621.374.32 : 681.142

Paper No. 2013 M
Mar. 1956

THE PROGRAMMING STRATEGY USED WITH THE MANCHESTER UNIVERSITY MARK I COMPUTER

By R. A. BROOKER, M.A.

(*The paper was first received 29th September, and in revised form 8th December, 1955. It was published in March, 1956, and was read at the* CONVENTION ON DIGITAL-COMPUTER TECHNIQUES, *10th April, 1956.*)

SUMMARY

The paper gives an account of the programming strategy developed for use with the University of Manchester electronic computer Mark I, a typical 2-level storage machine of which a brief description is included. The topics dealt with include: the method of storing and calling in routines; the representation of instructions outside the machine (tracing the reasons for the use of a teleprint or scale-of-32 code); the organization of the library of sub-routines; the mechanism of interpretive routines for double-length and floating-point operations; matrix operations; partial differential equations; an attempt at automatic programming; mistake diagnosis in programmes; and measures for dealing with machine breakdown. Finally, the basis of the strategy is considered in relation to the Mark II machine.

(1) INTRODUCTION

The Manchester University computer was the first of a series of electronic digital computers manufactured commercially, and based on principles explored with the University prototype machine. The first two models of this series (including the one at Manchester) were referred to as the Ferranti Mark I type, subsequent models being known as the Ferranti Mark I* type. Although these later models (many of which are now available in this country) differ from the Mark I in several matters of detail, the general structure remains the same, and for this reason it may be of interest to present some of the strategical considerations which have influenced the staff at Manchester in the manner of using the machine. Also it is of interest to see how these conclusions will be affected by the design of the Mark II machine.

(2) BRIEF DESCRIPTION OF MACHINE

A full description of the machine is given in Reference 1. For the purposes of this discussion it is convenient to summarize the main characteristics as follows:

The working store (cathode-ray tubes) contains eight pages.
The auxiliary store (magnetic drum) contains 512 pages.

1 page ≡ 64 lines; 1 line ≡ 20 binary digits.

The time to transfer one or two pages of information from the auxiliary store to the working store is 40 millisec; the reverse operation takes 90 millisec.

Each instruction occupies a single line, and is of the single-address variety. Modification of addresses by one of eight B-additives is used.

A number occupies two consecutive lines of the store. The binary point may be regarded most simply as lying at either end of the 40 binary-digit row, or, at slight extra cost in instructions, as in the middle of the row. The operations of addition, subtraction and multiplication occupy 1–2 millisec, including access to the working store.

The input medium is teleprint tape which is read by a photoelectric reader at a maximum rate of 200 characters/sec.

The output medium is teleprint tape and/or page printing, the rates being respectively 16 and 6 characters/sec.

Mr. Brooker is in the Department of Electrical Engineering, Manchester University.

(3) STORAGE OF ROUTINES

Since the working store is usually too small to hold all the material in a complete programme, everything is normally kept in the magnetic store, and the individual routines, vectors, etc., are transferred to the working store only when used. A further reason for this arrangement is that the magnetic drum is the more reliable storage medium, so that material written on it can be regarded as safe as if written on paper—indeed more so if there is any chance of losing the paper in question.

From the time-economy aspect the arrangement can be justified as follows: The time to transfer two pages of instructions is 40 millisec. If 50% of the instructions are included within some kind of loop and so repeated, say four times on the average, the time taken to execute the routine is approximately $64 \times 4 \times 1{\cdot}2 \simeq 300$ millisec, which bears quite a reasonable ratio to their access time. Of course, not all routines are as favourable in this respect, e.g. the cosine routine takes only 70 millisec, so that the access time is more than 50% of the execution time. However, even where it is 100%, the loss of time is important only if the routine forms part of the inner loop of the programme. In such cases it is advantageous to retain the most frequently used instructions in the working store, e.g. the sequences for carrying out floating-point operations.

For the rest of the programme, however, the system adopted is to keep the routines in the magnetic store and to transfer a copy of each routine to the first quarter of the working store (namely locations 0, 1, . . . 127) when it is 'called in'. Each routine thus obliterates the routine which called it in, so that with sub-routines it is necessary to recall the original routine from the magnetic store when the sub-routine has finished its task. The operations necessary to do this form the purpose of a special sequence of instructions which is kept more or less permanently in the working store throughout the operation of the programme. The details of the system can be found in Chapter 3 of Reference 2.

