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

III. ADAPTATION OF ROUTINES FOR ELABORATE ARITHMETICAL OPERATIONS

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

[*Manuscript received November* 12, 1953]

Summary

Certain routines, using the fixed index programming methods established for use in the C.S.I.R.O. Mark I computer, are described. These facilitate the use of the machine for performing elaborate arithmetical operations required for extended accuracy, floating index and complex variable arithmetic. Programming for such operations is greatly simplified by the use of an " interpretive " code, especially chosen by the programmer. The " interpretive " method of programme design is potentially very powerful, and relieves the programmer of the work associated with the details of store positioning, index control, scale changes, etc. The various operations called into use by the specially chosen codes are closely analogous to the operations existing in the normal machine code, although some are specially chosen to facilitate the use of variable commands and in transfers of control by linkage and in control of repetitions of routines which are stored in the machine in the special code.

I. INTRODUCTION

The C.S.I.R.O. Mark I computer is of the high speed fully automatic type which is organized and sequenced by a suitable programme of commands. The machine is capable of performing a number of elementary functions, multiplication with fixed index point, and certain logical functions. Each command corresponds to one of these elementary functions. Normally programmes use the system of library routines and sub-routines, which themselves consist of sequences of commands for organizing such operations as division, square rooting, evaluation of functions, and so on.

In Parts I and II (Pearcey and Hill 1953*a*, 1953*b*) the authors described the conventions used in programme design and showed how a programme is compiled and recorded in a manner suitable for use by the computer. The discussion was restricted to cases of single word numbers, i.e. numbers occupying only one storage location, and to cases of fixed index point.

This part discusses the extension of the library routine system to more elaborate arithmetical systems; to floating point, multiple precision, and complex arithmetic, and certain combinations of these. The use of routines providing arithmetical operations of these kinds is greatly facilitated by use of a very flexible method of programme organization termed " interpretive " (Wilkes, Wheeler, and Gill 1951), and a full description of the application of this method to the C.S.I.R.O. Mark I computer will be given.

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

The overall effect of use of interpretive methods for performing elaborate arithmetical operations is to provide the machine with a number of additional useful functions each of which may be called into use individually by single special commands. The code system used for recording commands in such programmes differs from that used for normal commands used by the machine, and can be chosen at will to suit the system of routines in use.

II. Arithmetical Systems

Most automatic computers like the C.S.I.R.O. Mark I operate in the fixed index convention, that is, the location of the binary point is fixed in relation to the digits of numbers which are restricted to a definite range of values, usually between $+1$ and -1. In such a scheme it is often difficult to keep the range of magnitude of variables within the storage capacity of registers. Variables may require to be scaled down initially, thus losing significant digits. Alternatively, suitable changes in the position of the index point may be " programmed " ; that is, routines are inserted where needed which cause variables to be scaled up or down during the course of the calculation. Both these expedients are inconvenient and call for considerable effort from the programmer.

Additional difficulty may arise in programming for calculations involving large sequences of arithmetical operations and those involving complex numbers. In the latter case the number of problem variables is at least doubled and organization of the calculation correspondingly increased.

In some types of calculation it is impracticable to predict the range of variables at any particular stage of the calculation. This occurs in evaluation of large determinants, transformation of matrices, and evaluation of roots of algebraic, logarithmic, and transcendental functions. In such calculations a semi-logarithmic form of recording data is most useful. Each datum is recorded in two parts ; a fractional component, X say, and an integral component, p. Two scale conventions are commonly in use, the decimal and binary index scales. In the first of these the components are related by a convention which represents the number $X \cdot 10^p$ $(0 \cdot 1 \leqslant |X| < 1\,;\ -N \leqslant p < N)$.* In automatic computers decimal index scale is commonly used but binary index scale is sometimes more convenient when the greatest precision from a given number of binary digits is required. In the latter case, the number represented would be $X \cdot 2^p$ $(0 \cdot 5 \leqslant |X| < 1\,;\ -M \leqslant p < M)$.*

The semi-logarithmic or " floating index " method of storage is quite suitable when the number of multiplications, divisions, square rooting, etc. is comparable with, or greater than, the number of additions and subtractions. The method fails in subtraction of nearly equal quantities, as also occurs in fixed index scales. Significant figures are lost, and can be retained only by holding additional figures in the fractional components throughout the calculation. Conditions can arise in which the result is dependent upon the order in which operations are performed. Such conditions occur in matrix operations, particularly with

* N and M are some integers determined by the maximum capacity of the register storing p.

sets of ill-conditioned coefficients. The floating index method fails also in summation of a large number of items of the same order of magnitude. In numerical integration a stage in the summation may occur at which the partial sum is so great that following increments lie outside the precision of the fractional component, and further increments will not affect the total. The alternative is to perform the addition by a fixed index method and allow the accumulator to " over-carry ", account being taken of the digits carried over. This amounts to retaining additional figures.

Floating index arithmetic is therefore not a cure-all for arithmetical difficulties, but must be used with care. Where suitable, it can save much work in programme design and avoids nearly all preliminary scaling adjustments. In case of inadequate accuracy, additional figures must be retained. This can be done in a computer only by allowing the digits of a number to extend beyond one storage location into one or more other locations. Such methods are known as " multiple precision " methods ; the commonest being the " double precision " method, in which just two storage locations are used to hold each number. In many cases double precision is sufficient to overcome most of the difficulties mentioned and also to reduce considerably the effort required in programming and in scale adjustments.

In certain cases both the multiple precision and floating index methods may be used together ; part of the calculation being carried out by one method and part by the other, or even by using the floating index method together with double or extended precision for fractional components.

These methods may be extended to deal with complex variables. The additional complication is that of organizing the relationships between the two parts of each variable. Arithmetical systems which it may be necessary to use may be listed as follows.

(i) *Fixed Index Methods.*—These may be of single or multiple precision, for real or complex variables or both. Some scale change and progressive adjustments during calculation sometimes cannot be avoided, and changes of index must be specially programmed.

(ii) *Floating Index Methods.*—These may be of single or multiple precision and for real or complex variables or both. No scale changes or index variations are needed nor require to be specially programmed, but the method must be used with care under certain circumstances to avoid loss of significant digits.

III. Function Blocks

For each arithmetical method a set of routines may be constructed for organizing arithmetical and other functions peculiar to the methods adopted. Such a set of routines, known as a " function block ", consists of two parts ; the " arithmetical block ", which provides for addition, subtraction, multiplication, square rooting, and so on, and the " organizational block ", which contains routines for organization of programmes using the function block.

