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

## IV. AUTOMATIC PROGRAMMING BY SIMPLE COMPILER TECHNIQUES

By G. W. HILL*† and T. PEARCEY*

[*Manuscript received January* 24, 1956]

*Summary*

Techniques are described for using the C.S.I.R.O. Mark I computer to assemble complete programmes from greatly reduced input programmes by selecting and transferring routines from standard libraries and forming appropriate commands to interconnect routines and the problem programme. The " compiler " process is described for computer-code programmes, and for hyper-code programmes which use interpretive routines to provide elaborate arithmetical operations corresponding to each hyper-code.

Examples of compiler routines are given in computer-code command form and as schematic outlines, and the compiler processes are illustrated by simple problem examples. Possible extensions of compiler techniques are considered and processes for using compiler techniques are discussed.

Use of compiler techniques reduces considerably the " human error " in programmes, training time of programmers, and the time required for preparation, insertion, and checking of problem programmes.

## I. INTRODUCTION

As the use of automatic computers expands, increasing attention is being paid to the principles involved and the techniques used for rapid programming, in particular to the more advanced aspects of programming known as " automatic programming " (Office of Naval Research 1954). This is a logical extension of the principles of programming already well established and is being developed largely from a need to assist the user by reducing the effort and specialized knowledge required of him, to reduce the frequency of errors in programmes etc. and thereby increase the use of a computer.

Much of the routine type of work required of a programmer can be transferred to the computer. The programmer need supply only basic, non-redundant information about his problem, in a suitably chosen " language ", leaving the computer to build up the detailed programme in its own, or command language. Recent work by Laning and Zierler (1954) indicates the trend towards the presentation of computing problems in language closer to that of the user. With further developments of automatic programming techniques we may expect a user to be able to submit his problem in his own language and to have no concern with the basic command code of the computer.

A practical automatic programming system involves a number of specially designed routines which facilitate the transformation of the information presented

* Division of Radiophysics, C.S.I.R.O., University Grounds, Chippendale, N.S.W.

† Present address : Division of Mathematical Statistics, C.S.I.R.O., University of Adelaide.

into a complete programme which the computer can " understand " and later perform. The first stage toward such a system is the development of compiler routines which are here discussed, the principles of which are illustrated in relation to a particular machine.

To simplify the task of programming, automatic programming of the C.S.I.R.O. Mark I computer by means of simple compiler techniques has been developed. This computer is of the high-speed fully automatic type, which is directed by a suitable programme of coded commands. Electronic equipment provides elementary arithmetical and logical operations as well as operations used for organization of sequences of operations and for communication between the computer and the user. To each elementary operation provided by electronic equipment corresponds a " computer code ". More elaborate operations are obtained by appropriate routines of commands expressed in computer code. Special groups of routines have been designed to enable elaborate operations to be invoked by " hyper-code " commands. Programme design in terms of these code systems has been described in previous papers ; computer-code programme techniques in Part II (Pearcey and Hill 1953*b*) and hyper-code programme techniques in Part III (Pearcey and Hill 1954). The notation there described has been freely used in the present discussion and has been used in the presentation of Tables 3 and 7 of the actual compiler routines.

Programming is the process of translating a problem from the mathematical language of algebraic expressions and methods of solution into an appropriate sequence of data to be punched on the input tape as " input codes ". Special insertion routines enable the computer to translate input codes into the coded commands of the problem programme. One of the simplest insertion routines, the " primary " routine, uses upper or lower halves of command codes as input codes and the programmer must translate the problem into the form of half command codes. The exacting task of evaluating addresses for upper-half command codes can be performed by the computer by use of a " control " routine to decode further input codes known as " control designations ".

Even more of the burden of programming can be transferred to the computer by extending the range of input codes, and increasing the complexity of the decoding routines. The technique of using a computer to construct commands for problem programmes is known as automatic programming and the decoding routines used for this purpose are called " compiler routines ".

Compiler routines designed for use on the C.S.I.R.O. Mark I computer decode special input codes representing elaborate operations supplied by routines stored as a library in the computer's slow-access store. The routines are transferred to the store containing the problem programme, and appropriate interconnexion commands are inserted in the programme.

## II. The Function of Compiler Routines

The input codes, used by the programmer to specify the operation required, may be arranged according to a variety of conventions. A simple convention considered in this paper requires the input codes to be arranged in a manner similar to that adopted in normal programming, thus exploiting the fact that

users will generally be acquainted with normal programming techniques. A logical extension to a more elaborate convention will be indicated which corresponds to coding of " three-address " operations with a corresponding reduction of the amount of punching required in preparing an input tape. This requires a few additional techniques to be learned, such as counting a three-address operation as corresponding to a number of commands when calculating addresses of subsequent commands.

A much more elaborate compiler routine is required to extend the convention for input codes to correspond with the widely known symbolism used in normal algebraic expressions. Such a " symbolic-compiler " routine greatly reduces the amount of specialized knowledge required of the user since input codes can be punched directly from algebraic expressions. The techniques required for this elaborate process include those to be considered in this paper.

The simple compiler routine to be considered merely relieves the programmer of the task of selecting standard routines and incorporating them into the problem programme. These standard routines are made continually available to the compiler routines by storing them as a " library " in store space $S_L$. Library routines must be selected and transferred from the library to the problem programme in store space $S_P$ and appropriate interconnexion commands formed. These operations are performed almost unconsciously by the human programmer but must be programmed explicitly and systematically for compiler routines.

Interconnexion commands in the resultant programme must transfer problem data from definite store locations implied by the problem programme to registers required by the library routine and transfer results from those registers back to the same or other store locations. The interconnexion commands also shift control to the routine and arrange for control to be returned to the programme commands following use of the routine.

