4D)$
	5	N	$28\pi\theta$	
	6	Y	F	
	7	T	$4\ D$	
	8	V	$6\ F$	Multiply digit by $10^{10} \cdot 2^{-34}$ and accumulate fraction in $0D$
	9	L	F	
	10	A	D	
$3 \rightarrow$	11	T	D	
$17 \rightarrow$	12	I	$6\ F$	Read symbol and jump if decimal digit
	13	A	$6\ F$	
	14	S	$27\ \theta$	
	15	G	$4\pi\theta$	
	16	S	$27\ \theta$	Jump if symbol should be ignored
	17	G	$12\pi\theta$	
	18	S	$2\ F$	
	19	H	D	

```
        20 | G     25πθ  |  Jump if symbol is + or φ
        21 | V     30πθ  |  Multiply fraction (negative) by −2^33 · 10^−10
  26 →  22 | L        D  |
        23 | Y        F  |
        24 | FS     2 F  |  Link order
           |-------------|
  20 →  25 | N     30πθ  |  Multiply fraction (positive) by 2^33 · 10^−10
        26 | F     22 θ  |
           |-------------|
        27 ||P      5 F  |
        28 ||L   1229 F  |  ]  −1/10
           | T     30 Z  |
           | P    610 D  |  ]  −2^33 · 10^−10
           | T     32 Z  |
           | G        F  |  ]  −8 · 10^9 · 2^−34
           | T     34 Z  |
```

*S*3 *Cube root.*

Forms cube root of $C(6D)$ and places result in $0D$. The root is formed digit by digit, using a shifting (negative) strobe.

```
           |  G     K  |
        0  |  A   3 F  | ]  Plant link order
        1  |  T  20 θ  | ]
        2  |  T     D  |    Set first trial (i.e., zero)
        3  |  S  24 θ  | ]  Set test digit
  19 →  4  |  T   4 D  | ]
        5  |  H     D  | ]
        6  |  V     D  | ]
        7  |  Y     F  | ]  Form (trial)^3 − C(6D)
        8  |  T   8 D  | ]
        9  |  V   8 D  | ]
       10  |  S   6 D  | ]
       11  |  E  21 θ  |
       12  |  T   8 D  | ]
       13  |  S   4 D  | ]  Increase trial
  23 → 14  |  A     D  | ]
       15  |  T     D  | ]
       16  |  A   4 D  | ]
       17  |  R     D  | ]  Shift test digit
       18  |  Y     F  | ]
       19  |  G   4 θ  | ]
       20  | (E     F) |    Link order
           |-----------|
  11 → 21  |  T   8 D  | ]
       22  |  A   4 D  | ]  Decrease trial
       23  |  G  14 θ  | ]
           |-----------|
       24  | || I   F  |    = 1/2
```

*S*11 *Reciprocal square root of double-length number held in the accumulator.*

Forms $C(0D)/\sqrt{C(\text{Acc})}$, where $C(\text{Acc})$ is a double-length number held in the accumulator.

Repetitive process:

$$a_{n+1} = a_n - \tfrac{1}{2}a_n c_n, \quad a_0 = C(0D), \qquad a_n \to C(0D)/\sqrt{C(\text{Acc})},$$

$$c_{n+1} = c_n^2(\tfrac{1}{4}c_n - \tfrac{3}{4}), \quad c_0 = C(4D) - 1, \quad c_n \to 0.$$

```
              G      K
       0      K  33  θ     Plant link order
       1      B      F     Clear B-register
 2 →   2      G   2  θ     Dynamic stop if C(Acc) < 0
       3      S  35  θ     Jump if C(Acc) > ¼
       4      E   9  θ
 8 →   5      L   1  F
       6      A  36  θ     Shift left until > ¼
       7      BS  1  F
       8      G   5  θ
 4 →   9      S  36  θ     Plant c_0
      10      T   4  D
      11      A      D
      12      F  15  θ
15 →  13      L      D     Shift C(0D) into position; stop if overflow occurs.
      14      BS  1  S
12 →  15      J  13  θ
      16      φ      D
      17      A   4  D
32 →  18      U   4  D     c_n to multiplier register
      19      H   4  D
      20      R   1  F
      21      S  36  θ     (¼c_n − ¾) to 4D
      22      T   4  D
      23      N      D
      24      R      D
      25      A      D     a_{n+1} to 4D               Repetitive process
      26      Y      F
      27      φ      D
      28      V   4  D
      29      T   4  D     Form c_{n+1}
      30      V   4  D
      31      Y      F
      32      G  18  θ     Jump until c_{n+1} = 0
      33     (Z      F)
      34      ES  2  D     Link order
      35      R      F     ¼
      36      S      F     ¾
```

T7 Sine.

1/2 sin [2C(4D)] to 4D

```
        G        K
        E    69  K        Call in R9 to read coefficients
        T    26πθ
26πθ   11 453  246  086  F
28πθ    2 290  648  539  F
30πθ      218  152  390  F    Coefficients of power series
32πθ       12  105  378  F
34πθ           419  996  π
        T        Z
  0     A     3  F
  1     T    25  θ        Plant link order
  2     H     4  D
  3     V     4  D
  4     Y        F        Form [C(4D)]²
  5     T        D
  6     H        D
  7     N    34πθ
  8     A    32πθ
  9     T        D
 10     N        D
 11     A    30πθ
 12     T        D
 13     N        D
 14     A    28πθ
 15     T        D        Summation of power series
 16     N        D
 17     A    26πθ
 18     T        D
 19     N        D
 20     T        D
 21     H        D
 22     V     4  D
 23     A     4  D
 24     T     4  D
 25    (E        F)       Link order
        T    36  Z
```

*T*11 *Cosine.*

Replaces C(Acc) by $\frac{1}{2}\cos\pi[C(\mathrm{Acc})]$

```
            | Subroutine       |
            |    R2            |  Included on T11 tape
            | T          30πθ  |
   30πθ     | 11,453,246,123 F |  a3  ┐
   32πθ     |  2,290,649,225 F |  a5  │
   34πθ     |    218,157,069 F |  a7  │ Coefficients in power series
   36πθ     |     12,119,837 F |  a9  │
   38πθ     |        440,721 F |  a11 │
   40πθ     |         11,144 F |  a13 ┘
   42πθ     | 13,493,037,705 π |
            | T              Z |
        0   | K         27   θ |  Plant link order
        1   | E          4   θ | ┐
        2   | T              D | │ Take modulus of argument
        3   | S              D | ┘
   1 →  4   | S         29   θ | ┐
        5   | T              D | │ [|C(Acc)| − 1/2] to multiplier register
        6   | H              D | ┘
        7   | N         42πθ   |  Multiply by π/4
        8   | L              D | ┐ x, = π/2 [| C(Acc) | − 1/2], to 4D
        9   | T          4   D | ┘
       10   | H          4   D | ┐
       11   | V          4   D | │ x^2 to 0D
       12   | Y              F | │
       13   | T              D | ┘
       14   | H              D |
       15   | N         40πθ   |
       16   | B         10   F | ┐
  21 → 17   | BS         2   S | │
       18   | AS        30πθ   | │ Cycle for power series; forms −a3x^2 +
       19   | T              D | │   a5x^4 − ··· + a13x^12
       20   | N              D | │
       21   | J         17   θ | ┘
       22   | T              D | ┐
       23   | H              D | │
       24   | V          4   D | │ x − a3x^3 + a5x^5 − ··· + a13x^13
       25   | Y              F | │
       26   | A          4   D | ┘
       27   | (B             F)| ┐ Link order
       28   | FS         2   F | ┘
            |------------------|
       29   | || I           F |  1/2
            | T         44   Z |
```

