THE EFFECT OF INTERPRETIVE TECHNIQUES ON FUNCTIONAL DESIGN OF COMPUTERS

By T. PEARCEY,* G. W. HILL,* and R. D. RYAN*

[*Manuscript received November* 12, 1953]

Summary

An analysis of the programmes for a number of computations performed by the C.S.I.R.O. Mark I computer shows that a great proportion of store space and operating time is occupied by the control of the course of the calculation, and that the proportion of the store space required as working space by a programme decreases as the size of the programme increases. The increasing use and great flexibility of interpretive techniques suggests that a very flexible, reliable computer, easy to use, could be constructed. Such a computer would possess only a relatively small amount of rapid-access erasable store, and a larger amount of rapid-access non-erasable store, in which would be held all interpretation routines, function blocks, and so on. The operator would require no knowledge of the actual machine code, but would place his hyper-programmes and data into a slow-speed backing store.

I. INTRODUCTION

In a previous paper (Pearcey and Hill 1954) a very powerful technique was described for the simplification of programme design in cases of complicated arithmetical computations. This is known as the interpretive technique and was described in detail as applied to the C.S.I.R.O. Mark I computer. The method is also applicable to a wide range of programmes and not necessarily restricted to essentially arithmetical computations.

The effect of the interpretive technique, so far as the user is concerned, is to provide the computer with a number of additional and convenient functions which are, to his view, built into the machine. Further, this set of functions can be changed by use of different function blocks.

The number of machine commands in the function blocks is frequently large, from 300 to 700 words or more depending upon the particular hyper-functions desired. Hyper-programmes are more frequently of the order of 100 hyper-words only.

An analysis of the interpretive function blocks used in the C.S.I.R.O. Mark I computer shows that less than 5 per cent. of the space occupied by an interpretive programme and its auxiliary routines is subject to change as the calculation proceeds. This estimate includes working space, variable commands and control parameters, link storage, and so on.

This indicates that a new form of storage of very simple design, little of which need be erasable, could be used. Such a computer would operate entirely

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

interpretively and possess a simple machine code, amounting at most to only 15 single operations. The user would require no knowledge of the machine code, but only of the hyper-code suitable for his computation. The programmer may choose a function block with a standard hyper-code or design one to suit the problem. This would greatly assist users to programme for " difficult computations ".

Such a computer, effectively capable of doing very elaborate operations, could be constructed with less equipment than exists in most currently operating machines. This would considerably improve reliability, assist maintenance, and achieve a more even balance of the responsibility for designing and constructing such a computer between the engineer and the mathematician. Simplification and reduction of equipment relieves the engineer of much effort and places more responsibility on the shoulders of the mathematician, who designs the routines for what the user could consider as " built-in " functions.

II. Analysis of Programmes

The commands of a programme can be grouped according to type (transfer, sequence shifts, addition, etc.), or according to their purpose (computation, sequence control, etc.). The manner in which the store is used is indicated by the proportion of store space occupied by such groups. The relative frequency of performance of commands of such groups is a useful basis for estimating efficiency of use of time.

A number of widely different programmes and interpretive function blocks have been analysed in these ways. The problem programmes include programmes for X-ray crystallographic synthesis, experimental data analysis, integration of singular functions, evaluation of determinants, matrix operations, and solution of various types of partial differential equations. The interpretive systems included those for floating decimal arithmetic, double precision arithmetic, and an arithmetic for which the index point occurs between the p_{11} and p_{10} digits, for both the real and complex variable.

III. Analysis of Store Usage

This analysis corresponds to a simple count of the commands of different groups as they occur in written programmes. The results of grouping commands according to type are shown in Table 1 for the case of problem programmes and interpretive function blocks. All values are percentages of the total store space occupied by the programme, excluding command space required only during insertion of the programme into the computer. The standard deviation listed with each percentage is believed to be reliable to 1 per cent.

It will be noticed that a rather large proportion, 32 per cent., of the total consists of plain transfers. The small standard deviation of this group, 2–4 per cent., indicates that such transfers form an essential part of the programmes and could not be reduced below about 30 per cent., even with the address system of Mark I. The overall length of programmes in a one-address system, like that of EDSAC, shows an average increase of 25 per cent. over the store space occupied by equivalent programmes in the Mark I system.

Most of the increase is due to additional transfers required, which bring the proportion of store space occupied by transfers in a one-address system to about 40 per cent.