Consequently, the programmer must supply extra input codes to specify the locations implied by the problem programme for operands and result, and a " compiler code " to specify which library routine is involved. The library store, $S_L$, contains " library data " associated with each routine, which specifies which registers are to hold operands and results and indicates the " linking " mechanism for returning control to the programme when the routine's operation is completed. The compiler routine combines the library data with the programmer's input codes according to the coding conventions associated with such interconnexion commands.

Selection and transfer of library routines must simulate the effects obtained by a human programmer using the control routine to incorporate parameters into routines. Provision must be made to transfer all routines involved in a problem, including all subordinate routines used by any superior routine involved. To economize on store space, any routine is to be transferred only once, even if involved more than once by the same problem.

During transfer, all routine commands referring to other locations in the store must be adjusted to refer correctly to new locations in $S_P$. This corresponds to operations of the control routine which add the address in which the first command of a routine is stored to those commands of the routine which are

coded with addresses relative to the first. Provision must be made to enable other parameters to be incorporated during transfer for the cases in which the parameters are supplied by the programmer or by superior library routines. In special cases provision can be made for parameters to modify the very process of transfer itself as is required in " unrolling " a repeated loop of commands into a linear sequence.

Addresses must be expressed in symbolic form if the compiler routine varies the number of commands in a section of the problem programme. For economy in store space the compiler routine may omit redundant interconnexion commands or insert transferred routines into the programme when they are first referred to. The programmer is unable to predict correct addresses of commands in such a programme and must therefore denote them by convenient symbols and let the compiler routine calculate the correct equivalent address when it has completed the variation of groups of commands.

The extra programming work imposed by use of " symbolic " addresses is rarely worth the small saving in store space obtained by suppressing redundant commands. It is simpler to adopt an invariant pattern of interconnexion commands including redundant commands and to avoid inserting " unpredictable " lengths of routine commands into the input programme by transferring routines after the programme is completely assembled. A process for organizing this will be subsequently illustrated.

Hyper-compiler routines, which construct programmes in hyper-code for problems involving elaborate operations provided by interpretive routines, must cater for transfer of the interpretive routines as well as of hyper-routines involved. Since interpretive systems enable inclusion of special operations as well as standard operations, the compiler must keep a record of all hyper-operations used by the problem and organize the transfer of all required interpretive routines and interconnect them to form an interpretive system.

## III. Conventions for Computer-code Compilers

The following conventions have been adopted for initial versions of computer-code compiler routines for the C.S.I.R.O. Mark I computer.

(1) *Notation.*—Stores are denoted by $S_L$ for storage of library routines and library data, $S_c$ for the compiler routine and working space, and $S_P$ in which commands of the problem programme are inserted. $R_x$, $R_y$, $R_z$ denote single-word registers of the arithmetic unit, while $x$, $y$, $z$ denote locations in stores, or the $D$ register, of problem data. " Transfer data " are to be stored in a block of locations starting at $t_0$ relative to which $t_c$ and $t_e$ are addresses in which is stored, or from which is extracted, a transfer datum. Similarly $i$ is the address of the first location of an " index block " in which the library data of the $m$th library routine are stored in location $i+m$.

$a_L$ is the address in $S_L$ of library data currently required and $a_P$ is the address in $S_P$ in which the next command will be stored. The particular $a_P$ corresponding to reference to the $m$th routine in the library is denoted by $a_P^m$ and that corresponding to the $t$th library routine transferred is denoted by $a_P^t$. The address in $S_P$ to which the " head " of a library routine is transferred is denoted by $h_P$

while $e$ denotes the address relative to the " head " at which a routine is to be " entered " by shifting control to $h_P+e$. A routine will have $n$ parameters, occupy $l+1$ store locations, and use $D_L$ ($L=15$, 14, etc.) as a " link register ".

(2) The entire input programme is assembled before transferring library routines. During compilation of the problem programme, transfer data are stored and are later used to specify which routines are to be subsequently transferred from $S_L$ to $S_P$. This avoids the necessity for symbolic addresses. For the same reason redundant commands are included in an invariant pattern of interconnexion commands.

(3) Conventional computer-code commands are assembled by the normal processes of the primary and control routine, but an extra control operation, $E$, is provided which transfers control to the compiler routine whenever $E$-designated compiler codes are input.

TABLE 1

COMPUTER-CODE CONVENTIONS FOR INTERCONNEXION COMMANDS

| Location in $S_P$ | Programme Commands | Input-code Commands | Library Data |
|---|---|---|---|
| $a_P-3$ | $(x)\rightarrow R_x$ | $(x)\rightarrow M$* | $R_x$ |
| $a_P-2$ | $(y)\rightarrow R_y$ | $(y)\rightarrow M$ | $R_y$ |
| $a_P-1$ | $(S)\rightarrow D_L$ | $(S)\rightarrow D_0$ | $L$ |
| $a_P$ | $h_P+e\rightarrow S$ | $m;e:$ $E$ | |
| $a_P+1$ | $(R_z)\rightarrow z$ | $(M)\rightarrow z$ | $R_z$ |

* The code $M$ is a zero or " blank " in the machine code, so that $(x)\rightarrow M$ is punched as $x;0,0X$.