(*a*) *Arithmetical Function Blocks*

Frequently the basic function of any arithmetical block is that of addition. Around this function all the others may be built. Thus, subtraction uses addition; multiplication, particularly in complex and multiple precision methods, uses addition; division uses both multiplication and addition; and so on. Consequently an arithmetical function block consists of a number of interrelated routines of various orders, linkage being made via registers D_{15}, D_{14}, etc.

A number of function blocks have been constructed and are used in programmes on the C.S.I.R.O. Mark I computer. Each system is designed according to certain conventions which define their manner of use. These are:

(i) *Floating Index Arithmetic.*—In this system numbers are stored in two parts, a fractional part, X say, and an integral part, p. X occupies one store location, i.e. 19 binary digits and one sign digit, while p occupies only 10 digits, including its sign digit, of a second storage location, usually the next higher to that holding X. Negative numbers are represented by the complement of the fractional part. There are two scales used, the decimal and binary.

(1) In the decimal system X is stored in binary form and is restricted to the range $1\cdot0>|X|\geqslant0\cdot1$, whilst p, in p_{11} units, is restricted to the range $512>p\geqslant-512$. The number pair X,p thus represents $X\cdot10^p$.

(2) In the binary system X is stored in binary form and is restricted to the range $1\cdot0>|X|\geqslant0\cdot5$, whilst p, in p_{11} units, is restricted to the range $512>p\geqslant-512$. The number pair X,p thus represents $X\cdot2^p$.

Values of X satisfying the appropriate conditions are said to be "normalized".

In the addition of two numbers, say $X\cdot10^p$ and $Y\cdot10^q$, the lower index, say q, is raised to equal the greater, the fractional part being correspondingly reduced to $Y\cdot10^{q-p}$. The sum $X+Y\cdot10^{q-p}$ is formed. If this exceeds unity in modulus it is multiplied by 10^{-1} and the index of the sum is changed to $p+1$. The sum is stored as one or other of the pairs

$$X+Y\cdot10^{q-p},\ p,$$
$$(X+Y\cdot10^{q-p})10^{-1},\ p+1.$$

Multiplication is performed by forming the product XY and the sum $p+q$. The fractional part may lie between $0\cdot1$ and $0\cdot01$, in the decimal case, and, if so, is multiplied by 10 and the index of the product changed to $p+q-1$. In the case of floating binary indices, multiplications by 10 or 10^{-1} are replaced by multiplications by 2 and 2^{-1} respectively to keep the fractional part of the product between 1 and $0\cdot5$ in modulus.

Iterative processes are used for division, square rooting, etc.; the results always being provided in the normalized condition in accordance with conventions of storage.

From a purely arithmetical point of view the floating binary method is the better since changes of the index correspond to multiplication by 2 or 2^{-1}, as against 10 or 10^{-1} in the decimal method ; the smaller factor assists in maintaining greatest possible accuracy from a given number of digits in the fractional part.

Simpler organization of left and right shifts rather than multiplications by 10 and 10^{-1} leads to a saving of about 10 per cent. of commands in arithmetical routines for floating binary compared with those for floating decimal scales. The binary index scale also provides somewhat faster operation. However, users find visual interpretation easier for floating decimal numbers than for floating binary numbers. It is therefore preferable to have data punched and printed in floating decimal form. Conversion to and from floating decimal form in input and output requires additional routines in the floating binary function block ; which more than compensates for the 10 per cent. saving in the arithmetical routines.

The function block for single word floating-index real-variable arithmetic is relatively simple, involving little interconnection between component routines performing the various operations. Thus the routine for subtraction uses addition, but that for multiplication does not, and the division routine is self-contained. The square root routine uses multiplication. All but division also use a " normalization routine " which converts results to the conventional form for storage.

In calculations which involve many separate groups of data, such as coefficients of matrices which cannot be inspected easily, a special floating decimal function block is used. In this a tally is kept of changes in the number of figures which are significant in the associated number. Thus, if a sum or difference is multiplied by 10^s $(s>0)$ to bring it to the normalized form, the number of significant figures is reduced by s, and so on for other operations. The tally of such changes is held as an integer in p_1 units in the p_1–p_{10} digit group of the word holding the index. Results can then be given a degree of significance automatically.

(ii) *Multiple Precision Arithmetic.*—In this system a number is divided into groups of 19 binary digits. The number of such groups may be made as great as desired, and a number occupying " n " groups is called an " n-fold " number, and the arithmetical system used with such numbers is called n-fold arithmetic. The most frequently used is the " double precision " case, for which $n=2$. The storage conventions adopted are that an n-fold number is stored in groups of 19 digits in adjacent successive storage locations, the most significant digit position in each location being zero, except in that holding the most significant group, in which it plays the role of a sign digit. A negative n-fold number is stored as a complement of the corresponding n-fold positive number.

Arithmetical blocks are designed in accordance with the convention that n-fold numbers are of unit magnitude or less, the index point being immediately to the right of the most significant p_{20} digit stored.

Two function blocks have been constructed, one for double precision and one for n-fold arithmetic, where n may be specified by the programmer. The latter system finds application in number theory computations.

(1) *Double Precision.* In this method numbers are stored modulo 2 in integral multiples of 2^{-38}, providing an equivalent accuracy of about 12 decimal digits. When addition is performed, digit groups of corresponding significance are added together starting with those of lower significance. Any over-carry arising from summation of the less significant components is detected and added as a unit to the sum of the more significant components. The sum is finally stored in the standard form.

The function block provides the functions of addition, subtraction, multiplication, division, and square rooting. Each function routine uses the routine for addition, and the last two use also multiplication, which itself uses addition. A product is always provided in the standard form as a group of *four* 19-digit words, the two least significant words being retained in case of need in the calculation. This allows double precision numbers to be treated, by suitable programming, as integers instead of fractions.

The organization of this arithmetical block is more complicated than that for the floating index method since there is a larger proportion of high order routines.

(2) n-*Fold Arithmetic.* In this case n may be specified by the user, but only the functions of addition, subtraction, and multiplication are provided. The organization is correspondingly more complicated than that for the double precision case. Its manner of use differs from that of other systems and will be discussed later.

(iii) *Complex Variable Arithmetic.*—The complications caused in the construction of programmes to deal with the arithmetic of the complex variable justify the use of a function block even in the case of single-fold or one-word arithmetic. This has been done for use on the Mark I computer in a special case which was found to be most frequently used. This case adopts the convention that the real and imaginary parts of a complex number are stored in two adjacent storage locations. The index points are considered fixed and to lie between the p_{10} and p_{11} digit positions in both components and *not* between the p_{19} and p_{20} digit positions. Negative components are represented by the complement of the corresponding positive number. Each component of a number may take values equal to all integral multiples of 2^{-10} from -512 to $512-2^{-10}$ and all the routines of the function block preserve this convention.

