PROGRAMME DESIGN FOR THE C.S.I.R.O. MARK I COMPUTER

I. COMPUTER CONVENTIONS

By T. PEARCEY* and G. W. HILL*

[*Manuscript received December* 5, 1952]

Summary

The organization of the C.S.I.R.O. Mark I computer is described. The code system is of a two-address type in which each operation called by an " order " or " command " is thought of as a quality of a transfer of the content of one " register " to another. Each command indicates a source and a destination. A special notation of mnemonic type is adopted for written programmes as an aid in the various stages of programme design. The written programme may be translated, by aid of a special encoding key-punch, from the written sheet onto paper tape, the medium through which the machine accepts data.

I. INTRODUCTION

The basic principles of programme construction for automatic computers have been known for some time, at least since the attempt by Charles Babbage to construct an analytical engine. These principles apply to modern automatic computers, and only the details differ from machine to machine. It is one of the aims of the machine designer to make the programming procedures as simple as possible for users.

Two technical developments, the use of electronic techniques and of rapid access " variables stores ", have made fully automatic computers possible in recent years. The introduction of electronic techniques has increased the computing rate by at least a thousandfold beyond the speeds attainable by electromechanical systems using punched cards and relay equipment. This has allowed of great simplification in the organization of modern computers, the main being that sufficient speed can be maintained by performing at one time only *one* of a small class of different actions, such as addition, subtraction, reading new data, recording results, and transferring numbers and so on.

Automatic computers, particularly those operating at electronic speeds, require an extended set of single steps, known as a " programme ", which controls the course of a calculation. Thus, each step, called a " command ", must be expressed as one of the restricted class of actions which the computer is capable of performing.

Until recently automatic computers have been provided with programmes from punched cards or paper tape, each command being read from the card or tape and performed immediately, whereupon the computer proceeds to the next

* Division of Radiophysics, C.S.I.R.O., University Grounds, Sydney.

card or position of the tape. The development of rapid access " stores " or blocks of number registers of large capacity allows the entire programme to be supplied to the computer in the form of sequences of numbers, and entered into the same store that is used for holding the incidental computing data etc. This makes it possible to subject the programme itself to changes as the computer progresses through the calculations.

The facility for a computer to change its current programme greatly reduces the number of commands required and the effort needed to construct the programme and code it into a suitable medium for transfer into the computer. Part of any programme, constructed according to this principle, must be devoted to controlling the changes to be made to itself, so that useful computing speed is somewhat reduced. The extra flexibility obtained far outweighs the disadvantages.

The problems of design and construction of modern computers have been discussed more frequently and at much greater length than have the problems of their programming and use. Only in the case of one particular computer, the EDSAC (Wheeler 1950 ; Gill 1951 ; Wilkes, Wheeler, and Gill 1951), has much been published on the latter subject.

The programming techniques developed for use in the EDSAC bear close resemblance to those developed for the present machine. This is so particularly in the use of basic programme elements or " sub-routines " which are designed for frequently repeated sets of operations, and in some of the techniques adopted for simplifying their method of use and incorporation into complete programmes.

Although the techniques used for EDSAC render programming work remarkably simple, no detailed knowledge of the machine being needed, it is believed that in some ways the method developed here is made even easier, and the demand on store space even smaller, by the incorporation into the machine of certain special features.

The terminology adopted here closely resembles that used by the Cambridge Mathematical Laboratory. Terms not well known or not immediately obvious will be defined when first introduced.

The initial aim of the programmer is of course to obtain the accuracy he needs in the results required, and this may decide for him the particular methods by which the computation is made. The programme will usually be compiled so as to use as little store space as possible, at the same time maintaining an adequate rate of output of results. These are very broad principles and subject to wide variations. Thus, a programmer designing a basic component or sub-routine will tend to occupy the least possible store space, almost irrespective of its speed, whilst a programmer designing a programme for a full-scale calculation using sub-routines is likely first to ensure that his demands do not exceed the store capacity and then choose the method requiring least machine time.

Detailed programming techniques adopted for any particular machine depend largely upon the " address system ", that is, the manner in which the machine interprets a command as a meaningful operation. Any programme implies a sequence of such operations, and must be written down in a manner easily read and understood. A programme initially designed on paper must

be transcribed onto some medium which is directly accepted by the machine, usually punched paper tape or punched cards, although magnetic tapes and wires are in use by certain machines. In our case the machine accepts paper tape although punched cards are used as an auxiliary medium. It is essential for simplicity in the last stage of programming, the stage which may be called " coding ", that the written programme be coded directly into the medium without further writing or extra mental effort.

In the present paper the organization of the machine is described only so far as is necessary for the understanding of the programming method, and this is followed by a description of the address system. The basic actions of the machine are listed in detail in Tables 2 and 3, together with a notation used in the construction of written programmes, which, although not ideal, appears to be suitable. The manner of coding from the written sheet is described, but a description of details of the construction of complete programmes will be given in Part II (Pearcey and Hill 1953).

For recording written programmes it is convenient that the notation symbols be available on a typewriter.

II. The Structure of the Computer

The C.S.I.R.O. Mark I computer is of the electronic serial-transfer type, and operates entirely in the binary scale. The programmer need know very little about the details of the machine but a general picture of its structure and its mode of operation is useful.

The stores, those components which hold the programme of commands and the data operated upon, are of two kinds. First, a main or high speed store (access time 1 msec), capable of storing 1024 separate data or " words " or basic groups of digits, is used always to hold the programme currently in use and incidental data as desired. This store consists of a group of mercury-filled acoustic delay lines each holding 16 words placed end to end. The register positions or " locations " in the store are numbered serially from 0000 to 1023.