*Z*5 *Post-mortem of orders*

```
PZ Z K    M2037F G K P 10 K P Z T 43 K P θ
   Z K    M2037F G K P 10 K P Z T 43 K P π 0 θ
   Z K    M2037F G K P 10 K P Z T 44 K P π 0 θ
Space
P Z P 800  F ZK M2037F GK P10K PZT 43K  P θ
             ZK M2037F GK P10K PZT 43K  P π 0 θ
                                 T 45K  P π 0 θ
                       I 43 K P F Q E 25 K T φ
```

"Dialing" subroutines, based on *M*20, to allow destination of *Z*5 to be dialed into 0φ(44*F*) (unless *Z*5 is to start in 800*F*—see specification), and start of post-mortem to be dialed into 0*H*.

```
                 Order                    Notes

              G         K
  51 →  0     O      18 θ     Print space
  53 →  1     (H        H)    Select order to be printed
        2     A       1 θ  ┐
        3     A       2 F  │  Increase address of order in 1 θ
        4     T       1 θ  ┘
Start → 5     (H     39 θ)    Becomes count
        6     J      14 θ     B-register contains zero at Start
        7     O      64 θ     Carriage return
  11 →  8     O       9 θ     Line feed
        9     I         F     Read character ("erase" first time)
       10     S         F  ┐
       11     G       8πθ  ┘  Test character read; jump if erase
       12     (B   1015 F)
       13     B    1034 F
   6 → 14     B    2047 F     b = 0 first time
       15     K      12 θ
       16     K       5 θ
       17     O      31 θ     Letter shift
       18     C      65 θ     C(R) to accumulator
       19     A      66 θ     Add —1 (to convert function digits to output
                                code)
       20     U         F
       21     O         F     Print function letter
  52 → 22     L       8 F     Shift 5 places left
       23     E      26 θ     Jump if B-digit was 0
       24     A      66 θ     Add —1 to remove sign digit by deliberate
                                overflow
       25     O       2 θ     Print S
  23 → 26     H      60 θ     32/100 to multiplier register
       27     B       2 F  ┐
       28     K       4 F  ┘  B 2 F to 4F
```

```
        29 | T        D  | Address 2^-10 to 0D
        30 | O     11 θ  | Figure shift
  48 → 31 | V        D  ]
        32 | R    512 F  ] Select most significant decimal digit of address
        33 | U        F  ]
        34 | A        F  ]
        35 | S      2 F  ] Jump to suppress initial zeros
        36 | G   1063 θ  ]
        37 | A     61 θ  ] Convert digit of address
        38 | U     39 θ  ]
        39 | (Δ       H) | Becomes 0-order to print digit
        40 | B   1022 F  | Clear B-register to end zero suppression
  36 → 41 | L        F  ]
        42 | L      2 F  ] Shift next digit of address into position
        43 | T        D  ]
        44 | A      4 F  |                                            ] Count
        45 | S     31 θ  |                                            ] digits of
        46 | H     67 θ  | 10 · 2^-16 · 2^11 to multiplier register   ] address
        47 | U      4 F  |                                            ] (not more
        48 | G    31πθ  | Cycle                                      ] than three)
        49 | A        F  | +
        50 | A     49 θ  | −1/4
        51 | G      πθ  | Jump if special indication digit not present
        52 | O     22 θ  | Print*
        53 | E     1πθ  | Cycle to select next order   0 ]
        ---------------
        54 | Δ        F  |                               1 ]
        55 | Y        F  |                               2 ]
        56 | θ        F  |                               3 ]
        57 | F        F  |                               4 ]
        58 | J        F  |                               5 ] Table of
        59 | φ        F  |                               6 ] characters
        60 | T    246 F  | 32/100                        7 ]
        61 | O     54 θ  | Base order                    8 ]
        62 | S        F  |                               9 ]
        63 | U        F  |                              10 ]
        64 | W        F  |
        65 | V   2047 D  | This number has a 1 in every digital position
        66 | K   4096 F  | −1
        67 | T        F  |
           | E      5 Z  ] Enter Z5 at order 5
           | P        F  ]
```

* At this point the special indication digit is in the 2^{-2} position in the accumulator.

APPENDICES

APPENDIX 1

INPUT AND OUTPUT CODES OF THE EDSAC

INPUT					OUTPUT		
On input tape	Decimal equivalent (in EDSAC)		Labels on keyboard		On output tape and in EDSAC	Symbols printed by teleprinter	
			Fig.	Letter		(Fig. Shift)	(Letter Shift)
10 · 000	0		0	*P*	00 · 000	No effect	
10 · 001	1		1	*Q*	00 · 001	/	*F*
10 · 010	2		2	*W*	00 · 010	Carriage return	
10 · 011	3		3	*E*	00 · 011	0	*D*
10 · 100	4		4	*R*	00 · 100	·	·
10 · 101	5		5	*T*	00 · 101	7	*H*
10 · 110	6		6	*Y*	00 · 110	2	*N*
10 · 111	7		7	*U*	00 · 111	10	*M*
11 · 000	8		8	*I*	01 · 000	Line feed	
11 · 001	9		9	*O*	01 · 001	8	*L*
11 · 010	10			*J*	01 · 010	5	*X*
11 · 011	11			π	01 · 011	′	*G*
11 · 100	12			*S*	01 · 100	9	*A*
11 · 101	13			*Z*	01 · 101	?	*B*
11 · 110	14			*K*	01 · 110	:	*C*
11 · 111	15		Erase		01 · 111	=	*V*
00 · 000	−16	16	(Blank tape)		10 · 000	(	*P*
00 · 001	−15	17		*F*	10 · 001	4	*Q*
00 · 010	−14	18		θ	10 · 010	3	*W*
00 · 011	−13	19		*D*	10 · 011	)	*E*
00 · 100	−12	20		ϕ	10 · 100	6	*R*
00 · 101	−11	21	+	*H*	10 · 101	|	*T*
00 · 110	−10	22	−	*N*	10 · 110	,	*Y*
00 · 111	−9	23		*M*	10 · 111	—	*U*
01 · 000	−8	24		Δ	11 · 000	1	*I*
01 · 001	−7	25		*L*	11 · 001	*	*O*
01 · 010	−6	26		*X*	11 · 010	Σ	*J*
01 · 011	−5	27		*G*	11 · 011	Figure shift	