The frequency of " add " or " subtract " commands is seen to be greater in problem programmes than in interpretive function blocks. Greater interconnection in function blocks increases the proportion of commands devoted to sequence shifts, which organize repetitions of the relatively smaller proportion

TABLE 1

STORE USAGE ANALYSIS : PERCENTAGE OF TOTAL PROGRAMME SPACE OCCUPIED BY COMMANDS OF VARIOUS TYPES

Programme	Type					
	Transfer	Add-Subtract	Control Shift	Discrimination	Variable	Other
Problem programmes	32±2	24±4	17±2	8±1	4±2	17±5
Interpretive function blocks	32±4	14±4	29±6	5±1	5±1	15±5

Length of problem programmes, 100–300 commands
Length of interpretive function blocks, 250–500 commands

of commands devoted to arithmetic operations. This emphasis, indicated by the proportion of sequence shifts, is found to increase with the degree of complexity of the arithmetical system.

In both types of programme, discriminations are relatively few in number, 5–8 per cent. Use of " *E* " and " *G* " type commands in the one-address EDSAC system for sequence shifts as well as for discriminations would bring the proportion of such operations to 20–30 per cent. It is very important to note that the

TABLE 2

STORE USAGE ANALYSIS : PERCENTAGE OF TOTAL COMMAND SPACE OF PROBLEM PROGRAMMES DEVOTED TO VARIOUS PURPOSES

Computation	43±16
Sequence control	23± 7
Linkage	15± 5
Command variation	13±10
Other	7± 1

proportion of variable commands is quite small, 4–5 per cent. Although this type of command occupies relatively little space, it is vitally necessary in flexible programme design.

The results of grouping commands in problem programmes according to purpose is shown in Table 2. This shows that less than half the programme space is devoted to calculation, and this includes all transfers etc. involved in

the actual calculation of results. Without transfers and similar organizational commands, the proportion devoted to calculation drops from 43 per cent. to about 30 per cent. Apart from the 7 per cent. devoted to connection with the user (input, output, hoots,* etc.) the remainder is devoted to controlling the course of the calculation.

IV. Machine Speed Efficiency

Since commands in some loops are repeated more often than others, a simple count of commands in written programmes is not a reliable indication of relative frequencies of performance of command groups. An analysis of performance frequencies shows considerably greater variation than the store space analysis. Depending on the programme, the proportion of time devoted to calculation of results varies from 30 to 70 per cent. The proportion devoted to organization varies from 70 to 30 per cent. Communication with the user by hoots, input, prints, etc. may occupy up to 50 per cent. of the operating time and is very variable. Of significance is the fact that multiplication occupies up to 25 per cent. of the time. In such cases, replacement of the machine operation by an equivalent routine for multiplication would decrease multiplication speed by a factor of about 240 and overall computing speed would be reduced by, at most, a factor of about 60.

V. Variables and Working Space

Only a small proportion of a programme changes during operation, and the working space it requires is only a small proportion of its size in the store.

Programmes require working space for the data operated on (e.g. accumulators, temporary storage space, etc.), store space to hold constants, parameters, variable commands, count parameters, and other cycling controls. All locations occupied by data of these types must be erasable, since the contents change both during calculation and from programme to programme. In general the proportion of programme space occupied by such variables tends to decrease as the size of the programme increases, due to use of the same stores for variables of many routines.

While single routines from the Mark I library, with lengths 30–90 commands, required 13–17 per cent. of their lengths for constants and working and variable spaces, problem programmes consisting of groups of library and special routines required only 4–10 per cent. of their overall length of 180–350 commands. This drop is largely due to use of the same working space for all routines, e.g. the working registers A, B, C, D, and H.

Problem programmes rarely exceed 350–400 commands in length. Problems requiring large groups of commands have been those suited to interpretive techniques, involving function blocks of lengths 300 commands upwards. The hyper-routines and hyper-programmes range in length from 100 commands upwards, but the amount of variable store required by the function block is independent of this further length. In estimating the variable space required, all hyper-registers ($\bar{A}$, $\bar{\bar{X}}$, $\bar{S}$, $\bar{\bar{K}}$, etc.), temporary storage spaces, and link and

* A "hoot" is a signal made audible by the loudspeaker.

loop control spaces must be included. About 20 locations are required for " control " registers, and a further $5n$ locations for " arithmetical " registers for n-word numbers. In cases of complex variables, this latter contribution is double that for real variables. Any increase in the effective value of n increases the total size of the function block (" $n=4$ " block is 40–60 per cent. longer than " $n=2$ " block), thereby tending to keep constant the ratio of variable space to programme space. For interpretive function blocks of lengths ranging from 250 to 600 commands used with the Mark I computer, this ratio was 4·5–5·5 per cent.