The second store, of slow access (10 msec), is divided into four groups each of a capacity of 1024 separate numbers and is of the magnetic drum type. The high speed store is of the " volatile " type, that is, the numerical content is destroyed whenever the power is removed ; in the magnetic drum type this is not so and information is retained.

The arithmetical unit consists of a group of acoustic delay registers known as registers A, B, C, H, and D. The main arithmetical register is A, and has a capacity of one word only, as also have B and C. Register H, however, stores only half the digit capacity of a full number, whilst D can store 16 words end to end.

Other registers, not part of the arithmetical equipment, are the " input register " from which data fed to the computer may be read into the stores or the arithmetical components ; the " output register " via which results may be recorded ; and " constant registers ", two of which have full digit capacity for a complete word and may be adjusted manually by sets of switches. Three others provide single unit digits in prescribed places.

The operation of the computer is controlled according to the numerical content of three registers. The " sequence register " specifies from which location in the high speed store a command is to be withdrawn for use. The " interpreter register " receives and holds the command in use. The " store control register " indicates which location in one or other of the stores is prepared for immediate use.

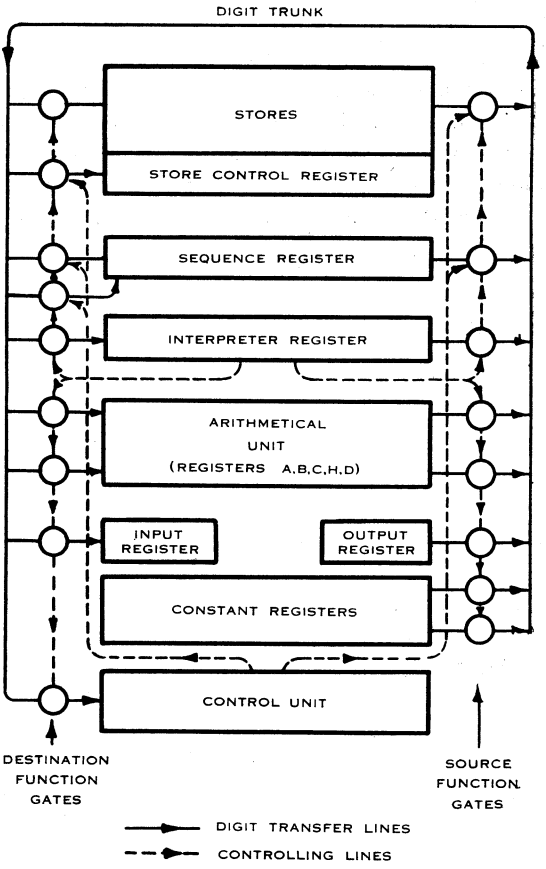

Fig. 1.—Block schematic diagram of the computer.

Figure 1 shows a block diagram of the major components of the computer and indicates the main interconnections. All the aforementioned registers are connected in various ways to a conductor known as the " digit trunk ", and it is along this conductor that the numbers are transferred from one register to another in the form of trains of suitably timed electrical pulses.

III. The Number and Command Code

A single location in the stores contains a single word or datum consisting of 20 binary digits, the distribution of which represents a single number, command, or partitioned datum according to the manner in which the programme uses the word. The positions or periods of the digits of a 20-digit word are denoted by

symbols, p_1, p_2, . . ., p_{20}, and are written in the time sequence at which the digits appear on the digit trunk. Each digit held within any one register may possess a weight of either unity or zero, according as at the appropriate digit instant an electrical pulse respectively does or does not appear in the digit trunk. Single digits are frequently referred to by their corresponding period symbol, that is, a unit digit at the instant p_r is called a p_r digit, and the time of appearance of such a digit is the p_r period.

(*a*) *Numbers*

In the case of numbers a weighting is applied to the digits which relates one digit to that which immediately follows in the time sequence. Thus, a unit p_r digit is given the additional weighing of 2^{r-20}. Negative numbers are stored in their complementary form and the " binary point " is considered as lying between the digit periods of p_{19} and p_{20}. Hence a single number can take values equal to integral multiples of 2^{-19} from -1 to $1-2^{-19}$.

Such a convention considers numbers as fractions, but by suitable programme design it is possible to compute with numbers as though the binary point were between any pair of adjacent digits.

(*b*) *Partitioned Data*

Not all data held in the store are modulo two numbers. Some 20-digit data may be used by a programme according to any desired convention of partition. Thus a word may, in the course of a programme, be divided up into a number of separate components either related to one another or not, e.g. into the groups p_1–p_{10} and p_{11}–p_{20}. Examples of partitioned data are the two components of a complex number, numbers which bear " tags ", that is, special digits describing the meaning of the accompanying main group of digits, and numbers in floating form which possess variable indices.

All commands are a form of partitioned data.

(*c*) *Commands*

A single command is coded as a sequence of 20 binary digits. The address convention of partitioning adopted, however, affects the entire design of the computer and the method by which programmes are designed.

An operation defined by any one command is regarded as the transfer of the digital content of one register to a second register. Any arithmetical operation which takes place during a transfer depends on the manner in which the first or " source " register transmits, and the second or " destination " register receives. In this sense the command code system is of the two-address type.