For many problems there is no question of using the entire magnetic store, so that there is no need to economize on instructions; indeed to do so is regarded as reprehensible, since it wastes the programmers' time. There are, however, occasions when it is worth while to economize on instructions in order to save space in the working store, and it is customary to pack them into one or two pages if otherwise they would just overflow. To do so makes the routine tidier and usually has time-economy effects.

There are, of course, problems where one can easily use 10^6 digits of storage and still not be satisfied, but here the space shortage applies mainly to working space rather than to routines, and since these usually have to be written down manually, this in itself has a limiting effect.

(4) THE INPUT ORGANIZATION

In machines which do not possess a magnetic-drum store, or use punched cards or magnetic tape instead, emphasis is placed

on the working store, where it is usual to pack routines 'head to tail', i.e. the last instruction of each routine is followed by the first instruction of the next one. A scheme is then adopted which enables routines to be written as if for a standard set of locations, e.g. 0, 1, 2, etc., the necessary modifications to enable the routine to operate in the locations in which it is going to be used (which, of course, are not necessarily known in advance—certainly not with library routines) being carried out by the input programme. It is usual to write numerical quantities in decimal form, so that the input programme will probably include decimal-to-binary conversion of the address part of each instruction. For at least two reasons, therefore, the form in which instructions are written on paper bears little resemblance to the form in which they are displayed on the monitors of the machine. A further development is the 'floating address' system, which enables the programmer to refer to any word in the programme by means of a label instead of by its address in the store. While the Cambridge (England) group have pioneered developments in this direction, we at Manchester have inclined to the opposite direction, i.e. towards a closer correspondence between the instruction as it is written in the programme and as it is displayed on the monitors of the machine.

This has been made possible for two reasons. First, as already explained, all routines in the Manchester machine operate in the same storage locations, namely 0, 1, . . . 127, so that a routine written for these locations may be used directly. Second, the use of a scale-of-32 numbering system eliminates the need for decimal-to-binary conversion of addresses. This is largely a consequence of the way magnetic transfers operate. For this purpose the working store is partitioned into eight pages, to any of which it is possible to transfer the content of any magnetic page, and vice versa. Rather than ignore these partitions when allocating working space, it is preferable to bear in mind a picture of the eight pages, each page being divided into two columns and each column into 32 lines. It was Dr. Turing who suggested naming the 32 lines in each column by the 32 characters in the teleprint code, namely

/E@A:SIU½DRJNFCKTZLWHYPQOBG"MXV£

and likewise to name the 16 columns by the first 16 characters of the sequence. (Originally it was intended to have 16 pages, or 32 columns, when two teleprinter characters would exactly suffice to enumerate the store.) It is just as convenient to know the lines in the store by their teleprint labels as by their decimal equivalents. Thus the address part of each instruction can be written down directly in teleprint form. If the remainder of the instruction is treated likewise, the instruction as written on paper is simply a shorthand form of the 20 binary digits which are displayed on the monitors. The correspondence has been made even closer by arranging the monitors to break the line into four groups of five digits, each separated by a space, a feature which greatly facilitates 'peeping'—a topic which is dealt with elsewhere.

(5) THE LIBRARY

The number of library routines has been kept as small as possible by ruthlessly excluding routines of marginal utility. Thus, no attempt has been made to provide alternative versions in which, for example, emphasis is laid on time or space economy measures. Instead, those that are available are described in detail, so that a user can modify it if he thinks this desirable. Thus there are general-purpose routines for decimal input and output; the division, square-root, cosine, exponential, logarithm and arc-tangent functions; and floating-point arithmetic and floating-point versions of the function routines. Other than these, however, there are very few routines, and as a result the entire library can be contained on a single length of teleprint tape which takes less than ½ min to read into the machine. This is the usual procedure, although, in fact, the auxiliary store is large enough to store them permanently. There is also a small library of complete programmes dealing mainly with matrix work.

(6) INTERPRETIVE ROUTINES

Interpretive routines are useful means for organizing double-length and floating-point arithmetic. In any interpretive routine it is important to make the selection and decoding of the instruction words as rapid as possible, for this forms part of the inner loop. With the Manchester machine this is most simply done by means of a B-modified control transfer instruction. Unlike other one-address code machines, control transfers on the Manchester machine involve a double reference: thus J,3 does not mean 'jump to 3,' but 'jump to the location number given in location 3.' Combined with the B-modification facility, the result is 'jump to the location number given in location $3 + r$', r being the content of the B-line in question. The second reference is known as a 'control number', because the first step in executing such an instruction is to transfer the content of the address named, 3 or $3 + r$, to the control register.