Routines contained in the function block include addition, subtraction, multiplication, division, and square rooting of complex numbers, together with other functions which apply only in the case of the complex variable, such as the evaluation of the modulus and the conjugate of a complex number.

Function blocks are also available for use of the complex variable in :

(1) floating index arithmetic,

(2) double precision arithmetic.

In each case the same conventions apply to both components as apply also to the corresponding case of the real variable. One exception is the floating index system where only three adjacent storage locations are used instead of four. The first and third hold the real and imaginary fractions, respectively, the second or centre location holds the indices of both components, that of the real in the p_{11}–p_{20} positions and that of the imaginary in the p_1–p_{10} positions. Whenever such a number is taken from store the indices are " unpacked " before the arithmetical operation is performed. When placed into store the indices are " repacked ".

In all cases routines for the complex variable are based upon routines in the corresponding function block for the real variable. Thus complex floating addition uses the routine for real floating addition, complex floating multiplication uses real floating multiplication and real addition, and so on, and all routines for complex operations are of higher order than the routines for real operations.

Further, since it frequently happens in calculations involving the complex variable that operations with real variables are also needed, the " real " operations are also provided in addition to the additional functions of modulus and conjugate, etc. which are peculiar to the complex variable.

The principles of the construction of a block for n-fold complex arithmetic would be similar to those already constructed.

(iv) *Other Systems.*—The most likely useful system in addition to those described above is the floating double precision system and its counterpart for use with the complex variable. In this the conventions of both the floating index and double precision methods are combined. Each number occupies two adjacent storage locations for its fractional part and its index lies in the p_{11}–p_{20} positions in the following location. A complex number occupies five locations, the first two for the real and the last two for the imaginary fractions, the indices being packed into the third location. The function blocks for these systems are built in direct analogy to those for the other systems. Both function blocks are necessarily large and occupy much storage space, and also are slower in operation since in effect all the operations of both the floating index and double precision systems must be performed.

(*b*) *Organizational Function Blocks*

Into the organizational part of the various function blocks are placed routines for taking fresh data from punched tape, page printing, or punching results on to tape ; and for withdrawing numbers from and placing numbers into store in standard form appropriate to the system used. For purposes of organizing the problem programme certain other operations are also included. These are analogous to the control shifts and counts which occur in normal routines and will be illustrated in more detail later. Of these operations only the input and output functions use the arithmetical block. Input tape for floating index arithmetic is punched with the fraction preceding the index. A designation follows each component to denote its sign. Decimal digits are received from tape and converted by the input routine of the function block and assembled into the equivalent binary form and the result is stored. For printing, the fractional part of a number is printed as six decimal digits with sign followed by the index printed as a three-digit decimal integer with sign. The input or output operations read or print data in standard form appropriate to the function block used. In the case of the complex variable, the real part always precedes the imaginary part.

The manner of placing into and taking from store is standardized according to the system's conventions. Thus, routines take and place the various parts of a number from and into successive storage locations in the correct standard order, doing the packing or unpacking of parts as required.

IV. Use of Function Blocks

The most straightforward way of using a function block is to call the various function routines into use by the standard link and cue method described in Part II (Pearcey and Hill 1953*b*). A problem programme would then consist largely of " control commands ", those which affect the sequence register and cause control to be stored or shifted, together with relatively few commands for operations which may not be provided by the function block and by other routines included into the programme.

This method suffers from the disadvantage that the link number must frequently be restored as routines of fresh orders are called into use, and that the withdrawal and placing into store of numbers in standard form must be made with reference to an address which must, each time, be previously placed into a special location. These factors are confusing for the programmer ; they make it more difficult to trace through the sequence of operations in a programme and draw his attention away from the fact that the function block really supplies the machine with what in effect is a new set of " wired-in " operations.

One way of helping the programmer is to provide a " directory " with the functions blocks. This consists essentially of a list of numbered addresses placed in sequence in the store ; each address corresponds to the place in the function block, to which control must be transferred in order to obtain the operation which the particular directory number corresponds to.

The use of directory numbers for calling functions in the block implies the use of special operation codes, not identical with the machine code.

This development leads directly to the " interpretive system " of programming, a method which greatly reduces effort needed to compile programmes.

V. The Interpretive System

Each function called via the directory may be coded into a programme in terms of the serial number of the directory position which causes control to be transferred to the required function routine, and a special routine is needed for interpreting those parts of a programme which are recorded in the special code.

The particular code adopted for interpretation is at the choice of the programmer ; and is not necessarily restricted to any particular address system.

Each coded word or partitioned group of digits may be regarded as a command of a special kind, the meaning of which may change from problem to problem and is not directly connected with the actual mode of operation of the machine. The special decoding or " interpretation routine " may be designed at the will of the programmer in accordance with his chosen code system.

The simplest way of using a list of coded data in a programme is to arrange the interpretation routine so as to select and interpret them one by one in order of storage location. The list of coded data may thus be considered as a routine in its own right. Many of the techniques used in the design of normal routines may be taken over into the design of routines to be interpreted and a strong parallelism can be drawn between normal techniques and interpretation techniques. Terms which are used in normal programming are frequently adopted with similar meaning in the interpretive method and are prefixed by the term " hyper- ". Thus the list of programme data for interpretation is known as a hyper-routine or hyper-programme and the code adopted is called the hyper-code; the function corresponding to any hyper-code number is a hyper-function. A single datum of a hyper-programme is called a hyper-command. If the datum to be currently interpreted is stored in location n, then hyper-control is said to be at n.

Similarly, some of the registers existing in the computer, particularly the accumulator A, have their equivalent in the interpretive system. These are known as hyper-registers, for example, hyper-accumulator, hyper-sequence register, hyper-interpreter, etc. They must not be confused with the accumulator, sequence register, or interpreter proper. Thus the physical location of the hyper-accumulator depends upon the function block used, and may in fact be one or more of the registers of D or some locations of the store.