(continued)

INPUT					OUTPUT		
On input tape	Decimal equivalent (in EDSAC)		Labels on keyboard		On output tape and in EDSAC	Symbols printed by teleprinter	
			Fig.	Letter		(Fig. Shift)	(Letter Shift)
01 · 100	—4	28		*A*	11 · 100		*S*
01 · 101	—3	29		*B*	11 · 101	+	*Z*
01 · 110	—2	30		*C*	11 · 110	Space	*K*
01 · 111	—1	31		*V*	11 · 111	Letter shift	

In the first and fifth columns above, the full stop represents the sprocket hole, which is always punched. The first column gives the codes corresponding to the keys labelled with the function letters in the fourth column. Some of these keys are also labelled with decimal digits or a plus or minus sign, as shown in the third column, so that, for example, letter *T* is synonymous with figure 5. The remaining keys are also labelled with punctuation or mathematical symbols, as well as with letters, but these are not shown in the third column since they are irrelevant as far as the EDSAC is concerned. In the second column the decimal equivalent of each Input character is given, a scale factor of a suitable power of 2 being assumed. In the lower part of this column two alternative decimal equivalents are shown. The negative equivalent applies when the character occurs as the five most significant digits in an order. The extreme left-hand digit is then a 1 and, for numerical purposes, acts as a sign digit, thus indicating a negative number.

APPENDIX 2

ORDER CODE AND CONTROLS OF THE EDSAC

Orders which are shown below with address n can refer to either short- or long-storage locations, that is, they can be terminated by either F or D. n must lie in the range $0 \leq n \leq 1023$ and, if the order refers to a long-storage location, n must be even. Orders shown with address m do not refer to long-storage locations and use the "special indication" digit for other purposes. They must therefore be terminated in such a way that the special indication digit is zero unless otherwise specified below.

(1) *Arithmetical and logical orders.*

A n Add the content of location n into the accumulator.

S n Subtract the content of location n from the accumulator.

ϕ n Transfer to location n the more significant part of the content of the accumulator, clear the accumulator and then, if the capacity of the accumulator has been exceeded since the accumulator was last cleared, stop the machine, light the "accumulator overflow" lamp and give audible warning.

U n Copy the more significant part of the content of the accumulator into location n, but retain the whole content of the accumulator.

T n Transfer the more significant part of the content of the accumulator to location n and clear the accumulator.

H n Replace the content of the multiplier register by that of location n.

V n Multiply the content of location n by that of the multiplier register, and add the product into the accumulator.

N n Multiply the content of location n by that of the multiplier register, and subtract the product from the accumulator.

Y m Round off the number in the accumulator to 34 binary digits; that is, add 2^{-35} to the accumulator. (The address in this order is usually zero but it may be mF where $0 \leq m \leq 1023$. It must *not* be mD. (Cf. order Y m D.)

C n Collate the content of location n with that of the multiplier register; that is, add a 1 into the accumulator in those (and only those) digital positions in which both $C(n)$ and $C(R)$ have a 1.

M n Clear the six most significant digits of the accumulator and then add the content of location n into the accumulator.

(2) *Shift orders.*

In the following orders the address digits (including the special indication digit) are used to specify the amount of shift.

$R \quad D$ Shift the content of the accumulator one place to the right; that is, divide it by 2.

$R \ \frac{1}{4} \cdot 2^p \ F$ Shift C(Acc) p places to the right ($2 \leq p \leq 11$); that is, divide it by 2^p.

$R \quad F$ Shift C(Acc) 15 places to the right; that is, divide it by 2^{15}.

$L \quad D$ Shift C(Acc) one place to the left; that is, multiply it by 2.

$L \ \frac{1}{4} \cdot 2^p \ F$ Shift C(Acc) p places to the left ($2 \leq p \leq 11$); that is, multiply it by 2^p.

$L \quad F$ Shift C(Acc) 13 places to the left; that is, multiply it by 2^{13}.

(3) *Jump orders.*

In these orders the address m always refers to a short-storage location. If the special indication digit is present it modifies the function of the order, as is shown in each individual case.

$F \ m \ F$ Take $C(m)$ as the next order to be obeyed.

$F \ m \ D$ If C(Acc) is not zero, take $C(m)$ as the next order; otherwise proceed serially.

$E \ m \ F$ If C(Acc) is positive or zero, take $C(m)$ as the next order; otherwise proceed serially.

$E \ m \ D$ If C(Acc) is positive or zero, clear the accumulator, and take $C(m)$ as the next order; otherwise proceed serially.

$G \ m \ F$ If C(Acc) is negative, take $C(m)$ as the next order; otherwise proceed serially.

$G \ m \ D$ If C(Acc) is negative, clear the accumulator, and take $C(m)$ as the next order; otherwise proceed serially.

$Y \ m \ D$ If the capacity of the accumulator has been exceeded since the last $Y \ m \ D$ order was encountered, clear the accumulator and take $C(m)$ as the next order; otherwise proceed serially. (Cf. order $Y \ m$.)

$J \ m \ F$ See below.

(4) *Orders which concern the B-register.*

$B \ n$ Replace the content of the B-register by the number n (N.B.: *not* by the content of location n).

$J\ m\ F$	If the content of the B-register is *not* zero, take $C(m)$ as the next order; otherwise proceed serially.
$K\ m\ F$	Place the order $B\ b\ F$ (where b is the content of the B-register) in storage location m. (Note that this does *not* clear the B-register; the register can be cleared, when necessary, by the order $B\ \ F$).

(5) *Input and output orders.*

$I\ n\ F$	Place $x \cdot 2^{-16}$ in storage location n, where x is the integer represented in input code by the row of holes on the input tape, and advance the tape by one row of holes.
$O\ m\ F$	Punch on the output tape a row of holes corresponding to the five most significant digits (including the sign digit) in storage location m; a hole corresponds to a 1 and a blank to a 0.

(6) *Orders which stop the machine.*

$Z\ m\ F$	Stop the machine and light the "program stop" light on the control panel.
$Z\ m\ D$	(a) If the $Z\ D$ switch on the control panel is on, this order operates exactly like $Z\ m\ F$ above. (b) If the $Z\ D$ switch is off, this order has no effect, and the machine passes straight on to obey the next order.

Notes: 1. $Z\ m\ D$ acts as a conditional stop order and is often of use in diagnosing mistakes in programs.

2. The address, m, in a Z-order is usually zero.

3. If the machine has stopped on a Z-order in location p, say, it may be restarted at the order in $p + 1$ by pressing the Reset button or operating the telephone dial.

(7) *Manual controls.*

The manual controls used in normal operation of the EDSAC are as follows:

(a) The Clear Store button: this clears every storage location, and is always pressed before starting a program.

(b) The Start button: this places the orders of the initial input routine in storage locations 0 through 40, and then begins to obey them, starting at the order in 0.