If each 'instruction' is simply equivalent to calling in a particular sequence of instructions, the 'programme' amounts to nothing more than a list of control numbers, and each sequence terminates by advancing by 1 a 'control' number, held in a B-line reserved for this purpose, and then executing the appropriate

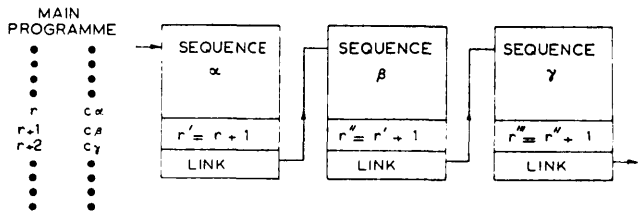

Fig. 1

B-modified control transfer instruction (the link). Fig. 1 illustrates the path of control for a 'programme' of three 'instructions'.

If it is desired to introduce, say, a one-word parameter for use with a particular sequence, this can be included in the 'programme' immediately following the control number which initiates the sequence in question. The sequence itself will be arranged to select this word by any appropriate instruction modified by the control additive r. In this case the sequence advances control by 2 (instead of 1) before executing the link.

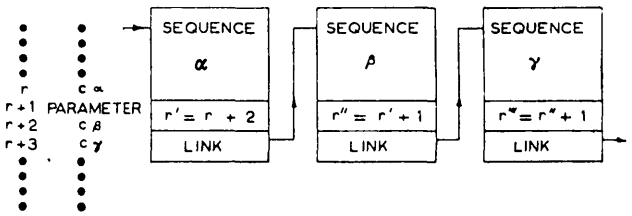

Fig. 2

Fig. 2 illustrates the modifications in the case where sequence α involves a parameter word.

A routine of 'instructions' occupies the first quarter of the working store in the usual way, and the instruction sequences themselves are stored elsewhere in the working store during the operation of the 'programme'.

(7) FLOATING-POINT ARITHMETIC

An interpretive routine for floating-point binary-arithmetic operations has been made on the lines described in the previous Section. The numerical part and the exponent of a number each occupies a 40-digit location; this is somewhat extravagant for the exponent, but the alternative of packing the two parts into a single storage location (and subsequent unpacking) is less economical in time of execution. This is one instance of buying economy in time at the expense of economy in space, justified in this case because space is almost free for the class of problem involving floating-point operations. Approximate times of floating-point operations are: addition (without subsequent normalization), 35 millisec; multiplication, 10 millisec; normalization, 25 millisec. The routine itself is used infrequently, but a great deal of floating-point work is done through the medium of the 'autocode' routine (described in Section 12), which is based on this interpretive routine.

(8) DOUBLE-LENGTH ARITHMETIC

Double-length and multi-length arithmetic operations can be coded fairly easily on the Manchester machine, because the design of the arithmetic unit and the related instructions were largely influenced by the work carried out with the prototype machine; much of this work was concerned with testing Mersenne numbers for primality—calculations which involved handling numbers of several hundred binary digits. However, very little double-length work has been done on the Manchester machine, and this has mainly been the calculation of 40-digit constants for use in library sub-routines. We attribute this to the choice of the 40-digit location: in this respect the difference between 30 and 40 digits appears to be rather an important one.

(9) MATRIX OPERATIONS

Because of the limited capacity of the working store, a matrix is normally retained in the magnetic store and processed in the working store in parts, for which purpose the simplest plan (provided that the order of the matrix does not exceed 64) is to record one column (or row) on each page pair (track), so that it may be transferred to the working store in a single operation. Columns are used in preference to rows, because columns of figures are easier to copy and check when preparing data for the machine. Moreover, this choice assists 'peeping', because the contents of a page are displayed columnwise on the monitors. It is not necessary to work with either rows or columns, however, and a programme exists for inverting a matrix partitioned into 4×4 sub-matrices. Although this form of partitioning makes the most economical use of the storage (since a 4×4 sub-matrix is exactly half a page), the programme itself is unwieldy, for it involves, in addition to the usual logic of elimination, a set of routines for carrying out arithmetic operations, including inversion, of the sub-matrices.