As the value of n increases, a stage is reached where it is more efficient to change from a one-address system to a three-address system. This avoids waste of store space for n-word arithmetical registers and waste of time for n-word transfers. Eliminating hyper-arithmetical registers causes the proportion of variable stores to decrease below 5 per cent. with increase in size of the function block. This suggests that there is an upper limit of about 100 store spaces to the amount of variable store needed by all types of programmes.

Experience so far shows a greatly increasing use of interpretive systems in Mark I, with frequent use of the same function blocks. The general utility, simplicity, and flexibility of interpretive methods tends to make them popular with users.

Computers, suitably designed for use of interpretive methods, could store the standard function block in a fixed or semi-fixed storage system of rapid access ; only a relatively small additional amount of rapid-access erasable store being required for working space, etc. Large amounts of problem data would normally be held in a slower-access backing store since such stores of large capacity are inexpensive. Small amounts of problem data may be held also in more expensive rapid-access stores.

VI. Operation Rates in Hyper-control

The advantages of interpretive methods are obtained at the expense of reduced speed of hyper-operations. Hyper-speed, the rate at which hyper-commands are performed, is considerably slower than machine-command speed and varies with the design of the function block used. A measure of the hyper-speed is the number of machine commands performed by each hyper-operation. This is listed in Table 3 for each hyper-function of a number of interpretive systems used in the Mark I computer. These values may be reduced to time in seconds for the Mark I computer by division by 500. The command symbolism for hyper-functions is that given in a previous paper (Pearcey and Hill 1954).

It will be noted that machine commands, those hyper-commands not terminated by Z, are performed most rapidly (14–18 command times). Transfers to and from the accumulator, real addition and subtraction, hyper-sequence and hyper-count operations are relatively rapid (40–60 commands). Multiplication hyper rate varies from 50 to 100 commands for the real variable. Hyper rate for division and square rooting decreases from 100–300 commands for single word precision to about 1500 commands for extended precision, due to organizational complexities introduced by extension to high precision. To the input

and output rates of 100–1000 commands must be added the time required for the mechanical operations of reading, printing, or punching.

For complex variables the hyper rate of transfers is only slightly decreased, that of addition and subtraction is roughly halved and of multiplication, division, and square rooting is decreased by four to five times.

The average rate of performance of any hyper-programme depends on the particular hyper-commands of which it consists. An analysis of the frequency of use of hyper-commands has been made for hyper-programmes for solution

TABLE 3

HYPER-SPEEDS

Real Hyper-operations*	No. of Machine Commands† per Hyper-operation				Complex Hyper-operations	No. of Machine Commands per Hyper-operation			
	A	B	C	D		A	B	C	D
$(n_p) \rightarrow \bar{A}$	40	40	43	50	$(n_q) \rightarrow \bar{A}$	50	50	44	59
$(n_p) \overset{+}{\rightarrow} \bar{A}$	43	72	50	160	$(n_q) \overset{+}{\rightarrow} \bar{A}$	90	180	57	400
$(n_p) \overset{-}{\rightarrow} \bar{A}$	55	72	51	180	$(n_q) \overset{-}{\rightarrow} \bar{A}$	140	180	61	420
$(n_p) \overset{\times}{\rightarrow} \bar{A}$	104	57	51	150	$(n_q) \overset{\times}{\rightarrow} \bar{A}$	430	350	77	1100
$(n_p) \overset{\div}{\rightarrow} \bar{A}$	1500‡	120‡	123‡	1400‡	$(n_q) \overset{\div}{\rightarrow} \bar{A}$	4700‡	750‡	280‡	4600‡
$(n_p)^{\frac{1}{2}} \rightarrow \bar{A}$	1500‡	260‡	250‡	1700‡	$(n_q)^{\frac{1}{2}} \rightarrow \bar{A}$	5000‡	1400‡	670‡	6500‡
$(n_p) \rightarrow 0t$	930§	180§	100§	980§	$(n_q) \rightarrow 0t$	1850§	300§	170§	1900§
$(I) \rightarrow n_p$	240§	190§	130§	320§	$(I) \rightarrow n_q$	470§	320§	220§	600§
$(\bar{A}) \rightarrow n_p$	40	40	46	50	$(\bar{A}) \rightarrow n_q$	52	52	52	66
$n \rightarrow \bar{S}$	34	35	40	42	$\lvert n_q \rvert^2 \rightarrow \bar{A}_p$	220	170	61	540
$n \rightarrow \bar{L}$	40	41	46	48	$(n_p) \overset{\times}{\rightarrow} \bar{A}_q$	250	130	79	390
$(\bar{L}) \rightarrow \bar{S}$	37	38	43	45	$(n_p) \overset{\div}{\rightarrow} \bar{A}_p$	3500‡	280‡	220‡	3000‡
$n_p \rightarrow D_r$	45	46	51	53	$n_q \rightarrow D_r$	45	46	51	53
$n_p \overset{-}{\rightarrow} D_r$	44	45	50	52	$n_q \overset{-}{\rightarrow} D_r$	44	45	50	52
$(D_r) \overset{+}{\rightarrow} \bar{K}$	43	44	49	51					
$f(A) \rightarrow \bar{A}$	60‡	60‡	65‡	65‡					
Machine command ..	14	14	16	18					