(c) The Manual Stop button: this stops the machine, which may later be started again by pressing the Reset button.

(d) The Reset button: this causes the machine to resume working from the point it had reached in the program when it was stopped by a Z-order, by a ϕ-order if the capacity of the accumulator was exceeded, or by the operator pressing the Stop button.

(e) The telephone dial: this is used for the occasional insertion of small amounts of numerical data. If any decimal digit n is dialed ($1 \leq n \leq 10$—note that the dial position labelled 0 is equivalent to ten, and not to zero) the number $n \cdot 2^{-15}$ will be added into the accumulator as the dial returns to rest. As soon as the rest position is reached the machine will be reset, just as if the Reset button were pressed. The dial is frequently used in conjunction with the control combination $Z\ K$—see Appendix 4, and subroutine $M20$ in Part 2 and Part 3.

(f) The $Z\ D$ switch: this causes the machine to stop when an order $Z\ m\ D$ is encountered. If the switch is off, the order is ignored (see Appendix 6).

Other manual controls exist (notably one which causes the machine, when stopped, to obey one order of the program, each time the button is pressed) but are rarely used except for investigating malfunctions of the machine.

APPENDIX 3

THE INITIAL INPUT ROUTINE OF THE EDSAC

Location	*Order*	*Notes*
0	(T F)	These orders cause control to be transferred to 20. They are not used after the start, but their locations are used as working space.
1	(E 20 F)	
2	‖ P 1 F	These are constants which are intended to be left here unaltered in any program.
3	‖ U 2 F	
12 → 4	A 39 F	Input of address. This group of orders is entered at 8 with the accumulator empty, so that 0 is cleared. The next digit on the tape is read and tested to see if it is less than eleven; if so it is doubled and added to ten times the content of 0, the sum being sent back to 0. The next digit is read, tested, etc., and this is continued until the whole address has been formed; the next digit read, x, is greater than ten and so corresponds to a code letter.
5	R 4 F	
6	V F	
28] 7	L 8 F	
38] → 8	T F	
9	I 1 F	
10	A 1 F	
11	S 39 F	
12	G 4 F	
13	L D	These test to see if x is greater than sixteen. If it is, the order $A(24 + x)F$ is formed and planted in 20. If x is sixteen or less a jump order $E(16 + x)F$ is formed and planted in 20.
14	S 39 F	
15	E 17 F	
16	S 7 F	
15 → 17	A 35 F	
18	T 20 F	
19	A F	This adds the address, which is always positive, into the accumulator.
20	(H 8 F)	This order places 10/32 in the multiplier register during the start and is later replaced by a manufactured one which either adds to the accumulator the number determined by x, or transfers control to an address determined by x.
21	A 40 F	This adds in the function digits of the order, so the accumulator now contains the order from the tape plus the number selected by x.

Location	*Order*	*Notes*
22	(*T* 43 *F*)	This (the Transfer Order) transfers the assembled order to its final place in the store.
23	*A* 22 *F*	These orders increase the address specified in the Transfer Order by unity.
24	*A* 2 *F*	
31 → 25	*T* 22 *F*	
26	*E* 34 *F*	Transfers control to 34.
T 20 → 27	*A* 43 *F*	Control is transferred to these orders by the order in 20 when π has been read from the tape. They add 2^{-16} to the address (which is in the accumulator) and transfer control to 8. The address now refers to a long storage location.
28	*E* 8 *F*	
Z 20 → 29	*A* 42 *F*	This adds the address in 42 to the accumulator.
K 20 → 30	*A* 40 *F*	This adds the function digits of the order to the accumulator. The result is that the number in the accumulator is positive if the order has function digits represented by *T* or *E*, while it is negative in the case of *G*.
31	*E* 25 *F*	If the content of the accumulator is positive, the order in the accumulator replaces the order in 22; if negative, the accumulator contains the address specified in order 22 which is then put in 42 (the storage location corresponding to θ).
blank 20 → 32	*A* 22 *F*	
33	*T* 42 *F*	
26 → 34	*I* 40 *D*	These read the function digits, shift them to their correct place, and transfer them to 40. The order in 35 is also used as a constant.
35	*A* 40 *D*	
36	*R* 16 *F*	
37	*T* 40 *D*	
38	*E* 8 *F*	
39	*P* 5 *D*	A constant used in the input of the address. It equals $11 \cdot 2^{-16}$.
40	(*P* *D*)	A constant used during the start. It equals 2^{-16}.

When the Start button is pressed, the initial input routine is placed in storage locations 0–40, and control transferred to 0. The first orders to be executed are the following:

Location	*Order*	*Notes*
0	$T \quad F$	Clear accumulator
1	E 20 F	Transfer control to 20
20	H 8 F	Place 10/32 in multiplier register
21	A 40 F	Add 2^{-16} to accumulator
22	T 43 F	Transfer 2^{-16} to 43 (the storage location corresponding to π)
23	A 22 F	Increase order 22 to T 44 F
24	A 2 F	
25	T 22 F	

The initial input routine is now ready to take in orders; the first part of the input tape is blank so that the first code letter is a space which corresponds to 16; control is therefore switched from 20 to 32, and the content of 22 is transferred to 42. This action will continue, the spaces being treated alternately as function digits and code letters. The first symbols encountered will be P and Z. There are two possibilities, either

(1) the last space was treated as a function digit, in which case the word read is "space" Z, which causes address $(n - 1)$ to be placed in 42, where n is the address in the Transfer Order; or

(2) the last space was treated as a code letter, in which case the word read is PZ, which causes the address in 42 to be placed in the Transfer Order.

In either case, the Transfer Order is unaltered and will place the first order read from the tape in 44, unless a control combination to reset the Transfer Order occurs first, as will usually be the case.

APPENDIX 4

CONTROL COMBINATIONS

$E\ m\ K\ P\ F$		Transfers control to the order in storage location m, leaving the accumulator clear.
$E\ Z\ P\ F$		Transfers control to the first order of the last subroutine to be read (i.e., to the order in storage location 0θ), leaving the accumulator clear.
$E\ m\ Z\ P\ F$		Transfers control to the mth order of the last subroutine to be read (i.e., to the order in $m\theta$), leaving the accumulator clear.
$E\ m\ K$ $E\ \ Z$ $E\ m\ Z$	Followed by any positive* order	These three control combinations transfer control to the order in storage location mF, 0θ, or $m\theta$, respectively, with the positive order following the control combination left in the accumulator.
$E\ 25\ K$	Followed by any positive* order	Transfers control to order 25 of the initial input routine. This then causes the Transfer Order (in $22F$) to be replaced by the positive order following the control combination.
$G\ K$		Places a reference address, equal to the current address in the Transfer Order, in storage location 42 (corresponding to code letter θ).
$G\ m\ K$		Places a reference address, equal to m plus the current address in the Transfer Order, in storage location 42.
$T\ m\ K$		Causes the next order (or pseudo-order) read from the tape to be placed in storage location m.
$T\ \ Z$		This causes the address (p, say) held in storage location 42 to replace that in the Transfer Order, so that the next order read from the tape will be placed in p.
$T\ m\ Z$		This causes the address in the Transfer Order to be replaced by $(m + p)$, where p is the address held in 42.
$T\ m\ \pi K$ $T\ m\ \pi Z$		These cause the Transfer Order to be replaced by $T\ m\ D$ or $T\ m + p\ D$, respectively. The next order (or pseudo-order) read from the tape will therefore be placed in the more significant half (the odd-numbered half) of the long-

storage location m or $m + p$, the least significant half, including the sandwich digit, being cleared. If the next item to be read is $P\ F$, the whole long-storage location will be cleared.