For a column vector a common scale factor is assigned to all the elements, so that it is recorded in the form $a = \alpha \,.\, 2^p$, where the length $\rho = (\alpha \,.\, \alpha)^{\frac{1}{2}}$ lies in the range $(\frac{1}{2} \leqslant \rho < 1)$; routines are available for combining, normalizing, and taking the dot product of vectors in this form. These routines facilitate the operation of matrix multiplication and the collineatory transformation to codiagonal form.[3]

The formal vector routines are not used in the programme for solving simultaneous equations, which is very similar to that used on the Ace.[4] The present Manchester programme is restricted to matrices of order up to 40 and deals with any number of right-hand sides (up to 40). At each stage we choose as pivotal element the largest coefficient in the reduced matrix, since with a magnetic drum all columns are equally accessible, whereas if the reduced matrix is recorded on a deck of punch cards it is appropriate to choose the pivotal element from the first column. Moreover, in order to avoid making decisions about the size of the answers, we use a floating binary point in the back-substitution process. This is not unreasonable if the number of right-hand sides is small, since only $0(n^2)$ operations are involved at this stage; however, it considerably extends the calculation time in the case of matrix division where $0(n^3)$ operations are involved at the back-substitution stage. The calculation time for a set of 30 equations with one right-hand side is approximately 10 min; to invert the matrix takes 15 min.

(10) PARTIAL DIFFERENTIAL EQUATIONS

Although the size of the magnetic-drum store permits the solution of 2-dimensional and possibly even 3-dimensional partial differential equations, it is generally agreed that for this kind of problem the working store is inconveniently small. Thus it is desirable to retain in the working store the data for an entire 2-dimensional mesh; otherwise it is necessary to partition it, which involves overlapping meshes. With the Mark I machine the cost of this is mainly reduced programming convenience, but in the Mark II machine, where the relative access times of the two stores are very different, the size of the working store (some of which is an optional extra) may also detract from the benefits which can be derived from increased speed of computation (see Section 15). However, the full complement of one thousand and twenty-four 40-digit words should be quite adequate for this purpose.

The B-modification facility considerably reduces the time of execution of the inner loop in an iterative scanning process, where the improved value at each mesh point is a function of the existing values at neighbouring points. To appreciate this, consider the simple case of a rectangular mesh where the points are stored in location numbers $a + bi + cj$, where i and j are row and column indices respectively. If, for example, the iteration formula for any point ϕ_0 involves eight surrounding points, as shown in Fig. 3, the location numbers of these points can be written in

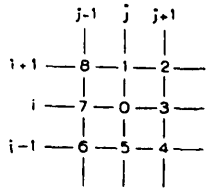

Fig. 3

the form $A_k + bi$, $k = 0, 1, \ldots 8$. Thus, when scanning the jth column essentially only one parameter need be altered, namely bi, which is advanced by b when moving to the next point. If this quantity is stored in a B-line, the values at the nine points can be selected by means of B-modified (store to accumulator) instructions having $A_0, A_1, \ldots A_8$ as presumptive addresses. In this way only one quantity need be adjusted instead of the nine addresses which would otherwise have to be altered if the B-modification facility were absent. In Laplace's equation, for instance, the time of execution of the inner loop is reduced by a factor of 3.

(11) NON-NUMERICAL PROGRAMMES

The capacity of the magnetic store (256 tracks) has so far satisfied the most ambitious programmers, although many problems have involved between 150 and 200 tracks. For non-numerical problems the storage capacity for instructions is virtually unlimited, and it is possible to design programme models

of many logical and physical systems. An example of such a model is the programme for playing draughts, written by Mr. C. Strachey. An early version of the Pegasus computer was also simulated on the machine in order to try out some library routines before Pegasus was actually built. The programme described in the next Section really amounts to a model of a hypothetical machine.

(12) THE AUTOCODE PROGRAMME

A conversion-interpretive routine exists for executing programmes prepared in a very simplified form, illustrated by the following sequence of instructions for evaluating the limit of the arithmetico-geometric sequence, $a_{n+1} = \frac{1}{2}(a_n + b_n)$, $b_{n+1} = \sqrt{a_n b_n}$, for the initial values $a_0 = 1$, $b_0 = 0{\cdot}5$.

$2v_1 = 1$	sets a_0
$v_2 = 0{\cdot}5$	sets b_0
$1v_3 = v_1 \times v_2$	$a_n b_n$
$v_4 = v_1 + v_2$	$a_n + b_n$
$v_1 = v_4/2$	a_{n+1}
$v_2 = F_1(v_3)$	b_{n+1}
$v_4 = v_1 - v_2$	$a_{n+1} - b_{n+1} (> 0)$
$j1, v_4 > 0{\cdot}000001$	test for convergence
$j2$	starts the programme

A full description of the code is given in Reference 5. The main points considered in the design of the code and the means of mechanizing it will be summarized here.