VI. The Hyper-code

Many hyper-code systems are possible and the programmer may design whichever suits his purpose. In most interpretive systems it is convenient to code a single hyper-command into a single location of the store, although if additional digits are needed two or more adjacent locations in the store may be used to hold a single hyper-command; or more than one hyper-command, if requiring few digits each, may be packed two or more together into single locations. The digits of each hyper-command are partitioned into groups each possessing a meaning depending upon the structure of the interpretation routine. The parallelism between normal, or routine, commands and hyper-commands is extended to the address conventions of the hyper-code, which may be of the one-address or three-address type and so on. A one-address hyper-code is that most commonly used with the C.S.I.R.O. Mark I computer.

A typical one-address hyper-command of one word consists essentially of a numerical address of 10 digits occupying the digit positions p_{11}–p_{20} and a function address of five digits in the positions p_6–p_{10}. The digit group p_1–p_5 in hyper-commands corresponds to the machine code for destination Z, or number 20, a destination which in the machine code has been left unused. Any machine command possessing this destination has no effect except in the case of the $c(A)$ source, when register A becomes cleared to zero.

The destination Z, or 20, distinguishes those hyper-commands which call hyper-functions from the function block from those which do not. It is not always convenient to construct hyper-routines using only the operations available in the function block ; occasionally quite simple operations are required for which it is not economical to provide store space for a short routine in the function block. Hyper-commands not terminated with the Z code letter are coded as machine commands and the interpretation routine deals with them as though they were machine commands. In this way, what are in effect machine commands may be entered into hyper-routines. Such commands are true hyper-commands but do not call the function block into use.

The numerical address of a Z-terminated hyper-command in the p_{11}–p_{20} digit group, possesses the same significance as it does in a machine command. The function address in Z-terminated hyper-commands is a serial number, from 0 to 31, providing space for 32 possible functions, some of which are essentially arithmetical and others organizational. This code number is selected from the hyper-command by the interpretation routine and used as a reference number to the directory, which is contained in the interpretation routine and which in turn directs control to the required position in the function block.

VII. p-Word Numbers

In the conventions of many function blocks a datum occupies more than one word of the store. Data may occupy different numbers of locations in the store for the same function block. Thus for complex arithmetic complex numbers require more words than real numbers in the same problem. For convenience the multi-word locations n, $n+1$, $n+2$, . . ., $n+p-1$ are referred to as " p-length location n " and symbolized as n_p.

For programmes using real and complex variables, two word lengths will occur ; p-length words for real numbers and q-length words for complex numbers. However, by incorporation of special features into the function blocks the programmer is relieved of any detailed concern with such " mixed " word lengths.

VIII. Hyper-registers

By analogy with the registers used in normal programming the hyper-registers serve equivalent purposes in hyper-programmes. The hyper-address code is of the one-address type, there is only one arithmetical hyper-register known as the hyper-accumulator. It is denoted by $\bar{A}$. The hyper-sequence register holds the store address of the next hyper-command to be adopted and is denoted by $\bar{S}$, and the hyper-interpreter, denoted by $\bar{K}$, holds the numerical address of the current hyper-command. A further hyper-register known as the " transfer register ", denoted by X, is used for transfers of numbers between store and the accumulator. A further hyper-register, which, like X, has no parallel in normal programming, is the hyper-link register, written $\bar{L}$. The physical position of these registers varies with the function block used. As an example, those for the floating decimal index system are listed below :

Hyper-accumulator	$\bar{A}=D_{10}D_{11}$ for fraction and index respectively
Transfer register	$X=A, B$ for fraction and index respectively
Hyper-sequence register	$\bar{S}=D_{13}$
Hyper-interpreter	$\bar{K}=D_{12}$
Hyper-link register	$\bar{L}$=three spaces in the function block
Normal link-stores	$=D_{14}$, D_{15}, used for links into and between function block routines
Stores for constant parameters in function blocks	$=D_8$, D_9, etc.
Stores for count parameters in hyper-programmes	$=D_0$, D_1, . . ., D_7

The advantage gained by placing hyper-registers into D whenever possible is the ease with which the operator may see, at a glance at the monitor tube faces, at which stage the machine is in the hyper-programme. With the more complicated function blocks it is necessary to extend the space occupied by hyper-registers into the function block itself.

IX. Hyper-control and Hyper-routine Linkages

Most of the techniques used in normal programme design are applied also to the design of hyper-programmes. However, some special functions are provided in the function blocks which simplify somewhat the design of hyper-programmes, and have no direct equivalent in normal programming methods.

Such is the case of those hyper-commands devoted to control of the hyper-sequence and linkage to and from hyper-routines. The hyper-registers involved in sequence control are known as : the hyper-interpreter $\bar{K}$, the hyper-sequence register $\bar{S}$, and the hyper-link register $\bar{L}$. The last of these comprises a group of consecutive storage locations and may be called $\bar{L}_1$, $\bar{L}_2$, $\bar{L}_3$, etc.

When control enters the interpretation routine hyper-control advances serially from the address held in $\bar{S}$ at the stage of entry. During each cycle of the interpretation routine a p_{11} unit is added to $\bar{S}$, e.g. to D_{13}.

A change of hyper-control to " n " is called by a hyper-command written as

$$n\rightarrow\bar{S}$$

coded as

$$n; m. Z,$$

where m is the address in the directory which calls the appropriate function in the function block. This function places the content of $\bar{K}$ (or D_{12}) into $\bar{S}$ (or D_{13}).

In this sense $\bar{K}$ and $\bar{S}$ are respectively analogous to real interpreter and sequence registers. Addition into $\bar{S}$ is not provided.

Transfer of hyper-control from one hyper-routine to another and back uses the hyper-link register $\bar{L}$. The hyper-command $n\rightarrow\bar{L}$ at location m, causes hyper-control to be transferred to n, and the number $(m+1)p_{11}$ is stored in $\bar{L}_1$ whilst the previous content of $\bar{L}_1$ is placed in $\bar{L}_2$, and that of $\bar{L}_2$ goes to $\bar{L}_3$, and so on. Such a hyper-command corresponds to the sequence

$$(S)\rightarrow D_{15},$$
$$n\rightarrow S$$

in normal programming.

A return of hyper-control, to $m+1$, that following the place from which hyper-control was transferred, is obtained by a hyper-command written as $(\bar{L})\rightarrow\bar{S}$.

This causes $(\bar{L}_1)$ to be placed in $\bar{S}$, $(\bar{L}_2)$ to be transferred to $\bar{L}_1$, $(\bar{L}_3)$ to go to $\bar{L}_2$, and so on. This is analogous to the sequence

$$p_{11}\overset{+}{\rightarrow}D_{15},$$
$$(D_{15})\rightarrow S$$

in normal programming.