(4) Table 1 indicates the conventional pattern of computer-code interconnexion commands for reference to a library routine which evaluates a function of the contents of registers $R_x$ and $R_y$ and stores the result in register $R_z$ when control is shifted to its $e$th command. The programme data occur in store locations $x$, $y$ and the result is to be stored in location $z$. The compiler routine must use the compiler code, $mp_{11}+ep_1\,;E$ (in which $E$ corresponds to tape code $7Y$ (cf. Table 3)) or in punched code $m;e,X:7Y$,* to organize transfer of the $m$th library routine to store locations starting from $h_P$ and then form the correct " cue command ", $h_P+e\rightarrow S$.

(5) When the interconnexion commands are first stored in $S_P$, the compiler code $mp_{11}+ep_1$ is held in location $a_P$. The address $a_P^m$ is stored in the $t_c$th location of the transfer data block, i.e. location $t_0+t_c$ of $S_c$. When the input programme is completely assembled the $t_e$th transfer datum from location $t_0+t_e$ is extracted for $0\leqslant t_e\leqslant$ final value of $t_c$. The transfer datum $a_P^m$ is then used to extract the corresponding compiler code and $m$, the library code, specifies the routine to be transferred. When $h_P$, the head address of the first command of the routine,

* For punching notation see Part I, p. 333 (Pearcey and Hill 1953*a*). This particular code may be written $m;e:E$.

is determined, it is combined with the entry code, $e$, to form the cue command $h_P+e\rightarrow S$, which is then inserted in location $a_P$ to complete the interconnexion commands.

(6) Routines involving subordinate routines are stored in the library with each cue command to a subordinate replaced by the corresponding compiler code, e.g. $m_s+e_s$. During transfer to $S_P$, each library command of the form $(S)\rightarrow D_r$ is detected as implying that the following library command is a compiler code. The compiler code is transferred unaltered to $S_P$ and its address $a_P$ is stored in the transfer data to ensure subsequent transfer of the subordinate routine.

(7) A library index is held in $S_c$ to enable the library code $m$ to be used to yield data specifying the location in $S_L$ of the corresponding library routine. Location $i+m$ of the index contains $a_Lp_{11}+np_2+tp_1$ in which $t=1$. When a routine requiring no parameters, $n=0$, is transferred, the index datum is replaced by $h_Pp_{11}+0p_1$ in which $t$ is given the fresh value of zero. This change is detected upon any subsequent reference to the same routine and is used as a criterion for avoiding a repetition of the transfer of the routine. In this case the cue command is formed and inserted as $h_P\rightarrow S$, thus completing the interconnexion commands correctly.

(8) The library data listed in Table 1 are packed in location $a_L$ as five-digit codes for the link number, $L$, and register codes $R_x$, $R_y$, $R_z$ in the form $Lp_{16}+R_yp_{11}+R_zp_6+R_xp_1$. For routines involving fewer registers, some register codes are zero and the programmer omits corresponding input-code commands. Thus for the operation $z=\log x$, involving one operand only, the second row of Table 1 is omitted, and for a " hoot " routine, which involves no operands and yields no results, the programmer supplies only $(S)\rightarrow D_0$ and the compiler code, while all register codes of the library datum are zero.

(9) The library data include specification of the length of the routine to be transferred by holding $lp_{11}$ in location $a_L+1$ and the commands of the library routine occur in locations $a_L+2$ to $a_L+l+2$.

(10) When referring to a library routine involving $n$ parameters, the programmer supplies the parameters in the input programme immediately preceding the compiler code at each reference of the routine. Library routines supply parameters for subordinate routines involving $n$ parameters by storing them immediately following the compiler code. In each case the parameters are transferred by the compiler routine to the transfer data block following the " cue address ", $a_P$, of the subordinate routine. The programmer may avoid a redundant second transfer of the same parametric routine by punching the entry code, equal to the number $t_c$ corresponding to the $a_P$ of the first reference involving the particular set of parameters. Otherwise the entry code for parametric routines is zero.

(11) Parameters are incorporated only into those commands which shift control to, or transfer data to or from, store locations. Such parametric commands have the form " $(M)\rightarrow ?$, $(?)\rightarrow M$, $(K)\rightarrow S$ ". The address digits $p_{18}$–$p_{20}$ of each parametric command are used to hold the number $p$ where $0\leqslant p\leqslant 7$, so that the $p$th parameter is added to the remainder of the command.

For $p=0$, the address, $h_P$, of the routine's first command in $S_P$ is added and therefore library parametric commands are coded with addresses relative to the head of each routine. This process corresponds to the use of control designations for incorporating parameters during input of commands.

Constants stored as pseudo-commands in library routines, whose binary codes would be confused with parametric commands, would be modified during transfer. Such constants are therefore assembled via the $H$ register as described in Part I (Pearcey and Hill 1953*a*, p. 330).

These conventions are summarized in Section IX from the point of view of the user, who is more concerned with the processes of constructing input tapes for library routines and problem programmes than with the principles of operation of compiler routines.

## IV. Example of Computer-code Compilers

A number of computer-code compiler routines have been designed to cater for various operations. Compiler routine $I1$ forms interconnexion commands according to conventions 4, 5, and 8, while compiler $I2$ caters for parametric routines as in convention 10. Transfer operations according to conventions 6 and 7 are provided by compiler $T1$, while up to seven parameters can be incorporated by compiler $T2$.

Compiler routines $I2$ and $T2$ are given in " schematic " form in Table 2 and in conventional computer coding in Table 3 for use in association with the problem programme, which, together with an outline of the library contents and resultant programme, is given in Table 4. In the schematic outline of Table 2, orders are to be performed serially unless otherwise indicated. In the equation notation used, $(x)$ implies " the current contents of location $x$ ", and new values are indicated by primes. Commas are used to indicate partitioning of 20-digit words into codes for symbols, most of which have been implicitly defined previously. Numbers in square brackets are addresses of computer-code commands in Table 3, corresponding to each schematic order of Table 2. The tape of $I2$ and $T2$ is read in and compiled into the second magnetic store $M''$ by standard methods.