$Z\ K\ P\ F$ This causes the Transfer Order to be replaced by $Z\ F$, and leaves the accumulator clear. The machine then stops when it obeys the Transfer Order. The machine will resume reading the tape when the Reset button is pressed, but the next item read from the tape must be a control combination to replace the Transfer Order, otherwise the machine will stop again when the $Z\ F$ is encountered.

$Z\ \pi\ K\ P\ F$ This behaves exactly as $Z\ K\ P\ F$ provided that the $Z\ D$ switch is on. If the switch is off, the control combination has no effect provided that the next item read from the tape is a control combination to reset the Transfer Order.

$O\ 40\ K\ \Sigma\ F$ This causes the symbol corresponding, in output code, to the binary equivalent of the character Σ to be punched on the output tape. This takes place during the input of the program and does not use up any storage space. The next item to be read must be a control combination to reset the Transfer Order.

$P\ Z$ This resets the Transfer Order after a section of blank tape has been read. In older literature on the EDSAC, including the first edition of this book, the control combination $P\ K$ was sometimes used, but $P\ Z$ is more general and is now preferred.

* By "any positive order" is meant any order or pseudo-order whose numerical representation in the machine is positive. In general, this means that the function letter on the tape must be positive, but there may be exceptions. For example, if the H-parameter is $P(n + 1)F$, a pseudo-order punched as $V\ 2047\ H$ will appear in the machine as $P\ n\ F$.

APPENDIX 5

SPECIMEN SOLUTIONS TO PROGRAMMING EXERCISES

There are many possible solutions to each example, and the solutions given here are not necessarily those which take the smallest number of orders or the shortest amount of machine time. They are chosen for their straightforwardness and to illustrate typical programming techniques. It is assumed that the first order of each program is in location 400, and that the constants are stored in locations 500 onward. The column headed "operand" gives the quantity referred to by the order.

Exercises A (page 9)

	Location	*Order*	*Operand*
1.	400	*A* 20	x
	1	*A* 22	y
	2	*T* 30	$x + y$
	3	*A* 20	x
	4	*A* 20	
	5	*S* 22	y
	6	*T* 32	$2x - y$
2.	400	*H* 0	x
	1	*V* 0	x
	2	*T* 0	x^2
3.	400	*H* 0	x
	1	*V* 0	x
	2	*T* 0	x^2
	3	*V* 0	x^2
	4	*T* 0	x^3
4.	400	*H* 4	
	1	*V* 10	$1/\pi^2$
	2	*T* 4	
5.	400	*H* 6	y
	1	*V* 6	
	2	*U* 8	y^2
	3	*H* 4	x
	4	*V* 4	
	5	*U* 0	$x^2 + y^2$

	Location	*Order*	*Operand*
	6	*S* 8	y^2
	7	*S* 8	
	8	*T* 8	$x^2 - y^2$
	9	*V* 6	y
	10	*V* 6	
	1	*T* 10	$2xy$
6.	400	*H* 10	x
	1	*V* 10	
	2	*T* 0	x^2
	3	*V* 0	
	4	*U* 0	x^3
	5	*V* 0	
	6	*T* 0	$x^4 + x^3$
7.	400	*H* 10	a
	1	*V* 11	b
	2	*H* 12	c
	3	*V* 13	d
	4	*H* 14	e
	5	*V* 15	f
	6	*T* 16	$ab + cd + ef$
8.	400	*H* 60	x
	1	*V* 100	a
	2	*A* 102	b
	3	*T* 4	$ax + b$
	4	*V* 4	
	5	*A* 104	c
	6	*T* 4	$ax^2 + bx + c$

9. Call the dimensions a, b and c.

	Location	*Order*	*Operand*
	400	*H* 50	a
	1	*V* 51	b
	2	*U* 101	ab
	3	*V* 52	c
	4	*H* 52	c
	5	*V* 51	b
	6	*U* 100	$ab + ac + bc$
	7	*A* 100	
	8	*T* 100	$2(ab + ac + bc)$
	9	*V* 101	ab
	410	*T* 101	abc

Exercises B (*page* 11)

```
               Location      Order              Notes

1.                  400      A      4
                      1      G    404        Jump to 404 if C(4) < 0
                      2      T      0     ]
                                          ]  Change sign
                      3      S      0     ]
              401 → 404      T      0        Transfer |C(4)| to 0

2.                  400      A      4     ]
                                          ]  Form C(4) − C(6)
                      1      S      6     ]
                      2      E    405        Test sign
                      3      T      0     ]
                                          ]  Change sign if negative
                      4      S      0     ]
              402 → 405      T      0

3.                  400      A      4     ]
                      1      E    404     ]
                                          ]  Form |C(4)|
                      2      T      4     ]
                      3      S      4     ]
              401 → 404      S      0
                      5      G    408π       Test if |C(4)| ≥ C(0)
                      6      A      0
                      7      T      0
              405 → 408

4.                  400      H      4     ]
                                          ]  Form C(4) · C(6)
                      1      V      6     ]
                      2      E    409        Test sign
                      3      T      0        Clear accumulator
                      4      A      4     ]
                      5      S      6     ]
                      6      E    409     ]  Form |C(4) − C(6)|
                      7      T      0     ]
                      8      S      0     ]
          402]
              → 409          T      0
          406]

8.                  400      A    500
                      1      S     50
                      2      T     50
                    500   || Z      0        This pseudo-order represents the
                                               constant 13/16 = 5/8 + 3/16
```

Exercises C (page 18)

```
             Location        Order          Notes

1.                400        B   100        Set b = 100
           403 → 401        BS     1 S  ]
                    2        TS  100     ]  This order clears locations 199, 198,
                    3        J   401    ]     . . . , 100 in turn

2.                400        T      4       Clear 4
                    1        B    100
           409 → 402        BS     1 S
                    3        AS   100   ]
                    4        E    407   ]
                    5        SS   100   ]   Form |C(100 + b)|
                    6        SS   100   ]
           404 → 407        A      4    ]   Accumulate sum of moduli
                    8        T      4    ]
                    9        J    402

3.                400        B    100
           404 → 401        BS     1 S  ]
                    2        HS   100   ]   Accumulate sum of squares
                    3        VS   100   ]
                    4        J    401   ]
                    5        T      4
```

6. Assume $C(500) = 1/10$.

```
                  400        B     50
           406 → 401        BS     1 S
                    2        T      0       Clear accumulator
                    3        SS   200
                    4        A    500       1/10
                    5        G    408π      Test if C(200 + b) ≤ 1/10
                    6        J    401
                    7        F    414
                           ──────────
           405 → 408        H    500       1/10
                    9        B     50
           413 → 410        BS     1 S  ]
                    1        VS   200   ]   Multiply numbers by 1/10
                    2        TS   200   ]
                    3        J    410   ]
           407 → 414        Next order
```

8. (i) Taking shortest possible machine time:

```
        400   H   10      x
          1   V   10
          2   T    0      x^2
          3   H    0      x^2
          4   V    0
          5   T    0      x^4
          6   H    0      x^4
          7   V   10
          8   T    0      x^5
          9   V    0
         10   T    0      x^9
          1   V    0
          2   T    0      x^13
```

Several other equally short solutions are possible.