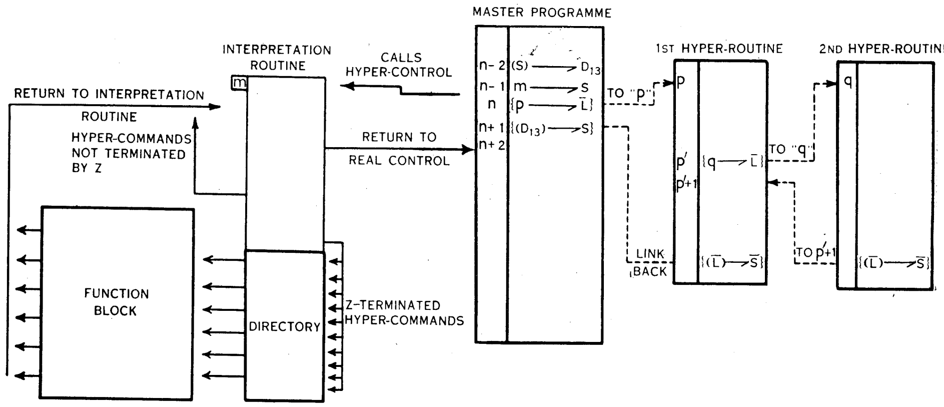

Fig. 1.—Transfers of real control and hyper-control.
Passage of hyper-control $--\rightarrow$
Passage of real control $\longrightarrow$

Hyper-control is called into use by transferring control direct to the interpretation routine in the usual way, that is, by direct transfer of real control to the head of the interpretation routine. To cease hyper-control and achieve return of real control to the main programme a link datum must have been stored in D_{13}; return to real control is attained by a command $(D_{13})\rightarrow S$ in the hyper-routine; a hyper-command not terminated by Z and therefore possessing the significance of a machine command.

For example, consider the case of transfer from real control at n to hyper-control at p, which in turn calls a second hyper-routine at q. Let the interpretation routine be entered at m. The scheme achieving this is illustrated in Figure 1, where transfers of real control are shown by full lines, and those of hyper-control by broken lines.

In the figure hyper-commands are distinguished from real commands by surrounding all the former by braces thus: {. . .}. Hyper-commands not terminated in Z are also distinguished in this manner.

X. Hyper-routine Loops

As in normal programme techniques, so also in hyper-programmes full use is made of the method of controlling repetitions of groups of hyper-commands by a counting and sign-testing process.

Two functions provided in the organizational part of the function block are used for this purpose. The first is written thus

$$n_p \rightarrow D_r,$$

the numerical part of which consists of the partitioned number $8n+r$, where $0 \leqslant r < 8$ and $0 \leqslant n < 64$; the second function is represented by

$$n_p \xrightarrow{-} D_r,$$

where n and r are coded in the same manner.

The first function places the number $np.p_{11}$ units into the upper half of register D_r, whilst the second subtracts $np.p_{11}$ from (D_r), tests the difference resulting and, if negative, increases $\bar{S}$ by an additional unit p_{11}, otherwise not.

The first of these hyper-commands is used to set the number of repetitions of the programme loop to which it refers, the latter combines the functions of counting repetitions and causing a conditional shift of hyper-control.

XI. Variable Hyper-commands

A special function is provided which simplifies the use of variable commands. Such a requirement is frequently associated with the loop control functions just described. This function causes a parameter to be transferred to $\overline{K}$, and to be held there so that the numerical part of next hyper-command which follows is added into it. This function is denoted by the symbol

$$(D_r) \xrightarrow{+} \overline{K}, \text{ where } 0 \leqslant r < 8.$$

If (D_r) is equal to $np.p_{11}$, the numerical address of the following hyper-command is effectively increased by $np.p_{11}$.

When an operation in a loop of a hyper-routine has its numerical address progressively changed, e.g. an address referring to serial hyper-locations in the store, the address is frequently used also as a counter for loop control. Thus the loop control commands also serve the purpose of assisting in the use of variable commands. In this case it is normally required to vary the address by multiples of p for p-length words and by multiples of q for q-length words in complex operations. Thus two sets of loop control operations are provided for complex variable hyper-programmes

$$n_p \rightarrow D_r, \qquad n_p \xrightarrow{-} D_r,$$
$$n_q \rightarrow D_r, \qquad n_q \xrightarrow{-} D_r,$$

having the effects previously described ; the second set being used for loops involving addresses for complex operations. For both p and q systems, the same operation $(D_r) \xrightarrow{+} K$ can be used where required.

Functions of the type $(D_r) \xrightarrow{+} K$ may not be placed adjacent to one another in a hyper-routine.

XII. Transfers to and from Store

One hyper-function transfers data to the hyper-accumulator and another transfers from the hyper-accumulator to the store. The former of these is written as

$$(n_p)\rightarrow\bar{A},$$

where n_p is the store location referred to ; and similarly the second is written

$$(\bar{A})\rightarrow n_p.$$

Closely associated with these are the two most elementary arithmetical functions involving transfer from the store to the hyper-accumulator. These are

$$(n_p)\overset{+}{\rightarrow}\bar{A},$$

which adds the content of " hyper-location n " into the hyper-accumulator and

$$(n_p)\overset{-}{\rightarrow}\bar{A},$$

which subtracts the content of hyper-location n into the hyper-accumulator and so on.

In cases of complex arithmetic equivalent q-length operations are supplied for transfers of q-word complex variables etc. Thus the operations provided would include

$$(n_q)\rightarrow\bar{A}\ ;\ (\bar{A})\rightarrow n_q\ ;\ (n_q)\overset{+}{\rightarrow}\bar{A}\ ;\ (n_q)\overset{-}{\rightarrow}\bar{A}.$$

Table 1 illustrates use of loop control and command variation operations and of braces, { }, to distinguish hyper-commands from machine commands. This routine accepts numbers from punched tape and stores them in sequential p-word store locations.

Table 1

USE OF LOOP CONTROL, COMMAND VARIATION OPERATIONS, AND BRACES

Location	Hyper-command	Operation
0	$\{n-1_p\rightarrow D_r\}$	Sets $(n-1)p.p_{11}$ in D_r
1	$\rightarrow\{(D_r)\ \overset{+}{\rightarrow}\bar{K}\}$	Adds $(n-s-1)p.p_{11}$ to next command
2	$\{(\bar{I})\ \rightarrow m\}$	Stores input number in p-word starting at $m+p(n-s-1)$
3	$\{1_p\ \overset{-}{\rightarrow}D_r\}-$	Reduces (D_r) by $p.p_{11}$ and tests the sign of (D_r)
4	$-\{1\ \rightarrow\bar{S}\}$	Return hyper-control to 1
5	$\{(\bar{L})\ \rightarrow\bar{S}\}\leftarrow$	Link to superior routine when $(D_r)<0$.