The tape programme of Table 4 is designed to print values of

$$y=(\text{arc sin } x)/\cos x-\sum_{r=0}^{5} a_r x^r$$

for values of $x<0\cdot5$ read from tape punched in decimal code. Such a programme might be used, for instance, to examine goodness of fit of a polynomial to the function $(\text{arc sin } x)/\cos x$.

The primary, control, and compiler routines are transferred from $S_L=M''$, the second 1024-word magnetic drum store, to $S_c=M$ by switch operations of the console switchboard. The input tape listed in Table 4 is placed in the tape reader, all registers are cleared, and the computer started. The problem programme is compiled in $S_P=M'$ in locations 0′ to 29′ with compiler codes not yet replaced by cue commands and their addresses stored as transfer data in locations 250–257. When the command $110\rightharpoonup S,D$, punched at the end of the programme

tape, shifts control to compiler $T2$, the routines specified in the transfer data are transferred until $t_e=t_c$. Command 186 then returns control to the primary routine to read $p_{20}\rightarrow T,D$ punched at the end of the input tape and the computer

TABLE 2

SCHEMATIC OUTLINE OF COMPILER ROUTINES $I2$, $T2$

*Compiler Routine* I2

1. Extract $(i+m)=a_L;2n+t$ [40–45]
2. If $n$ is non-zero shift to 9. [46–50]
3. Set $(t_0+t_c)'=a_P^m$ and $t_c'=t_c+n+1$ [51–54]
4. Set $(a_P^m)'=m,e$ and extract $(a_L)=L$, $R_y$, $R_z$, $R_x$. [55–60]
5. Set $(a_P^m-1)'=L+(a_P^m-1)$ and $r'=a_P^m$ [61–71]
6. If $R_y$ is non-zero set $(a_P^m-2)'=R_y+(a_P^m-2)$ and $r'=r-1$ [72–82]
7. If $R_x$ is non-zero set $(r-2)'=R_x+(r-2)$ [83–90]
8. Set $(A)'=R_z$ code, $a_P'=a_P^m+1$, shift to the primary routine [91–94]
9. Set $r=n-1$ and $t_c'=t_c+n$ [95–98]
10. Set $(t_0+t_c)'=(a_P^m-1)$ and $t_c'=t_c-1$, $a_P^{m\prime}=a_P^m-1$, $r'=r-1$ [99–105]
11. If $r$ is negative shift to 3 otherwise to 10. [106–109]

*Compiler Routine* T2

12. Set $t_e'=0$ [110]
13. Extract $(t_0+t_e)=a_P^t$ and $(a_P^t)=m,e$ and $(i+c)=a_Lp_{11}+(2n+t)p_1$ or $h_Pp_{11}$ [111–123]
14. If $t=0$ shift to 24. [124–125]
15. Extract $(a_L+1)=l$. If $n\neq 0$ shift to 31. [126–132]
16. Set $(i+m)'=h_P$ (i.e. $t=0$, $n=0$) [133–135]
17. Set $(t_0+t_e)'=h_P$ [136–137]
18. Extract $(a_L+2)=$library command, set $a_L'=a_L+1$ [138–141]
19. If of form $(?)\rightarrow M$, $(M)\rightarrow ?$, $(K)\rightarrow S$, shift to 26. [142–153]
20. Set $(a_P)'=$library command and $a_P'=a_P+1$ [154–157]
21. If of form $(S)\rightarrow D_r$ shift to 27. [158–161]
22. Set $l'=l-1$. If $l>0$ shift to 18. [162–164]
23. Extract $(i+m)=h_P$ [165–168]
24. Set $(a_P^t)=h_P+e\rightarrow S$, $t_e'=t_e+n+1$ [169–178]
25. If $t_e\geqslant t_c$ shift to primary routine, otherwise to 13. [179–184]
26. Add $(t_0+t_e+p)$ to library command and shift to 20. [185–197]
27. Extract $(a_L+2)=m_s+e_s$, set $a_L'=a_L+1$, $(t_0+t_c)=a_P$, $t_c'=t_c+1$ [198–204]
28. Extract $(i+m_s)=a_L$, $n_s$, $t$ [205–207]
29. Set $n_s'=n_s-1$, if $n_s$ negative shift to 20. [208–210]
30. Set $(t_0+t_c)'=(a_L)$, $t_c'=t_c+1$, $a_L'=a_L+1$, shift to 29. [211–218]
31. If $e$ is zero, shift to 17. [219–222]
32. Extract $(t_0+e)=h_P$, set $n'=0$, shift to 24. [223–226]

Programme constants [227–231] Library index [232–237] Transfer data [250–300]

stops with the problem programme completely assembled in $M'$. This example illustrates :

(1) *The method of forming interconnexion commands.*—The first $E$-designated compiler code has $m=1$, which causes $(232+1)\equiv 285p_{11}+p_1$ to be extracted.

After $a_P^m=1$ is stored in location 250, and the compiler code 0,1;0,0 in 1′, the content of location 285″, i.e. 15,0,$A$,0, is extracted and combined with (0′) to form $(S)\rightarrow D_{15}$, while $(A)\rightarrow M$ is left in register $A$ to be combined with the next input command, forming $(A)\rightarrow 30$ in 2′.