(ii) Using the fewest possible orders:

```
        400   H   10      x
          1   B   12
 401 -> 402   BS   1 S
          3   V   10
          4   U   10    ]
          5   T    0    ]  Store x^n in 0 and 10
          6   J  402
```

Exercises D (*page* 24)

```
        Location       Order            Notes
1.        400        A   500        T 0
            1        A     4
            2        T   403
          403       (Z     0)       Becomes T n
          500      || T    0

2.        400        A   500        B 0
            1        A     4
            2        T   403
          403       (Z     0)       Becomes B n
            4        SS  300    ]
            5        G   407π   ]  Replace C(300 + n) by |C(300 + n)|
            6        TS  300    ]
   405 -> 407       Next order
          500      || B    0
```

```
             Location          Order                  Notes

3.                400       A   500
                    1       A     4
                    2       T   403
                  403      (Z     0)          Becomes B n + 1
                    4       T     6           Clear 6
                    5       H     0           x
           410 → 406        B     1  S  ┐
                    7       V     6     │
                    9       AS  150     │     Sum power series
                    9       T     6     │
                   10       J   406     ┘
                  500     ‖ B     1
```

Exercises E (page 33)

We give the solutions below as typical examples.

```
1.         410 → 400        H   500
                    1       A   501
                    2       B   402     ┐
                    3       F   160     ┘     Call in D
                          ─────────────
             D → 404        B   404     ┐
                    5       F   250     ┘     Call in P
                          ─────────────
             P → 406        A   500           n/16
                    7       A   501           1/16
                    8       U   500           (n + 1)/16
                    9       S   502           11/16
                   10       G   400π
                  500     ‖ W     0           2/16
                    1     ‖ Q     0           1/16
                    2     ‖ π     0           11/16
```

Or, if terminating symbols F and θ are used, this becomes

```
                            G        K
             10 →      0θ   H   500  F   ┐
                        1   A   501  F   │
                        2   B     2  θ   │    Master routine
                        3   F   160  F   ┘
                          ────────────────
```

```
         Location       Order           Notes
     D →      4      B     4 θ  ]
              5      F   250 F  ]
     P →      6      A   500 F  ]
              7      A   501 F  ]  Master routine
              8      U   500 F  ]
              9      S   502 F  ]
             10      G     π θ  ]
            500   || W       F  ]
              1   || Q       F  ]  Constants
              2   || π       F  ]

                     G       K
3.   400 =   0θ      A   500 F  ]  Set n = 2 initially                    ]
              1      T   604 F  ]                                         ]
              2      T   602 D     Clear sum                              ]
              3      A   501 F     Pick up initial value 1/n! = ½         ]
              4      F    12 θ                                            ]
    20 →      5      A   503 F                                            ]
              6      T   604 F                                            ]
              7      A   600 D                ]                    1      ]
              8      R     4 F                ]  Divide     ------------  ]
              9      H   604 F                ]             16(n - 1)!    ]
             10      B    10 θ  ]             ]                           ]
             11      F   160 F  ]  Call in D  ]                by n/16    ]
 4 ]
 D ]  →      12      U   600 D                                            ]  Master routine
             13      B    13 θ  ]             ]                           ]
             14      F   250 F  ]  Call in P  ] Print 1/n!                ]
     P →     15      A   600 D                ] Add 1/n! to pre-          ]
             16      A   602 D                ] vious sum to re-          ]
             17      T   602 D                ] place it                  ]
             18      A   604 F                ]                           ]
             19      S   502 F                ] Test whether              ]
             20      G     5 θ                ] n = 10; if not, re-       ]
             21      A   602 D                ] place by n + 1; if        ]
             22      B    22 θ  ]             ] so, print sum and         ]
             23      F   250 F  ]  Call in P  ] stop.                     ]
     P →     24      Z       F                ]                           ]
            500   || W       F     2/16 = initial value of n/16
              1   || I       F     1/2  = initial value of 1/n!
              2   || J       F     10/16
              3   || π       F     11/16
         Location        Order             Notes
                         G       K
14.   400 =     0θ       B       θ    ] Call in R to read in constant
                1        F   200 F    ]   .05 (5 mins.)
       R →      2        T   500 F
      21 →      3        Z       F      Stop for insertion of data tape
                4        T   600 D      Clear current (600 F) and
                5        F    14 θ        maximum (601 F) waiting
                                          times
      16 →      6        A   600 F      Add previous waiting time
                7        S   500 F      Subtract 5 mins. to obtain
                                          present waiting time
                8        G     9πθ      If negative, clear and replace
       8 →      9        U   600 F
               10        S   601 F      Subtract previous max. wait-
               11        G    14πθ        ing time. If result negative,
               12        A   601 F        clear and jump; if positive,
               13        T   601 F        replace previous max. by
                                          present waiting time
      11 →     14        B    14 θ    ] Call in R to read in time occu-
               15        F   200 F    ]   pied by next patient
       R →     16        E     6 θ      Jump if positive number read
               17        T       F      If negative number read, clear
               18        A   601 F        accumulator
               19        B    19 θ    ] Call in P and print out maxi-
               20        F   250 F    ]   mum waiting time
       P →     21        F     3 θ      Return to program stop for
                                          new data
```

Master routine

Exercises F (*page* 40)

The three examples below use the typical master routines given in the specimen solutions to programming Exercises E above.

1. and 3.

```
 P      Z
 T 400  K
[Master routine]
 Blank tape
 P      Z
 T 500  K
[  Constants  ]
 Z      K
 P      F
```

```
            Blank tape
            P     Z
            E 400 K
            P     F

7.          P     Z
            T 400 K
          Master routine
            Blank tape
            P     Z
            T 500 K
            Constants
            Z     K
            P     F
            Blank tape
            P     Z
            E 400 K
            P     F
          ....................
            Data tape

14.         P     Z
            T 400 K
          Master routine
            Z     K
            P     F
            Blank tape
            P     Z
            E 400 K
            P     F
               05+
          ....................
            Data tape or tapes
```

BIBLIOGRAPHY

A. *Machines and programming*

Adams, C. W., Small Problems on Large Computers. *Proc. Assn. Computing Machinery, Pittsburgh*, p. 99, (1952).