This sequence causes the variable hyper-command at 2 to be adopted as $\{(I)\rightarrow\overline{n-s-1}.p+m\}$ on the $(s+1)$th cycle of the loop of n cycles.

XIII. Additional Hyper-functions

Additional hyper-functions may be provided simply by placing appropriate routines, additional to the function block, into the store. These are performed by a special hyper-command coded as n; 15,Z which transfers control to the head location of the additional routine at location n. Since the address digits of such a hyper-command are used in specifying n, additional hyper-functions cannot refer to store addresses of data and are frequently of the type written as

$$\{f(\bar{A}) \rightarrow \bar{A}\},$$

where $f(\bar{A})$ may be $\log(\bar{A})$, $\exp(\bar{A})$, $(\bar{A})^{1/3}$, etc.

Similar hyper-functions are already included in the normal function block, e.g. $(\bar{A}) \rightarrow 0t$, $(I) \rightarrow \bar{A}$, etc.

XIV. The Standard Hyper-code

The hyper-code has been standardized so as to apply to most of the function blocks so far designed; the same code calls the same functions whether the floating decimal complex or double length complex system is used. Variations occur in the additional functions called by the hyper-command described in the previous section.

In the present scheme, the p_5–p_1 digits are used to distinguish between hyper-commands which use the function block and those which do not. For hyper-commands in which this group is not Z, the conventions of normal machine operations apply as outlined in Parts I and II (Pearcey and Hill 1953*a*, 1953*b*).

The earliest 16 of the 32 code numbers in Z-terminated hyper-commands refer to arithmetical functions and transfers of real variables, and organizational hyper-commands, whilst some of the remaining 16 call the hyper-functions relating to the complex variable.

In calculations involving only real variables the latter group of hyper-functions and its corresponding function block are not used. The first part only of the standard code is used. This, and the fact that the code applies equally well to a number of systems, means that the programmer only rarely needs to take into account his arithmetical system whilst programming.

The hyper-code used is shown in Table 2. It includes the hyper-functions for the complex variable. The first six hyper-commands 0–5 provide transfer to $\bar{A}$ and arithmetical functions of the real variable, the appropriate word length suffix being included (here written as p). These functions include addition, subtraction, multiplication (product of $(\bar{A})$ by (n_p)), and square rooting. The next three, 6–8, provide input, output, and transfer from $\bar{A}$ to store; the following three, 9–11, provide sequence control operations; the next three, 12–14, refer to loop control. Number 15 calls any additional hyper-functions.

The arithmetical functions for the complex variable, called by code numbers 16–21, are similar to those for the real variable called by 0–5. Input and output of the complex variable are provided by 22–24, and functions peculiar to the complex variable and those involving both real and complex variables together are given the highest code numbers. These include such operations as multiplication and division of a complex by a real quantity, evaluation of the modulus and conjugate of a complex number, and so on.

Each function block can provide different groups of possible additional functions without change to the block. These depend upon the nature of the

TABLE 2

THE STANDARD HYPER-CODE

Code	Hyper-command Symbol	Meaning
n: 0, Z	$(n_p) \rightarrow \bar{A}$	Transfers hyper-word (n) to $(n+p—1)$ to $\bar{A}$
n: 1, Z	$(n_p) \xrightarrow{+} \bar{A}$	Adds content of hyper-word (n) to $(n+p—1)$ to $(\bar{A})$
n: 2, Z	$(n_p) \xrightarrow{-} \bar{A}$	Subtracts content of hyper-word (n) to $(n+p—1)$ to $(\bar{A})$
n: 3, Z	$(n_p) \xrightarrow{\times} \bar{A}$	Replaces $(\bar{A})$ by its product with hyper-word (n) to $(n+p—1)$ (fixed index systems hold less significant part of product in register $\bar{B}$)
n: 4, Z	$(n_p) \xrightarrow{\div} \bar{A}$	Replaces $(\bar{A})$ by its quotient with regard to (n), $(n+1)$, . . ., $(n+p—1)$
n: 5, Z	$(n_p)^{\frac{1}{2}} \rightarrow \bar{A}$	Places square root of hyper-word (n) to $(n+p—1)$ in $\bar{A}$
n: 6, Z	$(n_p) \rightarrow \bar{0}$	Print hyper-word in (n) to $(n+p—1)$ in standard form
n: 7, Z	$(\bar{I}) \rightarrow n_p$	Read one standard hyper-word from tape and place in (n) to $(n+p—1)$
n: 8, Z	$(\bar{A}) \rightarrow n_p$	Transfers $(\bar{A})$ to hyper-register (n) to $(n+p—1)$ in standard form
n: 9, Z	$n \rightarrow \bar{S}$	Transfer hyper-control to n
n: 10, Z	$n \rightarrow \bar{L}$	Transfer hyper-control to n and store present hyper-control number plus one in L
n: 11, Z	$(\bar{L}) \rightarrow \bar{S}$	Transfer hyper-control to last location stored in L
n: 12, Z	$n_p \rightarrow D_r$	Place hyper-count number n_p in D_r
n: 13, Z	$n_p \xrightarrow{-} D_r$	Subtract n_p hyper-count units from D_r
n: 14, Z	$(D_r) \xrightarrow{+} \bar{K}$	Form the numerical address of the next hyper-command by adding (D_r) to it
n: 15, Z	Specified as desired	Transfer control to n but return to interpretation routine
n: 16, Z	$(n_q) \rightarrow \bar{A}$	Transfer (n) to $(n+q—1)$ to $\bar{A}$ as a complex number
n: 17, Z	$(n_q) \xrightarrow{+} \bar{A}$	Add (n) to $(n+q—1)$ to $\bar{A}$ as a complex number
n: 18, Z	$(n_q) \xrightarrow{-} \bar{A}$	Subtract (n) to $(n+q—1)$ from $\bar{A}$ as a complex number
n: 19, Z	$(n_q) \xrightarrow{\times} \bar{A}$	Replace $(\bar{A})$ by the complex product with (n) to $(n+q—1)$
n: 20, Z	$(n_q) \xrightarrow{\div} \bar{A}$	Replace $(\bar{A})$ by the quotient with regard to the complex number (n) to $(n+q—1)$
n: 21, Z	$(n_q)^{\frac{1}{2}} \rightarrow \bar{A}$	Replace $(\bar{A})$ by the square root of the complex number (n) to $(n+q—1)$
n: 22, Z	$(n_q) \rightarrow \bar{0}$	Print the complex number in (n) to $(n+q—1)$ in standard form
n: 23, Z	$(\bar{I}) \rightarrow n_q$	Read one standard complex number from tape and place into location (n) to $(n+q—1)$
n: 24, Z	$(\bar{A}) \rightarrow n_q$	Transfer $(\bar{A})$ to (n) to $(n+q—1)$ as a standard complex number
n: 25, Z	$\lvert (n_q) \rvert^2 \rightarrow \bar{A}_p$	Replace $(\bar{A})$ by the squared modulus of the complex number (n) to $(n+q—1)$
n: 26, Z	$(n_p) \xrightarrow{\times} \bar{A}_q$	Replace the complex number in $(\bar{A})$ by its product with the real number in (n) to $(n+p—1)$
n: 27, Z	$(n_p) \xrightarrow{\div} \bar{A}_q$	Replace the complex number in $(\bar{A})$ by its quotient with regard to the real number in (n) to $(n+p—1)$
n: 28, Z	$n_q \rightarrow D_r$	Place hyper-count number nq in D_r
n: 29, Z	$n_q \xrightarrow{-} D_r$	Subtract $nq.p_{11}$ units from D_r.