The digits of any command are partitioned into three groups ; the first those in periods p_1–p_5, the second in p_6–p_{10}, and the third in p_{11}–p_{20} inclusive. The first group contains the code for the destination and the second the code for the source. The third, the largest group, is a " sub-address " and indicates one of the 1024 serially numbered locations in one or other of the stores if required. This last group may also contain any desired group of 10 binary

digits even if neither the source nor the destination calls upon the stores. The command code allows for a total of 32 sources and 32 destinations and stores of up to 1024 locations. It is believed that the C.S.I.R.O. Mark I is the only computer to use an address system of this type.

In cases of commands which refer to a location in the D register the digits in p_{11}–p_{14} indicate which location is called for use irrespective of the digits in p_{15}–p_{20}.

The address groups and the digit periods are illustrated in Table 1.

TABLE 1

THE ADDRESS SYSTEM

Digit periods	$p_{20}\ p_{19}\ p_{18}$ ..... p_{11}	$p_{10}\ p_9\ p_8\ p_7\ p_6$	$p_5\ p_4\ p_3\ p_2\ p_1$
Address groups	←Store sub-address→	←—Source—→	← Destination →
Mod. 2 number weightings	Sign 2^{-1} ..... 2^{-9}	2^{-10}.....	..... 2^{-19}
Typical command $c(A)\rightarrow 16$	0 0 0 0 0 1 0 0 0 0	0 1 0 0 1	0 0 0 0 0
The number 4/5	0 1 1 0 0 1 1 0 0 1	1 0 0 1 1	0 0 1 1 0

IV. FUNCTION GATES

To every source code number there exists a " source function gate " which becomes activated and able to transmit a datum to the digit trunk, whenever the current command contains the appropriate number in the source group. Similarly every destination code number corresponds to a " destination function gate " which is activated according to the destination address of the current command and allows a datum to pass through it from the digit trunk. These are indicated in Figure 1.

V. THE OPERATION SEQUENCE

Commands to be performed in succession are normally stored in and withdrawn from successively numbered locations in the high speed store. For any one command to be satisfied four transfers must take place in the following sequence :

(1) The content of the sequence register is transmitted to the store control register, thus preparing the store to transmit out the next command.

(2) The store transmits the contents of the position indicated by the content of the store control register, i.e. the new command, to the interpreter register.

(3) The sub-address group (p_{11}–p_{20} digits) of the current command is transmitted back from the interpreter register to the store control register, being substituted in place of the previous contents. This prepares the store for action if either of the addresses requires it.

(4) The appropriate source and destination function gates indicated by the addresses of the command are activated by suitable circuits connected to the interpreter register, thus allowing the desired transfer to take place.

This sequence of transfers, called the " computer routine " is invariable and is controlled by the " main control unit " shown in Figure 1, which provides the necessary electrical waveforms.

The content of the sequence register at stage 1 of the computer routine is known as the " control number ". The term " control at n " implies that the current command is selected from location n. Any change in the value of n is " shift of control ". Normally control shifts from n to $n+1$ during a single cycle of the computer routine. A single unit p_{11} is added to the control number in the sequence register during stage 3 of each computer routine cycle.

VI. The Source and Destination Codes

The following tables list the operations for which the various function gates are responsible. Table 2 lists the sources and Table 3 the destinations in the numerical order of the code from 0 to 31.

TABLE 2

SOURCE FUNCTION GATES

Code Number	Function	Symbol
0	Read out and hold the content of the high speed store location n ($0 \leqslant n \leqslant 1023$), indicated by the sub-address digits p_{11}–p_{20}	(n)
1	Read out the content of the input register (20 digits, p_1–p_{20}) ..	(I)
2	Read out the content of hand-set register No. 1 (20 digits, p_1–p_{20})	(N_1)
3	Read out the content of hand-set register No. 2 (20 digits, p_1–p_{20})	(N_2)
4	Read out the content of register A (20 digits, p_1–p_{20})	(A)
5	Read out the most significant digit of the content of register A (unit p_{20} if 1, zero if 0)	$p_{20}.(A)$ or $s.(A)$
6	Read out the content of register A divided by 2 (20 digits)	$\frac{1}{2}(A)$
7	Read out the content of register A multiplied by 2 (20 digits)	$2(A)$
8	Read out the least significant digit of the content of register A (unit p_1 if 1, zero if 0)	$p_1.(A)$ or $l.(A)$
9	Read out the content of register A and leave it cleared to zero	$c(A)$
10	If the content of register A is non-zero transmit a unit p_1 digit, otherwise transmit zero	$\bar{z}(A)$
11	Read out the content of register B	(B)
12	Read out a unit p_1 digit if the most significant digit of the content of register B is unity, otherwise transmit zero	(R)
13	Read out the content of register B shifted to the right one place	$r(B)$
14	Read out the content of register C	(C)
15	Read out the most significant digit of the content of register C (unit p_{20} if 1, zero if 0)	$p_{20}.(C)$ or $s.(C)$
16	Read out the content of register C shifted one place to the right (zero in p_{20} position)	$r(C)$
17	Read out the content of the location in register D indicated by the number m ($0 \leqslant m < 16$), represented by the digits p_{11}–p_{14}	(D_m)
18	Read out the most significant digit in the location in register D indicated by the number m ($0 \leqslant m < 16$), represented by the digits p_{11}–p_{14}	$p_{20}.(D_m)$ or $s.(D_m)$

TABLE 2 (*Continued*)