Aiken, H. A., and Hopper, G. M., The Automatic Sequence Controlled Calculator. *Electrical Engineering*, v. 65, pp. 384, 449, 522 (1946). [An abridged version of the *Manual of operation*.] See Harvard University.

Alt, F. L., A Bell Telephone Laboratories' Computing Machine. *M.T.A.C.* v. 3, pp. 1, 69 (1948).

Auerbach, A., The Elecom 100 General Purpose Computer. *Proc. Assn. Computing Machinery, Pittsburgh*, p. 47, (1952).

Backus, J. W., The I.B.M. 701 Speedcoding System, *J. Assn. Computing Machinery*, v. 1, p. 4 (1954). [An interpretive scheme.]

Booth, A. D., and Booth, K. H. V., Automatic Digital Calculators. *Butterworth's Scientific Publications, London.* (Second edition, 1956) [Mainly devoted to the design of digital computers, but contains some discussion of programming.]

Bennett, J. M., Prinz, D. G., and Woods, M. L., Interpretive Subroutines. *Proc. Assn. Computing Machinery, Toronto*, p. 81 (1952) [Includes an example of the use of an interpretive subroutine for making one digital computer simulate another.]

Brooker, R. A., and Wheeler, D. J., Floating operations on the EDSAC. *M.T.A.C.*, v. 7, p. 37 (1953).

Brown, J. H., Programming for the MAGIC I System. *University of Michigan* (report) (1955). [A description of a conversion routine developed under the direction of John W. Carr.]

Carr, J. W., Report of an informal discussion on programming methods. *Review of Electronic Digital Computers—Joint AIEE-IRE Conference, A.I.E.E., New York*, p. 113 (1952). [An early discussion of the methods described in Chapter 8 of this book, including references to floating addresses under the name "free" addresses.]

Carr, J. W., Progress of the Whirlwind Computer towards an Automatic Programming Procedure. *Proc. Assn. Computing Machinery, Pittsburgh*, p. 237 (1952).

Denman, H. H., Kopley, E. S., and Porter, J. D., Comprehensive System Manual. *Digital Computer Laboratory, Massachusetts Institute of Technology* (report) (1953) [Full description of a comprehensive conversion routine developed under the direction of Charles W. Adams.]

Diehm, I. C., Computer Aids to Code Checking. *Proc. Assn. Computing Machinery, Toronto*, p. 19 (1952). [The diagnosis of errors in programs.]

Eckert, W. J., and Jones, R. B., Faster, Faster. *McGraw-Hill Book Co., New York* (1955). [A description of the NORC.]

Estrin, G., The electronic computer at the Institute for Advanced Study. *M.T.A.C.* v. 7, p. 108 (1953).

Freedman, A. L., Elimination of waiting time in automatic computers with delay-type stores. *Proc. Camb. Phil. Soc.*, v. 50, p. 426 (1954). [A discussion of minimum-access coding, and other methods of reducing waiting time, written from the point of view of the design of a digital computer using a magnetic drum for storage.]

Gill, S. A process for the step-by-step integration of differential equations in an automatic digital computing machine. *Proc. Camb. Phil. Soc.*, v. 47, p. 96 (1951). [The Runge-Kutta-Gill method.]

Gill, S. The diagnosis of mistakes in programmes on the EDSAC. *Proc. Roy. Soc.* A., v. 206, p. 538 (1951).

Goldstine, H. H. and Goldstine Adele, The ENIAC. *M.T.A.C.*, v. 2, p. 97 (1946). [Contains a general description of the ENIAC and gives examples of how programs are plugged.]

Goldstine, H. H., and Von Neumann, J., Planning and Coding of Problems for an Electronic Computing Instrument. *Institute for Advanced Study, Princeton* (report), Part II, Vol. I (1947); Part II, Vol. II (1948).

Gordon, B., An Optimizing Program for the IBM 650. *J. Assn. Computing Machinery*, v. 3, p. 3 (1956).

Gorman, T. P., Kelly, R. G., and Reddy, R. B., Automatic Coding for the IBM 701. *J. Assn. Computing Machinery*, v. 2, p. 253 (1955).

Gotlieb, C. C., Running a Computer Efficiently. *J. Assn. Computing Machinery*, v. 1, p. 124 (1954).

Greenwald, S., Haueter, R. C., and Alexander, S. N., SEAC. *Proc. I.R.E.*, v. 41, p. 1300 (1953).

Harvard University Computation Laboratory, A Manual of Operation for the Automatic Sequence Controlled Calculator. *Annals of the Computation Laboratory, Harvard University*, v. 1 (1946).

Harvard University Computation Laboratory, Description of a Relay calculator. *Annals of the Computation Laboratory, Harvard University*, v. 14 (1949). [The Harvard computer Mk II.]

Harvard University Computation Laboratory, Description of a Magnetic Drum Calculator. *Annals of the Computation Laboratory, Harvard University*, v. 25 (1952). [The Harvard computer Mk III.]

Hopper, G. M., The Education of a Computer. *Proc. Assn. Computing Machinery, Pittsburgh*, p. 243 (1952). [Compiling routines.]

Hume, J. N. P., and Worsley, Beatrice H., Transcode: A System of Automatic Coding for FERUT. *J. Assn. Computing Machinery*, v. 2, p. 243 (1955).

Huskey, H. D., Characteristics of the Institute for Numerical Analysis Computer. *M.T.A.C.*, v. 4, p. 103 (1950). [The SWAC.]

Leiner, A. L., System Specifications for the DYSEAC. *J. Assn. Computing Machinery*, v. 1, p. 57 (1954).

LEVIN, J. H., Construction and Use of Subroutines for the SEAC. *Proc. Assn. Computing Machinery, Pittsburgh,* p. 173 (1952).

LIPKIS, R., The Use of Subroutines on SWAC. *Proc. Assn. Computing Machinery, Pittsburgh,* p. 231 (1952).

NATIONAL BUREAU OF STANDARDS, The Operating Characteristics of the SEAC. *M.T.A.C.,* v. 4, p. 229 (1950).

OFFICE OF NAVAL RESEARCH, A survey of automatic digital computers. *Office of Naval Research, Washington, D. C.,* (1953). [Includes a brief specification of all digital computers known to be operating, or under construction, at the time of compilation.]

PERKINS, R., EASIAC, A Pseudo-Computer. *J. Assn. Computing Machinery,* v. 3, p. 65 (1956).

RIDGWAY, R. K., Compiling Routines. *Proc. Assn. Computing Machinery, Toronto,* p. 1. (1952).

RUTISHAUSER, H., Some Programming Techniques for the ERMETH. *J. Assn. Computing Machinery,* v. 2, p. 1 (1955).