* The command symbols are those described by Pearcey and Hill (1954).

† The arithmetical systems involved in the columns of hyper-rates are : A, double precision arithmetic ; B, floating decimal arithmetic ; C, arithmetic with index point between p_{11}, p_{10} digits ; D, floating decimal index, double precision arithmetic.

‡ Average values only.

§ Not including mechanical printing or reading time.

of polynomial equations using complex arithmetic, integration of singular functions using floating index arithmetic, and matrix operations in floating index, double precision arithmetic.

Of all the hyper-commands performed, hyper-transfers form 10–40 per cent., hyper-control operations 10–40 per cent., addition and subtraction 5–20 per cent., multiplication 5–20 per cent., and machine commands less than 30 per cent., while the relatively slower operations, division and square rooting, require less than 5 per cent., with input and output less than 5 per cent.

By suitably combining the hyper rates with the frequencies of performance of hyper-commands, an average hyper rate of 50–90 is obtained for the cases cited. For an estimate of efficiency these figures must be compared with the average speed with which such operations could be performed by the direct technique of cueing and linking outlined in a previous paper (Pearcey and Hill, 1953). By direct techniques the duration of each hyper-operation could be decreased by omission of the 30 " redundant " machine commands in the interpretation loop reducing the equivalent operation speed to 20–60 machine commands per hyper-operation. At the same time most hyper-commands would be replaced by 2–3 machine commands for planting address, link datum, and operation code, thus doubling or trebling the length of the hyper-programme. Thus interpretive techniques are seen to be roughly twice as expensive in time and half as expensive in command space as direct programme techniques.

VII. Reference to Slow-access Store

If problem data and hyper-programme are stored in a slow-access store such as a magnetic drum, operating speed would be seriously reduced if transfers to and from the slow-access store were very frequent. While all the hyper-operations would require a transfer of the hyper-command from the slow-access store, a proportion p involve a further transfer of an n-word number. In an interpretive system for n-word numbers, with an average hyper rate of r machine commands performed for every hyper-command, some $r/(1+pn)$ machine commands would be performed on the average for very slow-access store transfer.

In the cases cited, $r/(1+pn)=15$, 20, 50, so that, unless the access time of the low-speed store were more than 50 times that of the rapid-access command store, there would be no serious reduction in operating speed. The store access ratio (ratio of low-speed to high-speed access times) should not be greater than 50 and preferably less than 15.

VIII. Bearing upon Functional Design

The results of these discussions have direct bearing upon the functional design of computers. The factors which are seen to be important are as follows :

(1) The interpretive method simplifies programme design and has the effect of providing the user with many additional " built-in " functions.

(2) The interpretive method is flexible, the code and address style being chosen at will.

(3) Function blocks can be designed for interpretive programmes which will be of very common use. A small number of standard blocks would satisfy the great majority of users.

(4) The proportion of the store required by a programme as working space decreases as the size of the programme increases. For interpretive arithmetical programmes this is about 5 per cent. of the programme space only. This only need be erasable during operation, apart from the problem data store.

(5) Considerable time is spent by transfers and control operations and in the control of the programme itself.

(6) Reference to data for computation is relatively infrequent. Such data could be held in a slow-access store with only a small degree of loss of time.