SOURCE FUNCTION GATES

Code Number	Function	Symbol
19	Read out the content of the location in register D indicated by the number m ($0 \leqslant m < 16$), represented by the digits p_{11}–p_{14}, shifted one place to the right	$r(D_m)$
20	Read out zero	(Z)
21	Read out the 10-digit content of register H in the position group p_1–p_{10}	(H_l)
22	Read out the 10-digit content of register H in the position group p_{11}–p_{20}	(H_u)
23	Read out the 10-digit content of the sequence register in the position group p_{11}–p_{20}	(S)
24	Read out a single unit digit in the p_{11} position	p_{11}
25	Read out a single unit digit in the p_1 position	p_1
26	Read out from the interpreter register the content of the digits of the current command held in positions p_{11}–p_{20}	(K)
27	Read out the content of the low speed store No. 1 from the location n ($0 \leqslant n < 1024$), indicated by the 10-digit sub-address of the current command	(n_1)
28	Read out the content of the low speed store No. 2 from the location n ($0 \leqslant n < 1024$), indicated by the 10-digit sub-address of the current command	(n_2)
29	Read out the content of the low speed store No. 3 from the location n ($0 \leqslant n < 1024$), indicated by the 10-digit sub-address of the current command	(n_3)
30	Read out the content of the low speed store No. 4 from the location n ($0 \leqslant n < 1024$), indicated by the 10-digit sub-address of the current command	(n_4)
31	Read out a unit digit in the position p_{20}	p_{20}

Most of the entries in Tables 2 and 3 are self-explanatory. The following notes will assist in defining more clearly the action of some of the functions.

Not all registers are associated with both source and destination gates; for instance, the output register possesses only destination gates, whilst the various constants have only source gates.

The output from any source gate may be transferred to any destination gate which may be consistently coded. Thus, to transfer the content of location n in the high speed store to location m ($n \neq m$) in the high speed store is an invalid transfer and cannot be coded, except in the case $n=m$ when the transfer is possible. Similarly a transfer from a store location $16r+p$ ($0 \leqslant p < 16$) to location q ($0 \leqslant q < 16$) of register D is possible only if $p=q$. Similar remarks apply to the low speed store.

The input register possesses 20-digit hand-setting facilities like those of N_1 and N_2 and the entire 20 digits are transferred if called as a source.

The destination I_t affects the mode of reading data from the medium, in this case punched paper tape, from which data are placed into the input register. In the " read binary " condition the row of digits represented by the configuration of holes or the current column of the tape is accepted directly into the register. In the " decimal read " state the configuration of digits is subjected to some transformation upon entering the input register. The effect in this state is that of interpreting tape punched with one hole per row in a 1 to 10 code for decimal digits and transferring each digit to the input register as the equivalent binary tetrad. This is more fully described later.

In the right-shift sources of registers B, C, and D, i.e. $r(B)$, $r(C)$, and $r(D)$, the lowest digit (at p_1) is omitted from transfer ; a unit digit in the p_r period is transmitted as a unit p_{r-1} and the digit transmitted in the p_{20} position is zero irrespective of the digit in the p_{20} period held in the register.

In the case of multiplication the modulo 2 convention is adopted and the product of the content of register C and the number entering register B, both stored modulo 2, is added into the combined registers A and B as though the binary point of the product lies between the p_{20} and p_{19} periods in register A. The original multiplier is lost in the process of multiplication and is replaced by the less significant 19 digits of the product. The multiplicand is retained in register C. In cases of the multiplier or multiplicand being negative (i.e. possessing unit p_{20} digits) suitable corrections are made to the product to provide a negative product in complementary form.

A loudspeaker has been found useful for providing audible signals which help the operator to follow the course of a calculation. It is selected by destination number 10.

VII. Notation

A special notation is adopted for writing programmes. Each command is represented by at least two symbols, one for the source and one for the destination. The symbols are those indicated in Tables 2 and 3. Usually these symbols involve a letter denoting the register involved and an accompanying symbol to indicate the particular function of the function gate selected.

The source symbol is written to the left of a rightward pointing arrow with a lower half-head only, i.e. $\rightharpoondown$, to the right of which the destination symbol is written. To aid the eye the functional component of the destination symbol is moved to a point over the arrow. This symbol does not exist on standard typewriters, but this is not a great inconvenience since programmes are finally typed on standard forms which have the arrow printed upon them.

The content of a register is implied by surrounding the letter representing the register referred to by round brackets, thus (A) means the numerical content of register A and is the symbol shown in Table 2. The bracket notation is used in indicating some source functions. In cases of the remaining sources the brackets are not necessary as, for instance, when a constant digit or set of digits is transmitted, e.g. sources of p_1, p_{11}, and p_{20} units. Further, if the source is the interpreter register, it is convenient to write the actual number transferred out in place of the source symbol omitting brackets, the interpreter register being