WEIK, M. H., A Survey of Domestic Electronic Digital Computing Systems. *Ballistic Research Laboratories, Aberdeen, Md.,* (report No. 971) (1955). [Includes brief specifications and photographs of all machines known to be operating, or likely to be available, in North America.]

WHEELER, D. J., Programme organisation and initial orders for the EDSAC. *Proc. Roy. Soc.* A., v. 202, p. 573, (1950).

WILKES, M. V., Pure and Applied Programming. *Proc. Assn. Computing Machinery, Toronto,* p. 121, (1952).

WILKES, M. V., Automatic Digital Computers. *Methuen* (1956).

WILKES, M. V., The use of a "Floating Address" system for orders in an automatic digital computer. *Proc. Camb. Phil. Soc.,* v. 49, p. 84 (1953).

WILKES, M. V., and RENWICK, W., The EDSAC—an Electronic Calculating Machine. *J. Sci. Inst.,* v. 26, p. 385 (1949). (Logical design.]

WILKES, M. V., and RENWICK, W., The EDSAC (Electronic Delay Storage Automatic Calculator). *M.T.A.C.,* v. 4, p. 61 (1950). [Logical design.]

WILKINSON, J. H., An 'Assessment of the System of Optimum Coding used on the Pilot ACE at the N.P.L. *Phil. Trans. Roy. Soc. Lond.,* A. v. 248, p. 253 (1955).

B. *Numerical Analysis*

HARTREE, D. R., Numerical Analysis. *Oxford Univ. Press* (2nd Edition, 1955).

HASTINGS, C. H., Approximations for Digital Computers. *Princeton University Press* (1955).

HILDEBRAND, F. B., Introduction to Numerical Analysis. *McGraw-Hill* (1956).

HOUSEHOLDER, A. S., Principles of Numerical Analysis, *McGraw-Hill* (1953).

HOUSEHOLDER. A. S., Bibliography on numerical analysis. *J. Assn. Computing Machinery,* v. 3, p. 88, (1956). [A list of 321 references intended to supplement and bring up to date the bibliography in the same author's "Principles of Numerical Analysis."]

Jones, C. W., Miller, J. C. P., Conn, J. F. C., and Pankhurst, R. C., Tables of Chebyshev Polynomials. *Proc. Roy. Soc. Edin.*, A, v. LXII, pt. II, p. 187 (1946).

Lanczos, C., Trigonometrical Interpolation of Empirical and Analytical Functions. *Journ. Math. Phys.*, v. XVII, p. 123 (1938). [Use of Chebyshev polynomials].

Milne, W. E., Numerical Analysis. *University of Princeton Press* (1949).

Milne-Thomson, L. M., The Calculus of Finite Differences. *Macmillan,* (1933).

National Bureau of Standards, Tables of Chebyshev Polynomials $S_n(x)$ and $C_n(x)$. *Applied Mathematics Series,* 9 (1952).

Whittaker, E. T. and Robinson, G., The Calculus of Observations. *Blackie* (1957).

INDEX

DAVID WHEELER

M. V. WILKES

STANLEY GILL

THE AUTHORS

M. V. Wilkes took the Mathematical Tripos at Cambridge University in 1934, and later did graduate work on radio wave propagation at the Cavendish Laboratory. During the war he was engaged in radar and operational research, and when the war was over was appointed Director of the University Mathematical Laboratory. He has visited the United States on a number of occasions and is well known there in the computer field.

David Wheeler entered the field of automatic computing in 1948. During the years 1948–51 he did graduate work in programming and numerical analysis in the Mathematical Laboratory at Cambridge. He obtained his Ph.D. degree in 1951 and a research fellowship at Trinity College later in the same year.

During the years 1951–53 he was Visiting Assistant Professor at the University of Illinois. He returned to England in 1953 and now holds a staff appointment in the Mathematical Laboratory.

Stanley Gill entered the electronic computer field in 1947 while at the National Physical Laboratory, where he joined the design team planning the Pilot Model A.C.E. Later he became a programmer at Cambridge, where he obtained his Ph.D. degree in 1953 for research into methods of applying the EDSAC to problems in mathematics and physics. This work included the introduction of mistake diagnostic routines and the development of a particular form of the Runge-Kutta process for the solution of differential equations.

Dr. Gill spent 18 months in the United States during 1953–54, where he succeeded Dr. Wheeler in the position of Visiting Assistant Professor at the University of Illinois; he also lectured at Summer Session courses at the Massachusetts Institute of Technology. He is now head of the Computing Research Group of Ferranti Ltd., in London.

PRINTED IN U.S.A.

OTHER A-W BOOKS OF INTEREST

ADVANCED CALCULUS

By Wilfred Kaplan, *University of Michigan*

From a review in QUARTERLY OF APPLIED MATHEMATICS: 679 pp, 241 illus (1952)

"An excellent addition to the list of books on this subject, because of its laudable emphasis on vector methods in all phases of the calculus in which they can be applied. The material contained covers all the usual subjects found in a course of advanced calculus, but the emphasis seems definitely slanted toward the advanced engineer or the applied mathematician."

INTRODUCTION TO MEASURE AND INTEGRATION

By M. E. Munroe, *University of Illinois*

From a review in AMERICAN MATHEMATICAL MONTHLY: 320 pp, 14 illus (1953)

"This book has been written primarily as a textbook for a first course in measure and integration theory. It presupposes that the reader has a basic knowledge of real function theory . . . The author's style is clear and concise. Motivation is provided and possible misunderstandings pointed out. It is an excellent textbook and in addition should prove to be a good reference for the non-specialist in integration."

THEORY OF MATRICES

By Sam Perlis, *Purdue University*

From a review in AMERICAN MATHEMATICAL MONTHLY: 237 pp, 5 illus (1952)

"This is a finely written, excellent introduction to matrices. The style is natural and clear. Aimed at intelligent but immature students, a complete but elementary treatise on basic ideas was the author's purpose. In this he has succeeded."

TOPICS IN NUMBER THEORY

By William J. LeVeque, *University of Michigan*

From a review in SCIENCE:

Vol. I, 198 pp, 6 illus (1956); Vol. II, 267 pp, 7 illus (1956)

"The first volume is eminently suitable for a graduate or advanced undergraduate course in the theory of numbers. It is also equally well suited to introduce a reader with an adequate background to the subject. The second volume gives a stimulating selection of more advanced topics with an unmistakable preference for analytic, rather than algebraic and combinatorial, methods."

AN INTRODUCTION TO RIEMANN SURFACES

By George Springer, *University of Kansas* 307 pp, 78 illus (1957)

A modern, self-contained treatment of the fundamental concepts and basic theorems concerning Riemann surfaces. While it is designed primarily for use as a textbook in mathematics courses on the advanced undergraduate-graduate level, physicists and engineers will also find it a valuable introduction to the field.

ADDISON-WESLEY PUBLISHING COMPANY, INC.
Reading, Massachusetts, U.S.A.