(2) *Incorporation of parameters into library routines.*—After 18 programme commands are formed, the parameters are stored temporarily in locations 18′, 19′ before being transferred to locations 255, 256, whence they will be added during transfer of the polynomial routine to commands whose $p_{20}$–$p_{18}$ digits represent $p=1$, $p=2$ respectively.

(3) *Transfer of library routines.*—Library routines are held in $M''$. When the command $110\rightarrow S,D$ is performed, $t_e$ is started at zero and the process of transferring library routines begins. The transfer datum $(250)=1p_{11}$ is used to extract the compiler code 0,1;0,0 from location 1′, and this specifies the routine to be transferred; the input routine for which $m=1$. The corresponding library data $(i+m)\equiv(232+1)=285p_{11}+p_1$ and $(286'')=l-1\equiv 26$ are extracted and, after the current $a_P\equiv h_P$ is stored in locations 233, 250 of the index and transfer data blocks, the 27 commands, 287″–313″, are transferred to locations 33′–65′. Finally the cue command $h_P+e\rightarrow S\equiv 33\rightarrow S$ is placed in location 1′ and the compiler routine proceeds to transfer the next library routine required.

(4) *The method of ensuring transfer of library subordinate routines.*—The "arc sin" routine uses the "sinecosine" routine as a subordinate routine in a trial and error process. As the arc sin routine is transferred from 375″–392″ to 66′–83′, the command from 385″ is detected as having the form $(S)\rightarrow D$ and the compiler code 0,2;0,0 in location 386″, which implies a reference to the sinecosine routine, is transferred unaltered to location 77, which number is then stored as a transfer datum in location 258. If any parameters had been involved by the subordinate routine, they would have been transferred also.

(5) *Omission of redundant transfers.*—Following transfer of the "print" routine, the transfer datum $(258)=77p_{11}$, is used to extract the compiler code 0,2;0,0 from location 77′. Since the corresponding sinecosine routine has been transferred, $(232+2)=84p_{11}$ has zero $t$ and therefore the transfer is not repeated, while the cue command is correctly set as $84\rightarrow S$ in 77′.

## V. Extension of Computer-code Compiler Techniques

The similarity of the pattern of interconnexion commands for the compiler routine just described to that used by human programmers enables users to exploit the advantages of compiler techniques with very little extra training; the actual programming is more rapid and errors less frequent. The volume of tape punching in preparing input tapes can be reduced further by using a more elaborate compiler routine for forming interconnexion commands. In this way the number of keyboard operations involved by the pattern of Table 1 may be reduced by about 30 per cent. by using the punching pattern

$$x{:}y{:}z{:}m;e{:}E.,$$

which is transformed by the compiler routine to the equivalent of Table 1. The data addresses, $x$, $y$, $z$, can be made to refer to $D$ registers for values less