(7) About 50–90 machine commands are performed to each hyper-command.

Accepting these points, a computer could be designed having adequate speed, relatively simple engineering requirements, and great flexibility and ease in programming. Thus we may adopt the following principles for its design :

(1) Use the interpretive method of programming entirely.

(2) The machine code should be short and possess only a small number of simple functions, e.g. a one-address system.

(3) The user would use only a hyper-code and would not require knowledge of the machine code.

(4) Commands should be adopted from a rapid-access store, data and hyper-programmes being held in a slower access or backing store.

(5) The rapid-access store need have only a small fraction of its total capacity erasable, the rest fixed, since function blocks would be stored permanently. The erasable store would be used as working space for the function block and its associated interpretation routine and directory. Provision should be made for adopting variable commands from the erasable part of the store.

From this it will be seen that the hyper-programme and problem data, the parts supplied by the user, would normally be held in a backing store only. The machine adopts its commands from the rapid-access store. The fact that only a small part of the high-speed store need be erasable should considerably simplify the engineering problems associated with rapid-access storage.

IX. The Store

As in the case of most computers, the logical design depends largely upon the physical nature of the storage system adopted. Much effort is expended in the design of large-capacity storage systems possessing write-erase features and rapid accessibility. This expenditure could be avoided with probable reduction of equipment and additional reliability if a small amount of rapid-access erasable store were used together with a larger amount of fixed or, rather, semi-permanent store. The erasable store would provide working space and all hyper-registers, and the function blocks providing the hyper-functions would be inserted into the non-erasable store. These stores will be called the variable and fixed high-speed stores. The serial mode of operation will be presumed on the grounds that less equipment is required in the control and operation of such a computer than in an equivalent parallel operating machine. Digit recurrence rates of at least one megacycle should be attainable.

It is also clear that a large backing store must be provided for storing large amounts of erasable problem data and the hyper-programme and hyper-routines.

X. Fixed High-speed Store

For the fixed store a " flying spot " system similar to that used in the transmission of films by television seems a possible technique, not requiring excessive effort for development. In this system the image on the screen of

a high definition cathode-ray tube is focused upon a film frame which is scanned by a light spot tracing a raster on the face of the screen. The light transmitted by the film is detected for transmission by a photomultiplier tube.

By replacing the film by a matrix representing the routines of the function block and other standard routines, and controlling the passage and position of the light spot on the screen, the output of the photomultiplier tube may be made to provide commands to the computer.

The advantages of such a system would be those of relatively small amounts of standard equipment, high digit capacity per frame of the matrix, and small access time equal to the time required to shift the spot from one place to another.

Commands and standard constants could be stored on the matrix in rows of spots in binary code. It seems possible to store up to one thousand 16-digit commands on one matrix. The application of suitable deflexion voltages corresponding to a given serial number would position the spot ready to read out a selected command. The spot may then be brightened and moved across the screen at a fixed rate so as to scan the required word on the matrix. Such a scheme would provide serial transmission of digits, with a possible digit rate of one megacycle.

In practice it may be possible to avoid the use of an optical system by placing the matrix directly against the cathode-ray oscilloscope face. The store could be extended by addition of similar units.

The disadvantage of the fixed store is that special matrices must be constructed either photographically or mechanically. Against this argument, however, such matrices would be standard, few in number (about 20 different matrices), made once only, and frequently used.

Other methods of constructing stores involve the use of one element for each digit as in the case of magnetic cores or dielectric elements. With the former of these a considerable reduction of the number of cores can be attained by suitably weaving a number of reading circuits through one row of cores. However, for convenience of rapid changing the optical system is preferable.

The use of fixed store matrices would relieve the programmer of much tiresome work in the use of multiple accuracy and other elaborate arithmetical methods.

XI. Erasable High-speed Store

This store could take one of a number of possible forms, such as magnetic core or electrostatic matrices, delay lines, etc. Delay lines, although " volatile ", may be made to operate at higher than one megacycle recurrence rates and would be highly suitable for serial operation. The access time would be reduced by having as few words as possible placed in each delay line, consistent with there being a reasonably small amount of equipment. One possible means of achieving this is to adopt a " multiplex " scheme in which more than one word is held in a delay line of one word " length " by interspacing the digits of the words stored. The access time to any word is thus reduced to a maximum of one word-time.