routines within the blocks and frequently provide additional input and output facilities; operations upon $(\bar{A})$ and on (X) and so on. These vary with the function block and are not listed in Table 2.

XV. Use of the Hyper-code

Diverse code systems for hyper-commands and machine commands may seem a disadvantage. It is, however, uneconomical to construct an interpretive routine whose hyper-code corresponds directly to the machine code and, moreover, such a system would involve frequent transfers from hyper-control to machine control and back in order to perform machine commands.

The apparent disadvantage of use of symbols for written hyper-commands differing from the code as it appears on tape is more than outweighed by the convenience of largely mnemonic symbols applied to all hyper-routines in the design stages.

It should be noted that $\{n \rightarrow \bar{S}\}$, which transfers hyper-control to n, is different from $\{n \rightarrow S\}$ or $\{(D_{13}) \rightarrow S\}$ which transfers real control without preparing to link back to the interpretation routine. In fact $(D_{13}) \rightarrow S$ switches back to normal machine operation; the subsequent commands being performed normally, not interpretively. Also, provision is normally made for transforming single counts to S into single counts to $\bar{S}$. Thus $\{s(A) \xrightarrow{c} S\}$ becomes in effect a function which may be written as $s(A) \xrightarrow{c} \bar{S}$, which causes hyper-control to advance by an additional unit if (A) is negative.

When coding hyper-routines on to tape full use is made of a special " input control routine H " (of 20 commands) which performs all the control operations of control D (listed in Pearcey and Hill 1953*b*) and two additional operations " P " and " Q ". The punch configurations, $P=31$, $3Y$ and $Q=31$, $15Y$ are placed immediately after the commands to which they refer. The effect of such control designations is to multiply the written address by p or q respectively, and then include the effect of any A type designation, punched immediately before that hyper-command, before inserting it into store. Thus with mp_{11} stored in the $1A$ location: $1A\{(n) \rightarrow \bar{A}\}$ P, punched as $0,1{:}7Y{:}n{:}0,Z{:}31,3Y{:}$, is stored as $\{(m+np) \rightarrow \bar{A}\}$, and $1A\{(\bar{A}) \rightarrow n\}Q$, punched as $0,1{:}7Y{:}n{:}8,Z{:}31,15Y$, is stored as $\{(\bar{A}) \rightarrow m+nq\}$. These special control designations, which automatically transform hyper-commands into the required form, together with the p-word and q-word loop control operations, virtually free programmers from concern with word lengths during design of programmes.

Insertion of the function block is usually performed in the order: primary B routine (from stepping switches); control H routine (from tape); spaces left for insertion parameters; first order function routines: second order function routines (which use first order ones); . . . ; highest order function routines; sequence and loop control routines; interpretation routine; directory; additional hyper-function routines.

XVI. Example of a Hyper-routine

As an example of a typical hyper-routine consider the routine required for the evaluation of a polynomial, of degree n, of the complex variable together with its derivative. The coefficients will be assumed to be complex numbers.

The method of computation by recurrence relation is adopted. Thus, if the independent variable be z and the coefficients be denoted by a_r, then p_r and p'_{r+1} are defined by:

$$f(z)=z^n+a_1z^{n-1}+a_2z^{n-2}+\ldots+a_n,$$
$$p_0=1,\quad p_r=p_{r-1}z+a_r,\quad (r=0,\ 1,\ 2,\ \ldots,\ n-1)$$
$$p_n=f(z),$$

and

$$p'_0=0,\quad p'_{r+1}=p'_rz+p_r,\quad (r=0,\ 1,\ 2,\ \ldots,\ n-1)$$
$$p'_n=f'(z),$$

where $f(z)$ is the function and $f'(z)$ its derivative with regard to z.

TABLE 3

A HYPER-ROUTINE FOR COMPUTING A POLYNOMIAL OF A COMPLEX VARIABLE AND ITS DERIVATIVE

Location	Symbol Code	Action	Tape Code
0 1S 3A,	$\{(\bar{A}) \rightarrow 0\}Q$	Causes z to be sent to $3S+0,\ 1,\ 2,\ 3$	3A; 0; 22, Z:, Q
1 2A,	$\{0 \rightarrow D_0\}$	Places order n in D_0 in form $8n$	2A; 0; 12, Z
2 3A,	$\{(3) \rightarrow \bar{A}\}Q$	Clears $(\bar{A})$	3A; 3; 16, Z; Q
3 3A,	$\{(\bar{A}) \rightarrow 1\}Q$	Clears p_r store	3A; 1; 22, Z; Q
4 3A,	$\{(\bar{A}) \rightarrow 2\}Q$	Clears p'_r store	3A; 2; 22, Z; Q
5 3A,	$\{(1) \rightarrow \bar{A}\}Q$	Sets p_{r-1} to $\bar{A}$	3A; 1; 16, Z; Q
6 3A,	$\{(0) \xrightarrow{\times} \bar{A}\}Q$	Forms $p_{r-1}z$	3A; 0; 19, Z; Q
7	$\{(D_0) \xrightarrow{\pm} \bar{K}\}$		0; 14, Z
8 3A,	$\{(4) \xrightarrow{\pm} \bar{A}\}Q$	Adds a_r to $(\bar{A})$	3A; 4; 17, Z; Q
9	$\{1 \xrightarrow{-} D_0\}$		8; 13, Z
10 1A,	$\{12 \rightarrow \bar{S}\}$		1A; 12; 9, Z
11	$\{(\bar{L}) \rightarrow \bar{S}\}$	$(\bar{A})'=p=f(z)$	0; 11, Z
12 3A,	$\{(\bar{A}) \rightarrow 1\}Q$	Substitutes p_r in place of p_{r-1}	3A; 1; 22, Z; Q
13 3A,	$\{(2) \rightarrow \bar{A}\}Q$	Sets p'_r to $\bar{A}$	3A; 2; 16, Z; Q
14 3A,	$\{(0) \xrightarrow{\times} \bar{A}\}Q$	Forms p'_rz	3A; 0; 19, Z; Q
15 3A,	$\{(1) \xrightarrow{+} A\}Q$	Adds p'_r	3A; 1; 17, Z; Q
16 3A,	$\{(\bar{A}) \rightarrow 2\}Q$	Replaces p'_r by p'_{r+1}	3A; 2; 22, Z; Q
17 1A,	$\{5 \rightarrow \bar{S}\}$	Returns hyper-control to 5	1A; 5; 9, Z