TABLE 3

COMPUTER-CODE COMMANDS OF PRIMARY, CONTROL, COMPILER *I2*, AND COMPILER *T2* ROUTINES (AS USED IN EXAMPLE IN TABLE 4)*

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $(0)\overset{c}{\rightarrow}S$ | 40 | $(A)\rightarrow D_4$ | 70 | $c(A)\rightarrow -1'$ | 100 | $(-1')\rightarrow A$ | 130 | $z(A)\overset{c}{\rightarrow}S$ | 160 | $z(A)\overset{c}{\rightarrow}S$ | 190 | $(A)\overset{\pm}{\rightarrow}A$ | 220 | $(Hu)\rightarrow A$ |
| 1 | $(C)\overset{\pm}{\rightarrow}K$ | 1 | $c(A)\rightarrow Hu$ | 1 | $(C)\rightarrow D_3$ | 1 | $(D_1)\overset{\pm}{\rightarrow}K$ | 1 | $p_1\overset{c}{\rightarrow}S$ | 1 | $198\rightarrow S$ | 1 | $(A)\overset{\pm}{\rightarrow}A$ | 1 | $z(A)\overset{c}{\rightarrow}S$ |
| 2 | $c(A)\rightarrow 0'$ | 2 | $(Hu)\overset{\pm}{\rightarrow}K$ | 2 | $(D_2)\rightarrow A$ | 2 | $c(A)\rightarrow 250$ | 2 | $219\rightarrow S$ | 2 | $p_{11}\overset{=}{\rightarrow}D_7$ | 2 | $(A)\overset{\pm}{\rightarrow}A$ | 2 | $136\rightarrow S$ |
| 3 | $p_{11}\overset{\pm}{\rightarrow}C$ | 3 | $(232)\rightarrow A$ | 3 | $0{,}31\overset{\cdot}{\rightarrow}A$ | 3 | $p_{11}\overset{=}{\rightarrow}D_1$ | 3 | $(D_4)\rightarrow Hu$ | 3 | $s(D_7)\overset{c}{\rightarrow}S$ | 3 | $(D_2)\overset{\pm}{\rightarrow}A$ | 3 | $(Hu)\overset{\pm}{\rightarrow}K$ |
| 4 | $(Z)\rightarrow B$ | 4 | $(A)\rightarrow Hu$ | 4 | $z(A)\overset{c}{\rightarrow}S$ | 4 | $p_{11}\overset{=}{\rightarrow}C$ | 4 | $(Hu)\overset{\pm}{\rightarrow}K$ | 4 | $138\rightarrow S$ | 4 | $c(A)\overset{\pm}{\rightarrow}K$ | 4 | $(250)\rightarrow A$ |
| 5 | $(D_0)\rightarrow Hl$ | 5 | $(Hu)\rightarrow D_5$ | 5 | $83\rightarrow S$ | 5 | $p_{11}\overset{=}{\rightarrow}D_3$ | 5 | $(C)\rightarrow 232$ | 5 | $D_4\rightarrow Hu$ | 5 | $250\rightarrow A$ | 5 | $D_6\overset{=}{\rightarrow}D_6$ |
| 6 | $(D_0)\overset{\pm}{\rightarrow}D_0$ | 6 | $\frac{1}{2}(A)\rightarrow Hl$ | 6 | $c(A)\rightarrow Hu$ | 6 | $s(D_3)\overset{c}{\rightarrow}S$ | 6 | $(D_2)\overset{\pm}{\rightarrow}K$ | 6 | $(Hu)\overset{\pm}{\rightarrow}K$ | 6 | $c(A)\overset{\pm}{\rightarrow}D_0$ | 6 | $170\rightarrow S$ |
| 7 | $s(D_0)\overset{c}{\rightarrow}S$ | 7 | $(Hu)\rightarrow A$ | 7 | $(Hl)\rightarrow A$ | 7 | $99\rightarrow S$ | 7 | $(C)\rightarrow 250$ | 7 | $(232)\rightarrow A$ | 7 | $154\rightarrow S$ | 7 | $\langle 0{,}0{,}0{,}31\rangle$ |
| 8 | $(0)\overset{c}{\rightarrow}S$ | 8 | $z(A)\overset{c}{\rightarrow}S$ | 8 | $(C)\overset{\pm}{\rightarrow}K$ | 8 | $(D_6)\rightarrow A$ | 8 | $(D_5)\overset{\pm}{\rightarrow}K$ | 8 | $(A)\rightarrow D_5$ | 8 | $(D_5)\overset{\pm}{\rightarrow}K$ | 8 | $\langle 0{,}0{,}31{,}0\rangle$ |
| 9 | $p_{11}\rightarrow B$ | 9 | $p_1\overset{c}{\rightarrow}S$ | 9 | $(-2')\overset{\pm}{\rightarrow}A$ | 9 | $51\rightarrow S$ | 9 | $(2'')\rightarrow A$ | 9 | $(D_5)\rightarrow A$ | 9 | $(2'')\rightarrow A$ | 9 | $\langle 0{,}0{,}31{,}31\rangle$ |
| 10 | $(Hl)\overset{\pm}{\rightarrow}A$ | 50 | $95\rightarrow S$ | 80 | $(C)\overset{\pm}{\rightarrow}A$ | 110 | $(D_2)\overset{=}{\rightarrow}D_2$ | 140 | $p_{11}\overset{\pm}{\rightarrow}D_5$ | 170 | $(D_4)\rightarrow Hl$ | 200 | $p_{11}\overset{\pm}{\rightarrow}D_5$ | 230 | $\langle 0\rightarrow S\rangle$ |
| 11 | $p_1\overset{c}{\rightarrow}S$ | 1 | $(D_1)\overset{\pm}{\rightarrow}K$ | 1 | $c(A)\rightarrow -2'$ | 1 | $(D_2)\overset{\pm}{\rightarrow}K$ | 1 | $(A)\rightarrow D_0$ | 1 | $(Hu)\overset{\pm}{\rightarrow}A$ | 1 | $(A)\rightarrow D_0$ | 1 | $\langle (S)\rightarrow D_0\rangle$ |
| 12 | $(Hu)\overset{\pm}{\rightarrow}A$ | 2 | $(C)\rightarrow 250$ | 2 | $p_{11}\overset{=}{\rightarrow}D_3$ | 2 | $(250)\rightarrow A$ | 2 | $(227)\overset{\cdot}{\rightarrow}A$ | 2 | $(230)\overset{\pm}{\rightarrow}A$ | 2 | $(D_1)\overset{\pm}{\rightarrow}K$ | 2 | $\langle 250p_{11}+p_1\rangle$ |
| 13 | $(I)\rightarrow D_0$ | 3 | $p_{11}\overset{\pm}{\rightarrow}D_1$ | 3 | $(D_2)\rightarrow A$ | 3 | $(A)\rightarrow D_3$ | 3 | $z(A)\overset{c}{\rightarrow}S$ | 3 | $(D_3)\rightarrow Hu$ | 3 | $(C)\rightarrow 250$ | 3 | $\langle 285p_{11}+p_1\rangle$ |
| 14 | $s(D_0)\overset{c}{\rightarrow}S$ | 4 | $c(A)\overset{\pm}{\rightarrow}D_1$ | 4 | $(227)\overset{\cdot}{\rightarrow}A$ | 4 | $c(A)\overset{\pm}{\rightarrow}K$ | 4 | $185\rightarrow S$ | 4 | $(Hu)\overset{\pm}{\rightarrow}K$ | 4 | $p_{11}\overset{\pm}{\rightarrow}D_1$ | 4 | $\langle 314p_{11}+p_1\rangle$ |
| 15 | $(B)\rightarrow S$ | 5 | $(D_4)\rightarrow A$ | 5 | $z(A)\overset{c}{\rightarrow}S$ | 5 | $(0')\rightarrow A$ | 5 | $(D_0)\rightarrow A$ | 5 | $c(A)\rightarrow 0'$ | 5 | $(A)\rightarrow Hu$ | 5 | $\langle 349p_{11}+p_1\rangle$ |
| 16 | $(Z)\rightarrow B$ | 6 | $(C)\overset{\pm}{\rightarrow}K$ | 6 | $91\rightarrow S$ | 6 | $(A)\rightarrow D_4$ | 6 | $(228)\overset{\cdot}{\rightarrow}A$ | 6 | $(D_6)\rightarrow Hu$ | 6 | $(Hu)\overset{\pm}{\rightarrow}K$ | 6 | $\langle 373p_{11}+p_1\rangle$ |
| 17 | $(D_0)\rightarrow Hl$ | 7 | $c(A)\rightarrow 0'$ | 7 | $(D_3)\overset{\pm}{\rightarrow}K$ | 7 | $c(A)\rightarrow Hu$ | 7 | $z(A)\overset{c}{\rightarrow}S$ | 7 | $Hu\overset{\pm}{\rightarrow}D_2$ | 7 | $(232)\rightarrow Hl$ | 7 | $\langle 393p_{11}+5p_1\rangle$ |

| | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | $(Hu)\overset{\pm}{\rightarrow}S$ | 8 | $(D_5)\overset{\pm}{\rightarrow}K$ | 8 | $(-2')\overset{\pm}{\rightarrow}A$ | 8 | $(Hu)\overset{\pm}{\rightarrow}K$ | 8 | $185\rightarrow S$ | 8 | $p_{11}\overset{\pm}{\rightarrow}D_2$ | 8 | $(Hu)\rightarrow A$ | | | | |
| 19 | $(29)\overset{\pm}{\rightarrow}A$ | 9 | $(0'')\rightarrow A$ | 9 | $(D_3)\overset{\pm}{\rightarrow}K$ | 9 | $(232)\rightarrow A$ | 9 | $(D_0)\rightarrow A$ | 9 | $(D_1)\rightarrow A$ | 9 | $2\overset{-}{\rightarrow}A$ | | | | |
| 20 | $(28)\overset{\pm}{\rightarrow}A$ | 60 | $(A)\rightarrow D_2$ | 90 | $c(A)\rightarrow -2'$ | 120 | $(A)\rightarrow Hu$ | 150 | $(229)\dot{\rightarrow}A$ | 180 | $(D_2)\overset{-}{\rightarrow}A$ | 210 | $s(A)\overset{c}{\rightarrow}S$ | | | | |
| 21 | $(27)\overset{\pm}{\rightarrow}A$ | 1 | $\frac{1}{2}(A)\rightarrow A$ | 1 | $(D_2)\rightarrow A$ | 1 | $(Hu)\rightarrow D_5$ | 1 | $(230)\overset{-}{\rightarrow}A$ | 1 | $s(A)\overset{c}{\rightarrow}S$ | 1 | $154\rightarrow S$ | | | | |
| 22 | $(19)\dot{\overset{\pm}{\rightarrow}}A$ | 2 | $\frac{1}{2}(A)\rightarrow A$ | 2 | $(228)\dot{\rightarrow}A$ | 2 | $\frac{1}{2}(A)\rightarrow Hl$ | 2 | $z(A)\overset{c}{\rightarrow}S$ | 2 | $111\rightarrow S$ | 2 | $(D_5)\overset{\pm}{\rightarrow}K$ | | | | |
| 23 | $c(A)\rightarrow 24$ | 3 | $\frac{1}{2}(A)\rightarrow A$ | 3 | $p_{11}\overset{\pm}{\rightarrow}C$ | 3 | $(Hu)\rightarrow D_6$ | 3 | $185\rightarrow S$ | 3 | $0\rightarrow B$ | 3 | $(2'')\rightarrow B$ | | | | |
| 24 | $[p_{20}\rightarrow T]$ | 4 | $\frac{1}{2}(A)\rightarrow A$ | 4 | $13\rightarrow S$ | 4 | $p_1.(A)\overset{c}{\rightarrow}S$ | 4 | $(D_0)\rightarrow A$ | 4 | $13\rightarrow S$ | 4 | $p_{11}\overset{\pm}{\rightarrow}D_5$ | | | | |
| 25 | $13\rightarrow S$ | 5 | $\frac{1}{2}(A)\rightarrow A$ | 5 | $(A)\rightarrow D_3$ | 5 | $169\rightarrow S$ | 5 | $(C)\overset{\pm}{\rightarrow}K$ | 5 | $(D_0)\rightarrow A$ | 5 | $(D_1)\overset{\pm}{\rightarrow}K$ | | | | |
| 26 | $40\rightarrow S$ | 6 | $0{,}15\dot{\rightarrow}A$ | 6 | $p_{11}\overset{-}{\rightarrow}D_3$ | 6 | $(D_5)\overset{\pm}{\rightarrow}K$ | 6 | $(A)\rightarrow 0'$ | 6 | $28{,}0\dot{\rightarrow}A$ | 6 | $(B)\rightarrow 250$ | | | | |
| 27 | $\langle 0{,}0{,}13{,}27\rangle$ | 7 | $(C)\overset{\pm}{\rightarrow}K$ | 7 | $(A)\overset{\pm}{\rightarrow}D_1$ | 7 | $(1'')\rightarrow A$ | 7 | $p_{11}\overset{\pm}{\rightarrow}C$ | 7 | $(A)\overset{-}{\rightarrow}D_0$ | 7 | $p_{11}\overset{\pm}{\rightarrow}D_1$ | | | | |
| 28 | $\langle 31{,}31{,}18{,}14\rangle$ | 8 | $(-1')\overset{\pm}{\rightarrow}A$ | 8 | $c(A)\rightarrow D_6$ | 8 | $(A)\rightarrow D_7$ | 8 | $(229)\dot{\rightarrow}A$ | 8 | $c(A)\rightarrow Hu$ | 8 | $209\rightarrow S$ | | | | |
| 29 | $\langle 31{,}4{,}26{,}0\rangle$ | 9 | $(C)\overset{\pm}{\rightarrow}K$ | 9 | $(C)\overset{\pm}{\rightarrow}K$ | 9 | $(D_6)\rightarrow A$ | 9 | $(231)\overset{-}{\rightarrow}A$ | 9 | $(Hl)\rightarrow A$ | 9 | $(D_4)\rightarrow Hl$ | | | | |