The digit period of both types of store must be the same since the transfers are presumed to be serial. The storage locations may be numbered serially (0 to $N-1$), the erasable store occupying the earliest positions (0 to $M-1$), with the fixed store following without a break (M to $N-1$).

XII. Low-speed Store

This will be referred to by the programme held in the high-speed store and will thus not be required to possess an access time as small as that needed in the high-speed store. Nevertheless, the access time for the backing store, or low-speed store, should be as small as possible within reasonable engineering safety margins. This store would probably take the form of a magnetic drum, a system which is known to be highly reliable and economical in space and equipment.

XIII. The Code System

Commands of the fixed store matrices are recorded in " machine code ", each code number corresponding to a single machine function. This code is fixed by the design of the machine. The interpretation codes are variable and are chosen by the user or the mathematician to suit the type of calculation. To each of the different interpretation codes there will correspond a set of machine code matrices. It will be assumed that the machine operates in the binary code, with complementary representation of negative numbers, the sign digit being the last digit transferred in each word.

XIV. The Machine Code

From the point of view of simplicity of use, of logical design, and of minimum equipment the best design would involve serial operation in a one-address code system with only essential machine functions. The consequent disadvantages of increase in programme size and reduction of machine speed is amply compensated by the size of the cheap store available and its reduced access time.

Each high-speed store location will be referred to by its address n, and its content denoted by (n). The total number of low-speed store locations will be N' and any particular such location will be referred to by its address n' with content (n').

We shall assume that one word occupies 16 binary digits, although this may be chosen at will, and that all transfers called by the machine code will be 16-digit transfers. There would be no basic change to make such a machine to possess a 16-digit command code and to transfer 32 digits under such a code.

A central accumulator A of double length capacity, i.e. 32 digits for 16-digit transfers, whose content is denoted by (A), must be provided. The accumulator could be of the delay line type with suitable input and output gates and special devices for multiplication and left shifts. Though multiplication could be performed by a programme stored in the machine by repeated addition and shifting, it is so frequently used in practice that it is best to provide built-in multiplication. In this case additional arithmetical registers must be provided but need not be referred to explicitly by the machine code.

Operation proceeds serially, that is, commands are adopted from sequential high-speed store locations except when transfers of control are called. The

sequence register containing the current command address will be referred to as S and its content as (S). The sequence register must be provided with a $\frac{1}{2}$-adder so that it may generate sequential addresses by addition of a unit digit from the sequence unit after extraction of each command. These facilities would also be used whenever sign test functions are called. The interpreter, which receives each command for decoding, will be denoted by K. The input register I and output register O connect the machine with the operator.

All commands will contain a store address and a function number. Some of these functions will refer to variable store $0 \leqslant n < M$, to the fixed store $M \leqslant n < N$, and to the backing store $0 \leqslant n' < N'$, and we want to allow N' to be as great as possible.

It is found that there are only three commands which refer to fixed store locations and to low-speed store locations. These are:

(1) Shift control to n	denoted by	$n \rightarrow S$
(2) Transfer (A) to n' in the low-speed store	,, ,,	$(A) \rightarrow n'$
(3) Transfer (n') in the low-speed store to A	,, ,,	$(n') \rightarrow A$

A special group of two digits in the function part of the code may be used to distinguish these from all other functions. These may be the two most significant digits, i.e. p_{16} and p_{15} of a 16-digit command consisting of digits $p_{16}, p_{15}, \ldots, p_1$ in descending order of significance.

Other commands refer only to the erasable high-speed store $0 \leqslant n < M$, for which fewer address digits are required, e.g. $p_1, \ldots, p_8$ provides for $0 \leqslant n < 256$.

Hence the following address scheme may be adopted:

p_{16}	p_{15}	p_{14}–p_9	p_8–p_1	Function
1	1	$0 \leqslant n' < N'$		$(n') \rightarrow A$
1	0	$0 \leqslant n' < N'$		$(A) \rightarrow n'$
0	1	$0 \leqslant n < N$		$n \rightarrow S$
0	0	Function code	$0 \leqslant n < M$	Other functions

As illustrated, N and N' would be limited to 16,384 locations. This is probably excessive for N.