There are two main difficulties in conventional programming which should be taken into account in any attempt to simplify coding, namely the scale-factor difficulty and the necessity for distributing the material over two levels of storage. The first difficulty can be largely removed by the use of floating-point arithmetic at the cost of many more machine operations; the second can be removed by using the machine as a one-level storage machine in the following way. After reserving the first 128 tracks on the drum for instructions, the 40-digit locations on the remaining 128 tracks may be labelled $v_1, v_2, \ldots v_{8192}$. To gain access to any one of them the interpretive routine first determines the track on which it lies, then transfers the contents of the track in question to the working store, and finally selects the particular line required. The access time for any operand is thus at least 40 millisec (the time for a reading transfer), and for the corresponding recording operation at least 90 millisec (the time for a writing transfer). These figures about match the times of floating-point operations, so that little is to be gained by improving the access time.

The position would be very different, however, if the arithmetic operations took only a few milliseconds or less (as, for example, in the Mark II machine), for in that case most of the time would be spent in selecting and recording numbers on the drum. To achieve any sort of balance in these circumstances it would be essential to take advantage of any natural ordering of the numbers. It usually happens that any such ordering is reflected in the allocation of storage in the hypothetical one-level store (however crudely this may be done), and the correspondence between this and the magnetic-drum store means that very often successive operands will be found on the same track. Many unnecessary magnetic transfers may therefore be eliminated by the conversion routine which examines the programme at the input stage before it is executed. Some cases cannot be resolved beforehand, however, and for these it should be possible to design equipment to test for duplicated transfers during the operation of the programme.

In the Mark I machine, however, coding can be simplified only at the cost of increasing the execution time of the resulting programme by anything from 5 to 50 times. This rules out the possibility of employing it on, say, partial differential equations, and its use will therefore be confined to moderately large problems where the coding must be completed quickly, and to 'small' problems. But customers with 'small' problems will not be encouraged if they have to spend more than half a day on learning to code. For this reason the principal consideration in the final form of the instructions was economy of description: what was aimed at was two sides of a foolscap sheet, with possibly a third side to describe an example. The psychological advantage which such an economy of description brings to the prospective reader makes the attempt worth while, even at the expense of considerable compression of the material (see Section 17).

(13) FINDING MISTAKES IN PROGRAMMES

Fault diagnosis at Manchester is done largely by 'peeping'. This is a derogatory term for the business of working through a programme manually on the machine while observing from the monitors the behaviour of relevant storage location. By itself the process does not make very efficient use of the machine, but combined with the use of facilities for correcting mistakes on the spot, it can, in the hands of an experienced operator, be made reasonably efficient. Indeed, when developing a very large programme (e.g. 4000 instructions), it would be intolerable to have to vacate the machine every time a mistake was found, correct the tape and then re-run the corrected version; it is far more efficient to use the machine to locate and correct mistakes until either the programme is working or an error is discovered which necessitates rewriting a page or two of instructions.

Fault-diagnostic routines of the kind described by Gill[6] have not been appreciated on the Manchester machine, largely because they interfere too much with the layout of the programme in the working store. It has already been explained that the first two pages (0, 1) are reserved for instructions. The duties of the remaining pages are likewise partly ruled by conventions. Thus page 2 is reserved for the powers of 2 and the routine changing sequence, page 3 for unsystematic working space, and so on. If one or two pages were reserved for a fault-diagnostic routine, it would be equivalent to reducing permanently the size of the working store.

There is, however, a switch on the console which enables the operator to change from the normal mode of operation to a mode in which the function character of each actual instruction is printed or punched immediately after it has been obeyed. The aggregate of output symbols thus gives a record of the course of the programme which can be studied at leisure. This facility is seldom used, however, largely because the record produced does not yield any information beyond the fact that the programme is going through the motions correctly, and the most difficult mistakes to find are those which are responsible for numerically incorrect results, not those which cause the programme to jump out of control.

(14) MEASURES AGAINST TRANSIENT MACHINE BREAKDOWN

Where possible, running checks of a mathematical nature are used to prevent loss of time through transient machine faults, e.g. the use of row sum checks in solving simultaneous equations. For many problems, however, mathematical checks are not available, or may be tedious to incorporate into the programme. In these circumstances the running check may simply consist of breaking the calculation into sections, executing each section twice and comparing the results (inside the machine) before passing on to the next section. An alternative arrangement is to print out some numerical quantities at the completion of each section, on the basis of which the operator judges whether the

calculation has been proceeding correctly before allowing the machine to proceed to the next section.