TABLE 3

DESTINATION FUNCTION GATES

Code Number	Function	Symbol
0	Read and substitute into the high speed store in location n ($0 \leqslant n \leqslant 1023$), indicated by the digits p_{11}–p_{20} of the current command	n
1	If at least one unit received, change the mode of input reading from decimal code to straight binary or vice versa, otherwise no change	I_t
2	Read into the output register the digits received in positions p_{11}–p_{15} and print the corresponding character	O_t
3	Read into the output register the digits received in positions p_1–p_5 and punch onto tape	O_p
4	Substitute into the register A (20 digits)	A
5	Add into the content of register A and hold the sum ..	$+A$
6	Subtract from the content of register A and hold the difference	$-A$
7	Replace the content of register A by the digit by digit product of its content and the entering digits (e.g. conjunction)	$\cdot A$
8	Replace the content of register A by the digit by digit logical sum of its content and the entering digits (e.g. disjunction)	$\vee A$
9	Replace the content of register A by a unit digit wherever the content differs digit by digit from the digits entering	$\sim A$
10	Transfer the entering digit train into the loudspeaker ..	P
11	Substitute into register B	B
12	Read into register B, form the product of the content of B and register C (modulo 2) and add into the content of register A, retaining the lowest 19 digits of the product in B with a zero in the p_1 position of B	$\times B$
13	Only if a unit p_{20} is received shift the content of registers A and B one place to the left, the digit in p_{20} position of B shifting to the p_1 position of A	L
14	Substitute into register C	C
15	Add into the content of register C and hold the sum	$+C$
16	Subtract from the content of register C and hold the difference ..	$-C$
17	Substitute into the location m ($0 \leqslant m < 16$) of register D, indicated by the p_{11}–p_{14} digits of the current command	D_m
18	Add into the content of location m ($0 \leqslant m < 16$) of register D, indicated by the p_{11}–p_{14} digits of the current command, and hold the sum	$+D_m$
19	Subtract from the content of location m ($0 \leqslant m < 16$) of register D, indicated by the p_{11}–p_{14} digits of the current command, and hold the difference	$-D_m$
20	Null	Z
21	Substitute into register H the digits from group p_1–p_{10} of the entering number	H_l
22	Substitute into register H the digits from the group p_{11}–p_{20} of the entering number	H_u
23	Substitute into the sequence register the digits entering in the group p_{11}–p_{20}	S

TABLE 3 (*Continued*)

DESTINATION FUNCTION GATES

Code Number	Function	Symbol
24	Count the number of digits entering into the content of the sequence register (each unit digit has unit weighting)	cS
25	Add into the content of the sequence register the digits entering in the group p_{11}–p_{20} and hold the sum	$+S$
26	Substitute the p_{11}–p_{20} digit group of the interpreter register by the p_{11}–p_{20} group of digits entering. Hold the content until the next command enters when the corresponding group will be added in .	$+K$
27	Substitute in the low speed store No. 1 into the location n ($0 \leqslant n < 1024$), indicated by the p_{11}–p_{20} group of the current command	n_1
28	Substitute in the low speed store No. 2 into the location n ($0 \leqslant n < 1024$), indicated by the p_{11}–p_{20} group of the current command	n_2
29	Substitute in the low speed store No. 3 into the location n ($0 \leqslant n < 1024$), indicated by the p_{11}–p_{20} group of the current command	n_3
30	Substitute in the low speed store No. 4 into the location n ($0 \leqslant n < 1024$), indicated by the p_{11}–p_{20} group of the current command	n_4
31	If one or more unit digits received, stop the sequence ; do not proceed to the next command	T

then understood as the source. The symbol (K) for the interpreter source is used only in the early design stage of a programme when the sub-address number may be unknown until a later stage in the design when the symbol is replaced by the actual number.

The following examples illustrate the notation :

(i) $(C) \rightarrow A$: Read out the content of register C and substitute it into register A.

(ii) $(57) \stackrel{+}{\rightarrow} A$: Read out the content of store position 57 and add it into register A.

(iii) $57 \rightarrow S$: Read from the interpreter the digits in positions p_{11}–p_{20}, i.e. 57 p_{11}, and substitute them into the sequence register. Note that (K) has been replaced by the number 57 with no brackets.

(iv) $p_{11} \stackrel{+}{\rightarrow} D_4$: Read a p_{11} unit from the p_{11} source and add it into location No. 4 of the register D.

Commands comprising a programme are written one under the other in the order in which they are entered into the store. Usually commands are placed into successive locations. To the left of each command symbol a serial number

or the store location into which the command is to be transferred upon entry into the computer is written. To the right of the list of commands the effect or action of commands or groups of commands is stated and any additional notes are made.

Two vertical lines placed to the left and three to the right of the list of commands provide one column between the location number and the commands and two adjacent spaces between the commands and the notes on the right. An example of the layout will be found in Table 4.

Commands which change during the course of the operation of the programme are surrounded by square brackets, [], whilst commands which are not actually used by the machine directly as such, but which are used as quantities from which other commands may be built, are surrounded by angle brackets, <>, and are called " pseudo- " or " latent commands ". When written symbols are to be grouped together they are overlined, e.g. $\overline{n+p+r}$.

It is sometimes necessary in descriptions of programmes to refer to the content of a register at the completion of a particular command. Such a quantity is indicated by a prime. Thus $(A)'=x$ written to the right of a command means " the content of register A at the completion of the command is x ".

During the normal course of the operation of the machine through the programme, commands are adopted from successively numbered locations in a uniform sequence. Such a sequence is frequently broken, the next command being adopted from a location other than the next in numerical order. In such cases the serial number of the command next selected is indicated to the right of the current command in the second or rightmost of the blank columns formed by the vertical lines and is enclosed in square brackets.

A list of commands is sometimes broken into convenient groups of commands by underlining the last of such a group.

It is not always possible to denote quantities entered into the computer in the form of commands or pseudo-commands. Special numbers may be written in the decimal scale during the initial stages of design of a programme and later converted into a form which is suitable for recording onto the input medium, the paper tape. Such special symbols which the programmer may use should be placed in a position which would otherwise be occupied by a command symbol.