Notes

1. The four-word hyper-locations of the data block; 3A, 0–3, 4–7, 8–11, 12–15, hold z, p_{r-1}, p'_r, and zero respectively.
2. The coefficients a_r occupy hyper-locations 3A 16–19, 20–23, etc.
3. During insertion of this hyper-routine, 32 holds the head location of this hyper-routine (1S) and location 33 holds $8np_{11}$, where n is the degree of the polynomial, while location 34 holds the head location of the 3S-data block.

The hyper-routine designed to compute $f(z)$ and $f'(z)$ is shown in Table 3. It is assumed that upon entry of hyper-control into this hyper-routine $(\bar{A})=z$. Storage space is provided in a data block for the variable z, for the partial polynomial p_{r-1} and the derivative p'_r, and the number zero used for clearing $\bar{A}$ upon entry into the hyper-routine. Hyper-command number 10 could be replaced by $\{p_1 \xrightarrow{c} S\}$, since single counts into the sequence register made interpretively advance hyper-control one extra step forward.

When used in hyper-programmes, the coefficients would be inserted from tape and the make-up of the tape would be :

(i) Control H (including P and Q facilities for hyper-programme).
(ii) nT (which changes insertion-command to leave space for control parameters).
(iii) Interpretive programme (function block, directory, and any additional function routines) starting at location n.
(iv) $3S$, $x'T$ (which leaves space for data to be inserted subsequently and stores the head location of this block).
(v) Input hyper-routine (outlined in Section XII) starting at x'.
(vi) $4S$, input programme ; uses input hyper-routine and returns to primary B after $n+4$ items of input.
(vii) $4A$, $0 \rightarrow S$, D (transfers to input programme to insert data following).
(viii) Four zeros, clearing working space, and coefficients a_r in decimal code.
(ix) $x'T$ (prepares to over-write hyper-programme just used). ($c(A) \rightarrow 33$, D, $8np_{11}$, U) sets parameter for degree of polynomial.
(x) Polynomial hyper-routine (reference to $3S$-ed data block now possible).
(xi) xS master hyper-programme (varies with problem).
(xii) $xA, 0 \rightarrow S, D$ (transfer control to hyper-programme and starts computation).

XVII.—Other Hyper-address Codes

The great flexibility of the interpretive method is obtained at the expense of time, one hyper-command corresponding in time to a full cycle of the interpretive loop and any function selected from the function block. Usually, in the systems adopted, the time spent in the function block performing essentially difficult and involved operations cannot be avoided and is considerably greater than the time occupied in one traverse of the interpretation routine.

However, as the length of a hyper-word becomes greater, more time is spent in transfers to and from the hyper-accumulator. A change of the hyper-code to a three-address code avoids much of this extra organizational shifting of data. This method is adopted in the n-fold accuracy function block. In this system there is no hyper-accumulator. One hyper-command occupies two store locations. The first contains a numerical address $m_n^{(1)}$, an operation code F, say, and the distinguishing Z. The following location contains the second and third numerical addresses, $m_n^{(2)}$ and $m_n^{(3)}$, say, in p_1–p_{10} and p_{11}–p_{20} groups. Single address words without Z are interpreted as normal machine operations. Thus, for instance, the two-word hyper-command coded as $m_n^{(1)}$; F, Z: $m_n^{(2)}$; $m_n^{(3)}$ may cause $(m_n^{(1)})$ to be added to $(m_n^{(2)})$, each n words long, and placed into the n locations $m_n^{(3)}$, and so on according to the code number of F.

When, as occurs with large matrices, there are relatively few groups of data in the store, the address digits of one-word hyper-commands may be partitioned into a, b, c; F, Z. Here a, b, c, have values 0–8 (3 digits each) and refer to two of 16 store locations. Each of these groups of two words may be considered as partitioned into N, n, m, l, where N is the location of the first word in an $n \times m$ matrix of p-word numbers occupying l store locations in all.

The associated function block can be designed to provide operations such as matrix multiplication, inversion, division, etc. Additional calculations are performed, for each operation, to adjust the value of N, n, m, l of the result. Thus, in the multiplication which we may denote by $\{(a).(b)\rightarrow c\}$, $n_c=m_a$, $m_c=n_b$, and $l_c=p\times n_c\times m_c$ are calculated in the course of the operation and N_c may be provided by the programmer. By suitable design of the function block, the computer can be made to keep a record of usage of store space and adjust N_c itself. In these ways the programmer may be relieved of concern for lengths of data units and may programme in terms of a, b, c only.

The interpretive system has so far been applied mostly to various arithmetical methods as described, but has great potentialities for problems other than those that are purely arithmetical, since hyper-commands may be partitioned and coupled together in any manner desired so long as a suitable interpretation routine can be designed.

XVIII. References

Pearcey, T., and Hill, G. W. (1953*a*).—Programme design for the C.S.I.R.O. Mark I Computer. I. *Aust. J. Phys.* **6**: 316-34.

Pearcey, T., and Hill, G. W. (1953*b*).—Programme design for the C.S.I.R.O. Mark I Computer. II. *Aust. J. Phys.* **6**: 335-56.

Wilkes, M. V., Wheeler, D. J., and Gill, S. (1951).—" The Preparation of Programmes for an Electronic Digital Computer." pp. 162-4. (Addison-Wesley Press Inc.: Cambridge, Mass.)