No attempt has been made to check the input-output units by means of self-checking numerical codes, because these units are no more unreliable than other parts of the machine. If it was only the reliability of these units that was in question, it might be worth while to incorporate a redundant code such as the 2-out-of-5 decimal code. However, a single error in output need not lead to serious consequences if some redundancy is present; e.g. when plotting curves from data produced by the machine it is not often that one needs to plot every point. Table-making is an exception to this rule, but so far we have managed to avoid making any large tables. Another exception is the preparation of programme tapes by the machine itself. Here there is no redundancy and a single incorrect digit may render the programme useless. Then the only alternative to a checked output is to run the job twice and compare the tapes.

(15) CONCLUSIONS

The balance between the operating speeds of the working store and the magnetic store enables the magnetic store to be regarded as the main store, with a consequent simplification of programming strategy. It has not been possible to preserve this balance on the Mark II machine, where the operating speed of the working store has been increased by a factor of 10, and that of the magnetic store by a factor of only 2. On the other hand, the working store will be four times larger than it is in the Mark I machine, so that far more computation can take place between references to the magnetic store, although to arrange this may mean that more attention will have to be paid to the distribution of material between the two levels of storage.

The paginated nature of the magnetic store will probably continue to be reflected in the use of the working store, which can most simply be regarded as consisting of 32 pages among which the programmer allocates routines, vectors, tables and other units of material into which the programme can be divided. Part of each block will, of course, be wasted (since such units do not arbitrarily limit themselves to any particular size), but in return for this the material can be transferred to and from the magnetic store as self-contained units.

Considerations of the above kind will be important for problems such as numerical weather forecasting and 3-dimensional Fourier synthesis, where the maximum possible benefit from the increased speed of computation is desirable.

The Mark II is also a floating-point machine, however, and for a large number of problems which the Mark I machine can already handle in the economically acceptable time, it is the floating-point facilities which will be of the maximum benefit rather than the increased speed of computation.

(16) REFERENCES

(1) KILBURN, T., TOOTILL, G. C., EDWARDS, D. B. G., and POLLARD, B. W.: 'Digital Computers at Manchester University', *Proceedings I.E.E.*, Paper No. 1362 M, August, 1952 (**100**, II, p. 487).

(2) 'Programmers' Handbook for the Manchester University Electronic Computer' [University of Manchester, September, 1953], (Mimeographed) (3rd Edition).

(3) BROOKER, R. A., and SUMNER, F. H.: 'The Method of Lanczos for Calculating the Latent Roots and Vectors of a Real Symmetric Matrix' (see page 114).

(4) WILKINSON, J. H.: 'Linear Algebra on the Pilot Ace', Paper No. 18, Proceedings of a Symposium on Automatic Digital Computation held at the N.P.L., March, 1953 (H.M. Stationery Office, London, 1954).

(5) BROOKER, R. A.: 'An Attempt to Simplify Coding for the Manchester Electronic Computer', *British Journal of Applied Physics* (1955), **6**, p. 307.

(6) GILL, S.: 'The Diagnosis of Mistakes in Programmes on the Edsac', *Proceedings of the Royal Society*, A, 1951, **206**, p. 538.

(17) APPENDIX: THE AUTOCODING SYSTEM

By means described elsewhere, the electronic computer can be made to accept programmes written in a simplified form, described below. In this form a programme of calculation consists of an ordered sequence of instructions arranged in a single column; the instructions are chosen from a permissible set and employ only the symbols given in Table 1. Ultimately the programme is presented to the machine in the form of a length of perforated paper tape which is scanned by a photo-electric tape reader, the input unit of the machine. The machine gives its results on a page printer.

Table 1

Letters	Figures	Letters	Figures
FS	FS	P	0
A	1	Q	>
B	2	R	⩾
C	*	S	3
D	4	T	*j*
E	(	U	5
F	)	V	6
G	7	W	/
H	8	X	⊗
I	≠	Y	9
J	=	Z	+
K	−	LS	LS
L	*v*	·	·
M	LF	?	*n*
N	SP	£	CR
O	,	†	†

The instructions mostly take the form of equations giving the new value of a computed quantity in terms of one or two previously calculated quantities or parameters. Instructions are also necessary, however, for selecting alternative courses of action, for printing results and for the input of further data. The instructions involve the following kinds of quantity:

(*a*) *Variables.*—The quantities or intermediate quantities which it is necessary to compute are denoted by *v*1, *v*2, *v*3, etc. The number of variables which can be introduced is for practical purposes unlimited: it is, in fact, about 5000. The range of magnitudes of a variable is virtually infinite; 2^p, where $2^{18} > p \geqslant -2^{18}$: the precision is limited to 11 decimal figures.