Sometimes, but only rarely, the sub-address may not be entered directly into the command symbols constructed from Tables 2 and 3. In such cases the sub-address number is written to the left of the command symbol, e.g. 57 ; $(C)\rightarrow A$ has the effect of the command $(C)\rightarrow A$, but is also the number $57\times 2^{-9}+46\times 2^{-19}$, and may be used as such. The number 57 affects only the time in the computer routine at which the transfer takes place.

It is sometimes convenient to use the notation for digit-periods to represent the important unit occurring in a sub-address. Thus a number, say n, representing a location, is normally stored in the p_{11}–p_{20} group only and may be written in the command in the form np_{11}, but usually the p_{11} is implied. In other cases the notation is extended to cover digits in which the p_{11} unit is not the important unit, e.g. mp_{15}.

VIII. Programme Devices

Not all the 32×32 possible combinations of source and destination transfers are useful and some inefficiency in the use of digit space in a programme must be accepted. However, compared with a straight one-address system considerable extra flexibility and some saving in the total number of commands in programmes which serve similar purposes is attained.

Certain elements in the design of the machine distinguish the method adopted for programme design from those used for other computers. These are the main factors ; first, the computer possesses more than one accumulating register. In fact A and C and all 16 locations in D are capable of the elementary arithmetical functions, although register A has extra facilities and in practice becomes the main accumulator and is used in preference to other registers when available. The accumulating registers and the registers B and H may be used as " working space " during a calculation and a considerable saving of commands is attained, compared with an equivalent programme designed for a one-address system, by calling transfers between these registers without reference to the main store. Further, these registers are frequently used at one stage of a programme for the assembly of data to be operated upon by parts of the programme immediately to follow. Thus, for instance, prior to a programmed division the dividend is transferred to register A and the divisor to C and the division which follows uses only one further register, B, and the quotient appears in register A.

Facilities are available for making discrimination, that is, choosing alternative paths through a programme according as a condition is or is not satisfied by the content of a particular register. Discrimination is usually made according to the state of the sign of a register content, that is, the p_{20} digit of the content in question is either unity for negative content or zero for positive content including zero. Sign conditions of (A), (C), and (D_m) $(0 \leqslant m < 16)$ may be detected by the sources $s(A)$, $s(C)$, and $s(D_m)$, and also on the content of B by the source (R). The evenness or oddness of (A) may be detected by the source $p_1.(A)$. Digits from these gates may be transferred to any desired destination but the most common ones are described below.

Breaks in the normal sequential course through a programme of commands are frequent. Commands which cause these breaks involve sequence register destinations. The choice of alternative paths through a programme can be made via the destination cS. Usually only one digit is allowed to enter this gate. Thus, the command

$$s(A) \xrightarrow{c} S$$

from location n in the store will cause the next command to be selected from the position $n+1$ immediately following if the sign digit of (A) is zero, or from the position $n+2$ if unity.

Sometimes, however, numbers of digits may be counted in the same transfer, thus making possible a choice between a number of alternative operations to follow. Thus the command in position n

$$(A) \xrightarrow{c} S$$

will, if r unit digits are contained in (A), cause the next command to be selected from position $n+r+1$. Such a break in the uniform progress through a programme is called a " conditional shift of control ".

The substitution of an entirely fresh number into the sequence register via the destination S causes an " absolute shift of control ". Thus the command in position n

$$(A)\rightarrow S$$

replaces (S) by the 10 most significant digits of (A), i.e. those in the period p_{11}–p_{20}, neglecting any digits in the group of lower significance, p_1–p_{10}. Thus, if (A) is the partitioned datum

$$(A)=qp_{11}+rp_1, \qquad 0\leqslant q<1024,\ 0\leqslant r<1024,$$

(S) becomes qp_{11} and the next command is withdrawn from position q, that is,

$$(S)'=q.$$

A " relative shift of control " can be attained via the destination $+S$. Thus the command in location n

$$(A)\overset{+}{\rightarrow}S,$$

where again $(A)=qp_{11}+rp_1$, causes control to be shifted " forward " to $n+q+1$, that is,

$$(S)'=n+q+1.$$

The extra unit is introduced because the addition of a p_{11} unit, which controls the normal advance of (S) from n to $n+1$, is not removed. Normally such an operation represents an increase in the actual value of (S). Since, however, (S) is a number stored modulo 1024 in terms of p_{11} as a unit, the addition of the complement of q, i.e. $\overline{1024-q}\ p_{11}$, will cause $(S)'$ to be $n-q+1$ to which control will be shifted " backwards ".

The sequence register may also be called as a source. Thus a command in location n

$$(S)\rightarrow A$$

substitutes $\overline{n+1}\ p_{11}$ into register A. Such an operation is usually followed by a command causing (S) to change, say, to q, a point in the programme which might be reached by more than one route. As the programme proceeds from location q a reference to the route along which q was reached is stored for use as (A).

Many of the functions just described are used in conjunction with the interpreter register as source. In this case the digits in positions p_{11}–p_{20} of the current command are available for use. This is so whenever the sub-address of the current command does not refer to a position in the stores or register D. Thus, in accordance with the notation previously stated, the command in location n

$$57\rightarrow S$$

would cause the number $57p_{11}$ to be transmitted from the interpreter and to be substituted into the sequence register. Control would be shifted to 57.

The command in location n

$$57\overset{+}{\rightarrow}S$$

would cause control to be shifted forward to $57+n+1$.