The remaining functions may be chosen as follows and refer only to $0 \leqslant n < M$:

(1) Transfer (n) to A (accumulator)	denoted by	$(n) \rightarrow A$
(2) Transfer (A) to n	,, ,,	$(A) \rightarrow n$
(3) Add (n) to (A) and hold the sum in A	,, ,,	$(n) \xrightarrow{+} A$
(4) Subtract (n) from (A) and hold the difference in A	,, ,,	$(n) \xrightarrow{-} A$
(5) Form digit by digit product of (n) with regard to (A) and place in A	,, ,,	$(n) \xrightarrow{\cdot} A$

(6) Form product of (n) and (A) and place in A	,,	,,	$(n)\overset{\times}{\rightarrow}A$
(7) Rotate (A) by n places to the left	,,	,,	$n\rightarrow L$
(8) If (A) is $-$ve add unit to S (sequence register)	,,	,,	$S(A)\overset{c}{\rightarrow}S$
(9) Transfer content of input register (I) to location n	,,	,,	$(I)\rightarrow n$
(10) Transfer (n) to output register O and record	,,	,,	$(n)\rightarrow O$
(11) Transfer (S) to location n	,,	,,	$(S)\rightarrow n$
(12) Transfer (n) to S	,,	,,	$(n)\rightarrow S$
(13) If (n) non-zero stop sequence	,,	,,	$(n)\rightarrow T$
(14) Transfer (n) to the interpreter K and add the next command to it	,,	,,	$(n)\overset{+}{\rightarrow}K$

Of these commands, (5) is useful in any interpretation routine and will therefore be used in all programmes, and (7) allows of both the greater and lesser half of products to be transferred from A and also facilitates the use of strobe methods in routines.

Command (8) allows all discriminations to be made, with the aid of the shift command (7) on any digit not necessarily the sign digit. Commands (11) and (12) allow links to be stored in the erasable store and hence allow transfers to and from routines in the fixed store to be made. Command (14) is special and allows for variable commands. The address may be stored as a parameter in the erasable store at n. If $(n)\overset{+}{\rightarrow}K$ is placed in m and $(A)\rightarrow 4$ is placed in $m+1$, then the latter command is adopted as $(A)\rightarrow\overline{4+(n)}$. It will be noticed that machine commands may be adopted from either the fixed or erasable stores and that the serial ordering of the locations of the erasable part continues into the fixed store, some locations of which may exist for which $n<M$. Standard data used by the function blocks will lie either in the erasable store or in that part of the fixed store for which $n<M$.

XV. Store Selectors

Considerable convenience is achieved by adopting commands from the erasable store as well as from the fixed store. This would assist in insertion of hyper-programmes and data by enabling the insertion routine to be held in the erasable store. Further, short programmes of standard type or short hyper-programmes could be stored in the erasable part.

There would be three store selector units, each of a different type; one for the erasable store, one for the cathode-ray oscilloscope fixed store, and one for the backing store. In selection of commands the contents of the sequence register would be transmitted to two selectors, that for the erasable store and that for the fixed store. If $0\leqslant(S)<M$, the p_{14}–p_9 digits would be zero and p_8–p_1 digits in the erasable store selector would operate a " tree-type " selector preparing the erasable store to transmit. For $M\leqslant(S)<N$, the p_{14}–p_{11} digits (p_{14}–p_9 digits not all zero) in the fixed-store selector would apply the required

deflection potential for placing the spot on the cathode-ray oscilloscope at the beginning of the required trace. The digit group p_{14}–p_9 thus selects which part of the high-speed store is used ; if this group is clear of units the erasable store will be operated ; otherwise the fixed store would be used.

The backing-store selector could be of the counting-coincidence-selector type operated by digits p_1–p_{14} ($0 \leqslant n < N'$) and would be used in selection of locations for transmission of data or hyper-commands during performance of machine commands.

XVI. The Hyper-Codes

Hyper-functions will vary and be chosen at will, and for each set suitable matrices will be designed.

In particular, three hyper-functions may refer to addresses n' in the low-speed store where $0 \leqslant n' < N'$. These will be :

(1) Transfer hyper-control to n'	denoted by	$(n') \rightarrow \bar{S}$
(2) Transfer the p-fold hyper-word in locations n' to $\overline{n+p-1}'$ to the hyper-accumulator	,, ,,	$(n_p') \rightarrow \bar{A}$
(3) Transfer the content of the p-fold hyper-accumulator to the p-fold hyper-word in location n' to $\overline{n+p-1}'$	,, ,,	$(\bar{A}) \rightarrow n_p'$

In these functions p will be implicit and specified in the routines of the fixed store (see Pearcey and Hill 1954). All other hyper-functions will refer at most to the M locations of the erasable store. In such cases, $n(<M)$ must be specified in the hyper-command, for instance, in digit positions p_1 to p_8, leaving digits p_9 to p_{14} for the remaining hyper-function code providing for 64 possible hyper-functions.