(*b*) *Constants.*—These are variables whose value is known in advance. The same restrictions of magnitude and precision apply as for variables. The numerical values are written in the form:

integral part, decimal point, fractional part.

Absolute standardization of form is unnecessary; thus, to six significant figures π may be written 3·141 59, 03·141 59, 03·141 590 and +03·141 590. All these and similar variations are accepted by the machine. Of the number 003·141 592 653 589 79, however, only the first 11 significant figures would be recorded inside the machine, i.e., the final 8 979 would be omitted. Negative numbers must be preceded by their sign.

(*c*) *Indices.*—In order to take advantage of the repetitive nature of calculation it is desirable to have some means of specifying any of the elements of a sequence, e.g. the components of a vector. For this purpose the notation *vn*1, *vn*2, *vn*3, etc., is introduced, where *n*1, *n*2, *n*3, etc., are indices restricted to integral values but otherwise computable like variables. The number of indices which can be introduced is limited to 18. The possible range of values of an index quantity is $2^{18} > n \geqslant -2^{18}$, which, of course, far exceeds its usefulness as an index proper. In writing down the numerical value of an index the decimal point may be omitted.

Those instructions which take the form of equations are given below [(i)–(iii)]. Permissible instructions are obtained from these

basic forms by replacing x, y, and z by the group of symbols denoting a variable, v, a combination, vn, or—except in (iii)—an index, n. In addition, x and y may also be replaced by constants.

(i) $z = x$.
(ii) $z = x\theta y$: θ is replaced by one of the symbols $+ - \otimes /$.
(iii) $z = Fm(x)$: The integer m refers to the function Table 2.

Table 2

1	$\sqrt{x}$
2	$\cos 2\pi x$
3	ε^x
4	$\log_\varepsilon x$
5	$\frac{1}{\pi}\arctan x$
6	$\lvert x\rvert$

Instructions are normally obeyed in the order in which they are written down, until a jump instruction is encountered. This may be conditional or unconditional and takes the form (iv) or (v) as follows:

(iv) jm means 'jump to instruction labelled m'.
(v) jm, $x\phi y$ means 'jump to instruction labelled m if $x\phi y$', where ϕ is replaced by one of the symbols $\geqslant > = \neq$.

The label is a positive whole number which is written immediately before (i.e. to the left of) the instruction concerned.

The remaining instructions are

(vi) The symbol * inserted before any of the instructions (i, ii, iii and viii) causes the computed value of z to be printed on a new line in the style described earlier, namely integral part, decimal point, fractional part, the latter to 10 decimal places. Insignificant zeros are suppressed. For a number outside the range 2 ± 38 the machine prints on the same line both the integral part and the fractional part of the number in the form $a.2^p$, where $\frac{1}{2} > |a| \geqslant \frac{1}{4}$, $2^{18} > p \geqslant -2^{18}$.
(vii) The letter H is a dummy stop instruction: the machine halts; it can be made to resume operation by pressing a key on the console.

As a simple exercise in coding, write down instructions for evaluating the sum of squares of $v1, v2, \ldots v100$. A suitable sequence is

$$\begin{aligned} n1 &= 1 \\ v101 &= 0 \\ 2v102 &= vn1 \otimes vn1 \\ v101 &= v101 + v102 \\ n1 &= n1 + 1 \\ j2,\ 100 &\geqslant n1. \end{aligned}$$

Such a group of instructions may form part of a larger programme involving the variables in question.

A complete programme of instructions and numbers is normally recorded inside the machine before being executed; it is therefore necessary to explain the input procedure.

The programme tape is prepared on a keyboard perforator on which are engraved the standard symbols. The symbols are punched in the conventional sequence, namely from left to right and down the column. Each instruction is followed by the symbols CR (carriage return) and LF (line feed). The keyboard is normally on 'figures', which means that capital letters such as F and H have to be preceded by LS (letter shift) and followed by FS (figure shift). 'Figure shift' corresponds to blank tape, and any number of blanks may separate the figure symbols proper, provided that the order is preserved. About 6 in of blank should be left at the head of the tape. Associated with the keyboard is a printer which gives a printed copy of whatever is punched; this should agree with the original manuscript.