The notation is important here ; the round bracket is not present and the interpreter is implied as the source. By contrast, the command if written as

$$(57)\overset{+}{\rightarrow}S$$

would cause the most significant group of 10 digits of the content of store position 57 to add into (S).

In conjunction with the H register current commands may be used to store useful constants. To form a full 20-digit datum from a pair of 10-digit groups available in the current commands the H register is used for shifting one group from the p_{11}–p_{20} period to the p_1–p_{10} period. A typical set of commands would be :

Location of Command	Command
n	$r\rightarrow A$
$n+1$	$s\rightarrow H_u$
$n+2$	$(H_l)\overset{+}{\rightarrow}A$,

where the first places rp_{11} into A, the second places sp_{11} into H, which is read out by the third as sp_1 and added into A. Thus $(A)'=rp_{11}+sp_1$ or $r2^{-9}+s2^{-19}$ numerically.

The basic constants, the p_{20}, p_{11}, and p_1 unit digits, are used largely as follows : p_{20} for the constant —1 and sign reversals and modulo 2 counting, p_{11} for unit counts, in particular for changes by unity to the sub-address of commands changed by the programme, and p_1 for unit counts to complete 20-digit numbers etc.

The operation of the computer may be stopped either conditionally or unconditionally. Thus the command

$$s(A)\rightarrow T$$

will stop the sequence of operation if (A) contains a p_{20} unit, otherwise not. An unconditional stop is obtained by a command such as

$$p_{20}\rightarrow T \text{ or } p_{11}\rightarrow T.$$

Manual control may be applied in any case by use of the switches in registers N_1 and N_2. Thus

$$(N_1)\rightarrow T$$

will stop the operation only when at least one switch of N_1 is depressed.

Registers N_1 and N_2 are frequently used to hold parameters which may be varied at will by the operator, e.g. parameters which control the rate of convergence of an iterative process.

A register may read out of one of its source gates and receive into one of its destination gates at the same time. Thus the command

$$(A)\overset{-}{\rightarrow}A$$

causes (A) to be read out and subtract into itself so that $(A)'$ is zero. Alternatively,

$$c(A)\overset{-}{\rightarrow}A$$

causes (A) to be read out and at the same time cleared to zero and the original content transmitted is received and subtracted from zero. The effect is to replace (A) by $-(A)$.

Simple multiples of (A) may be obtained by the sources $\frac{1}{2}(A)$ and $2(A)$ functions. Thus

$$2(A)\overset{+}{\rightarrow}A$$

gives $(A)'=3(A)$ and so on.

Operations of the form $(D_m)\overset{+}{\rightarrow}D_m$ and $(D_m)\overset{-}{\rightarrow}D_m$, i.e. doubling and clearing the mth location in D are permissible.

Multiplication by a quantity already stored in register B may be caused by the command

$$(B)\overset{\times}{\rightarrow}B,$$

which causes (B) to be read out and to enter the multiplying gate and re-enter B and so start the multiplication process.

The command $c(A)\overset{\times}{\rightarrow}B$ is frequently useful in forming continued products when (A) is already the product of a previous multiplication.

Occasions arise when " time wasting " must take place as when a command is removed from the programme leaving a " free space ". Such spaces may be filled with what may be called " null commands ", either $(A)\rightarrow A$, which reads (A) back to the same place, or by any " store to store " command like $(n)\rightarrow n$.

It will be noticed that the code representing such a command is equivalent to the number $np_{11}+0p_1$, so that a blank space $0p_{11}+0p_1$ causes the content of store location 0 to be resubstituted into the same location. Thus, blank spaces, if adopted as programme commands, affect no registers or stores.

As a simple example of the construction of a programme consider the commands which will cause the product of the modulus of the content of store location 47 to be multiplied by 2^{-4}, rounded off to 19 binary places, transferred to location 12, and control shifted to 132. This set of commands, listed in Table 4 in the standard manner, is entered at the leading command placed in location m.

The action caused by these commands may be followed from the description in the right-hand column. The command in $m+1$ tests the sign of (A), and causes a conditional shift of control, and, if positive, (A) is replaced by its complement *twice*, thus leaving it positive. This set of commands is not unique and other arrangements can be contrived which will provide the same final effect.

IX. Input and Output

Programmes, following the initial design stages, are written in the symbolic form in the order in which they are to be provided to the computer. Thence they may be transcribed onto the input medium. Both commands and numerical data are supplied to the machine on punched paper tape onto which up to 12

holes per row can be recorded. Single commands when punched onto tape normally occupy two rows consisting of two groups of 10 binary digits each corresponding to 10 of the 12 punching positions in a row. The remaining two positions are reserved for special controlling purposes. One column is punched with the binary representation of the sub-address intended to be placed in periods p_{11}–p_{20} and the following column with the other addresses to enter p_1–p_{10}. The latter is supplied with a hole in the 11th or X position, and distinguishes p_1–p_{10} rows from p_{11}–p_{20} rows.