By suitably choosing the hyper-code it may be possible to extend the code to additional hyper-functions and to change from one to another mode of interpretation, packing more than one hyper-command into a single word, or extending to a two- or three-address hyper-code as desired.

XVII. Organization of the Computer

The computer would contain only the essential registers. The cycle of operations would be as follows :

(1) The content of the sequence register is transmitted to the high-speed store selectors.

(2) The selected command is transmitted from the high-speed store to the interpreter register where it is decoded, and a unit is added to the sequence register.

(3) The numerical address held by the interpreter is transmitted to the high-speed store selector and to the low-speed store selector and the function addresses decoded by suitable selectors and suitable gating operations.

(4) The required transfer between registers takes place.

In the first of these operations the sequence register content is transferred to both fixed store and erasable store selectors. If the p_9–p_{14} digit group is zero, the command is selected from the erasable store ; otherwise from the fixed store. In the second operation of the computer sequence, the interpreter receives the command. The transfer occurs over a minor cycle period coincident with digit periods p_1–p_{16}.

Decoding would occur in three parts. The interpreter may be thought of as a distributed constant delay line and similar to the sequence register. Digits p_{15}–p_{16} are connected to a separate decoder providing four outputs. Digits p_9–p_{14} are connected to a " function decoder " of the tree type. Digits p_1–p_{14} are transmitted in parallel to the variable store selectors and to the backing store selectors by activation of gates by the sequence unit waveforms.

Transfers from the sequence and interpreter registers could be made in parallel by initially tapping along the lines for the output digits.

For two of the configurations of p_{15}, p_{16} digits (1,1 and 0,1) the backing store output or input gate is activated, the store location being selected by the p_1–p_{14} digits in the backing store selector. If the p_{15}, p_{16} digits are 1,0 respectively, the p_1–p_{14} digits of the interpreter are gated in parallel to the sequence register. the configuration 0,0 of p_{15}, p_{16} digits permits the p_9–p_{14} decoder to be activated, selecting the required function by opening gates for transferring the p_1–p_8 digits in the erasable store selector to select the datum location.

Thus, in the first operation of the computer-sequence, either erasable or fixed stores may be called to transmit, and in the third operation either the backing store or erasable store is involved. In the case of low-speed backing store transfers, the sequence unit must not stimulate action until coincidence is found by the selector.

XVIII. Speed of Operation

The speed of operation obtained depends upon the access time of both rapid and slow stores. It seems reasonable to choose these speeds such that the access time of the low-speed store is about 20–40 times the access time to commands, so that operating time is fairly evenly divided between high-speed commands and backing store transfers.

The one-address system suggested turns out to be similar to that adopted for EDSAC (Wilkes, Wheeler, and Gill 1951) and differs considerably from that of the C.S.I.R.O. Mark I computer. It may be expected, therefore, that the function blocks designed for the proposed machine would occupy 25 per cent. more words than those listed in Table 3. The estimate of high-speed commands to each backing store transfer, suitably weighted, would amount to about 30 high-speed commands.

If we put the access time of the backing store at about 4 msec (in practice the smaller this is the better) and assume that a further 4 msec be occupied by the 30 high-speed operations, we arrive at a period of about 133 μsec per high-speed function. At a 1 megacycle recurrence rate, the two serial transfers would occupy 32 μsec, leaving over 50 μsec each for operations 2 and 4 in the

computer cycle which involves switching or transferring digits to selectors and raising the selectors to suitable marginal levels. These speeds appear readily attainable and might even be exceeded.

At such a speed of operation the proposed computer would be able to do floating index and double precision operations at about one-eighth of the speed now required by the Mark I to perform normal operations, and corresponds to an increase in real speed over the Mark I for similar operations by a factor of five.

XIX. References

PEARCEY, T., and HILL, G. W. (1953).—Programme design for the C.S.I.R.O. Mark I computer. II. Programme techniques. *Aust. J. Phys.* **6**: 335-56.

PEARCEY, T., and HILL, G. W. (1954).—Programme design for the C.S.I.R.O. Mark I computer. III. Adaptation of routines for elaborate arithmetical operations. *Aust. J. Phys.* **7**: 485.

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.)