As the programme tape is scanned the instructions are normally recorded in the store where they will ultimately be obeyed. The rate of scanning is approximately 2 sec per instruction, and the rate of execution about six instructions per second. When instructions are included between brackets each instruction is executed immediately after it has been read and is not recorded in the programme proper. This facility is used to start the programme simply by including an unconditional jump instruction between brackets, e.g. ($j1$), which means 'stop reading the tape and start obeying the programme at the instruction labelled 1'.

Once the programme has been started it may be necessary to call on the input medium for further numerical data. This may be done in two ways, for which it is necessary to introduce two further instructions, namely

(viii) $z = I$, which means 'replace z [which has the same significance as in (i) and (ii)] by the number formed by the next group of symbols on the tape'.
(ix) The letter T causes the machine to start reading further instructions from the tape, adding them to those already recorded in the store.

Instruction (viii) may be used to read a tape bearing numbers only.

The T instruction, combined with the 'bracket' facility, allows data to be input in the form (z = constant). This is a convenient method of altering parameters in between different runs. Thus, if the supplementary instructions take the form

$$\begin{aligned} (&v23 = 0{\cdot}012 \\ &v24 = 0{\cdot}965 \\ &n3 = 12 \\ &j1) \end{aligned}$$

the effect will be to repeat the run with these new values of the quantities $v23$, $v24$ and $n3$.

To illustrate the coding scheme described above, the following calculation is programmed.

It is required to compute

$$\alpha = \frac{1 - (p_2/p_1)^{2/7}}{1 - (p_2/p_3)^{2/7}}, \quad \text{where } p_2 = 30 \text{ and } p_3 = 40$$

$$r_e = 1{\cdot}3 + 2\left[1 - \frac{\alpha^{3/2}(p_2/p_3)^{2/7}}{(p_2/p_1)^{2/7}}\right]$$

$$r_m = 1{\cdot}3 + 2\left[1 - \frac{\alpha^{1/2}(p_2/p_3)^{2/7}}{(p_2/p_1)^{2/7}}\right]$$

for values of p_1 of the form $40 - \frac{10n}{156{\cdot}4}$, where $n = 0(1)156$.

The programme is given below.

$2v1 = 40$	$(p_1 = 40)$
$v2 = 30$	$(p_2 = 30)$
$v3 = 40$	$(p_3 = 40)$
$v4 = 10/156{\cdot}4$	
$v5 = v2/v3$	(p_2/p_3)
$v6 = \mathrm{F4}(v5)$	
$v7 = 2/7$	
$v8 = v7 \otimes v6$	
$v9 = \mathrm{F3}(v8)$	$(p_2/p_3)^{2/7}$
$1v10 = v2/v1$	
$v11 = \mathrm{F4}(v10)$	
$v12 = v7 \otimes v11$	
$v13 = \mathrm{F3}(v12)$	$(p_2/p_1)^{2/7}$
$v14 = 1 - v13$	
$v15 = 1 - v9$	
$*v16 = v14/v15$	forms and prints α
$v17 = \mathrm{F1}(v16)$	

$v18 = v17 \otimes v9$	
$v19 = v18/v13$	
$v20 = v16 \otimes v19$	
$v21 = 1 - v20$	
$v22 = 2 \otimes v21$	
$*v23 = 1 \cdot 3 + v22$	forms and prints r_e
$v24 = 1 - v19$	
$v25 = 2 \otimes v24$	
$*v26 = 1 \cdot 3 + v25$	forms and prints r_m
$v1 = v1 - v4$	adjusts p_1
$j1, v1 > 29$	tests for last cycle
H	halt
$(j2)$	starts programme.

The following notes are of interest:

(*a*) No attempt has been made to economize on the use of variables. Thus, instead of $v9 = F3(v8)$, one could write $v8 = F3(v8)$, since the argument is no longer needed. However, nothing is gained by so doing, since space is virtually infinite for problems of this kind.

(*b*) A further example of laziness is the means adopted for evaluating 2/7. Instead of evaluating the fractions we have simply included an instruction for doing so, namely $v7 = 2/7$.

(*c*) The value of any intermediate quantity can be printed simply by inserting an * before the appropriate instruction. This may be useful in locating mistakes; the * can afterwards be erased by turning it into a †.

(*d*) The maximum number of instructions allowed cannot be given very precisely, but a safe estimate is 500, which, in view of their comprehensive nature, should be adequate for the class of problems envisaged.