When the machine is switched on all registers and the store are cleared to zero and a group of 20 commands is transferred to the first 20 locations in the store (0–19) via the fixed wiring on a set of rotary stepping switches. This set

TABLE 4

A " ROUNDED MODULUS " ROUTINE

Location	Command		Action
$m+0$	$(47)\rightarrow A$		$(A)'=(47)$
$m+1$	$s(A)\overset{e}{\rightarrow}S$		Tests $(A)\gtrless 0$
$m+2$	$c(A)\overset{-}{\rightarrow}A$		Skipped if $(A)<0$, if $(A)\geqslant 0$, $(A)'=-A$
$m+3$	$c(A)\overset{-}{\rightarrow}A$		If at $m+1$ $(A)\leqslant 0$, then $(A)'=-(A)$, otherwise $(A)'=(A)$
$m+4$	$32\rightarrow C$		Sets a p_{16} unit to C from interpreter
$m+5$	$c(A)\overset{\times}{\rightarrow}B$		Forms $\mid(47)\mid\times 2^{-4}=(A)'$
$m+6$	$(R)\overset{+}{\rightarrow}A$		Round-off to 19 binary digits
$m+7$	$(A)\rightarrow 12$		$(12)'=\mid(47)\mid\times 2^{-4}$
$m+8$	$132\rightarrow S$	[132]	Shifts control to 132

of commands is sufficient to assemble the digits from pairs of adjacent rows of the input tape into full 20-digit data and to store them in successive positions in the store immediately following those already filled. This set of commands is called the " primary routine ".

In cases of commands which have zero as the sub-address the first of the pairs of rows may be omitted. In all commands a punch in the X position is provided on the column containing the source and destination addresses so that the total number of such holes is equal to the number of commands punched.

A special keyboard of 32 keys is provided for transcribing programmes onto tape. Each key corresponds to a number in the scale 32, to a source, and to a destination. The number and the corresponding address symbols of Tables 2 and 3 are indicated on each key. Two keys are depressed in succession before one of a set of " punch keys " is depressed which initiate the punching. Thus the sub-address is punched by depressing first the key m, say, followed by n and the punch key is depressed when the 10 binary digits are punched in order on the

tape row currently under the tape punches. The most significant digit is punched in the first position, the others follow in order to the lowest in the 10th position. The integer thus punched will be : $32m+n$. Following the punching of a row the tape is automatically moved forward to the next row. The source key is depressed followed by the destination key and the " X-punch " key is depressed. The source and destination code is then punched together with punch in the X position. There is no need to remember the actual code numbers of the addresses. The keying for a typical command is therefore :

m, n, punch ; source, destination, X punch.

As an example, the programme at the end of the previous section would be punched by depressing keys in the sequence shown in Table 5. In this table semicolons imply the depression of the punch key, whilst the colons imply the depression of X-punch key.

TABLE 5

PROGRAMME ENCODING METHOD

Example shown for the routine in Table 4

Depression of Keys	Corresponding Written Command
1 , 15 ; (0) , A :	$(47)\rightarrow A$
$s(A)$, cS :	$s(A)\overset{c}{\rightarrow}S$
$c(A)$, $-A$:	$c(A)\overset{-}{\rightarrow}A$
$c(A)$, $-A$:	$c(A)\overset{-}{\rightarrow}A$
1 , 0 ; (K) , C :	$32\rightarrow C$
$c(A)$, $\times B$:	$c(A)\overset{\times}{\rightarrow}B$
(R) , $+A$:	$(R)\overset{+}{\rightarrow}A$
0 , 12 ; (A) , 0 :	$(A)\rightarrow 12$
4 , 4 ; (K) , S :	$132\rightarrow S$

All the symbols in Table 5 are those listed in Tables 2 and 3 and almost all are the same as are used in writing programmes onto paper. The exception, symbol (K), exists on the key representing the interpreter as source and is implied in the written commands. With this latter provision the list of written commands may be translated into keying operations with no rewriting and very little mental effort.

In cases where a 20-digit binary number to be entered into a programme is not a command and is not necessarily in the form of a pseudo-command, the number, multiplied by 2^{19}, must first be converted to the scale of 32 and keyed in the same manner as a normal command. Such data are constants like e^{-1}, $\pi/4$, $2^{-\frac{1}{2}}$, etc. Thus, for instance, in the final steps of compiling the programme $\pi/4$ would be entered by keying the sequence :

12, 18, punch ; 3, 31, X punch,

since $0{\cdot}7853982\times 2^{19}=12, 18, 3, 31$ to the nearest unit in scale of 32.

Normally very few such cases occur, as decimal data in large quantities are normally punched in one of two ways ; each decimal digit is punched either in a decimal code similar to that used on cards of the Hollerith type, i.e. the 1 in 10 position code, or as an equivalent binary tetrad. When using either of these codes part of a programme previously entered into the store must be designed so as to assemble the digits transferred from the punch holes into the binary representation needed later by the programme.

Results are normally page printed or punched onto tape. The computer is provided with a suitable printing or output punching programme, which controls the transformation of the results from binary form as stored internally to the corresponding decimal digits, which are printed or punched in descending order of significance. The printer is provided with a full range of letters, brackets, stops, and positive and negative signs, which may be used by suitable programmes.

Decimal results may be punched in the form of binary tetrads or as single holes in the 1 in 10 code system. This allows punched results to be reinserted into the computer during a later calculation.

X. References

Gill, S. (1951).—The diagnosis of mistakes in programmes in the EDSAC. *Proc. Roy. Soc.* A **206** : 538.

Pearcey, T., and Hill, G. W. (1953).—Programme design for the C.S.I.R.O. Mark I Computer. II. Programme techniques. *Aust. J. Phys.* **6** : 335.

Wheeler, D. J. (1950).—Programme organization and initial orders for the EDSAC. *Proc. Roy. Soc.* A **202** : 573.

Wilkes, M. V., Wheeler, D. J., and Gill, S. (1951).—" The Preparation of Programmes for an Electronic Digital Computer." (Addison-Wesley Press Inc. : Cambridge, Mass.)