| Reg. | Contents |
|---|---|
| $C$ | $a_p p_{11}$ |
| $D_0$ | Input codes or library command |
| $D_1$ | $t_c p_{11}$ |
| $D_2$ | $L$, $R_y$, $R_z$, $R_x$ or $t_e p_{11}$ |
| $D_3$ | $np_{11}$ or $rp_{11}$ or $a_p^t p_{11}$ |
| $D_4$ | $mp_{11}+ep_1$ |
| $D_5$ | $a_L p_{11}$ or $hp_{11}$ |
| $D_6$ | $np_{11}$ |
| $D_7$ | $lp_{11}$ |

* *Explanatory Notes.*—(1) The notation used in this table and Table 7 is outlined in Parts I, II, and III of this series of papers (Pearcey and Hill 1953*a*, 1953*b*, 1954). (2) For convenience, the commands are given in the form in which they would occur after insertion into the computer (i.e. explicit addresses) rather than in the form in which they would be punched (i.e. using control operations on " symbolic " addresses). (3) In the case shown above, $i=232$ and $t_0=250$, so that the library index will have the values shown in locations 232–237. (4) To aid in reading the programme, a list is given of the way in which registers are used. Thus the $C$ register contains the address, $a_p p_{11}$, in which the current input programme command or library command is to be stored. (5) Primes denote addresses in magnetic stores I-IV. (6) This programme is held in the second magnetic store $M''$.

TABLE 4*

PROGRAMME COMPILATION FOR $y=(\arcsin x)/\cos x - \sum_{r=0}^{5} a_r x^r$

| Programme Commands | | | |
|---|---|---|---|
| No. | As Punched | As Compiled Initially | Final Form |
| 0 | 1S (S)→$D_0$ | (S)→$D_{15}$ | (S)→$D_{15}$ |
| 1 | 0,1;0,0. E | 0,1;0,0 | 33→S |
| 2 | 1A (M)→30 | (A)→30 | (A)→30 |
| 3 | 1A (30)→M | (30)→A | (30)→A |
| 4 | (S)→$D_0$ | (S)→$D_{14}$ | (S)→$D_{14}$ |
| 5 | 0,4;0,0. E | 0,4;0,0 | 66→S |
| 6 | 1A M)→31 | (A)→31 | (A)→31 |
| 7 | 1A (30)→M | (30)→A | (30)→A |
| 8 | (S)→$D_0$ | (S)→$D_{15}$ | (S)→$D_{15}$ |
| 9 | 0,2;0,1. E | 0,2;0,1 | 85→S |
| 10 | 1A (M)→32 | (A)→32 | (A)→32 |
| 11 | 1A (31)→M | (31)→A | (31)→A |
| 12 | 1A (32)→M | (32)→C | (32)→C |
| 13 | (S)→$D_0$ | (S)→$D_{15}$ | (S)→$D_{15}$ |
| 14 | 0,3;0,0. E | 0,3;0,0 | 117→S |
| 15 | 1A (M)→31 | (A)→31 | (A)→31 |
| 16 | 1A (30)→M | (30)→A | (30)→A |
| 17 | (S)→$D_0$ | (S)→$D_{15}$ | (S)→$D_{15}$ |
| | 0,5;0,0 | — | — |

Library Contents

| Compiler Routines | $a_L$ | $l$ | Comment |
|---|---|---|---|
| Primary | 0 | 16 | |
| Control | 16 | 14 | These routines are used to compile programmes using routines from a restricted library as listed below |
| Input parameters | 30 | 10 | |
| Compiler *I2* | 40 | 70 | |
| Compiler *T2* | 110 | 122 | |
| Library index | 232 | 18 | |

| $m$ | Library Routines | $a_L$ | $(a_L+1)$ | $(a_L)=L,R_y,R_z,R_x$ | $e$, $n$, etc. |
|---|---|---|---|---|---|
| 0 | Print | 250 | 32 | 15,0,0,*A* | $e=0,7$ |
| 1 | Input | 285 | 26 | 15,0,*A*,0 | $e=0$ |
| 2 | Sinecosine | 314 | 32 | 15,0,*A*,*A* | $e=0,1$ |
| 3 | Division | 349 | 21 | 15,*C*,*A*,*A* | $e=0$ |
| 4 | Arc sin | 373 | 17 | 14,0,*A*,*A* | $e=0$† |
| 5 | Polynomial | 393 | 11 | 15,0,*A*,*A* | $n=2$ |

† Command in library, location 385 is (S)→$D_{15}$, is followed in location 386 by 0,2;0,0, the compiler code of the subordinate sinecosine routine for which $m=2$

| | | | | | Transfer Data | | | |
|---|---|---|---|---|---|---|---|---|
| | 1*A* 0,24;0,0 | | — | — | $t_0+t_c$ | $(t_0+t_c)'=a_P$ | $(t_0+t_e)'=h_P$ | Routine |
| 18 | 0,5;0,0. | *E* | 0,5;0,0 | 139→*S* | 250 | $1p_{11}$ | $33p_{11}$ | Input |
| 19 | 1*A* (*M*)→32 | | (*A*)→32 | (*A*)→32 | 251 | $5p_{11}$ | $66p_{11}$ | Arc sin |
| 20 | 1*A* (32)→*M* | | (32)→*A* | (32)→*A* | 252 | $9p_{11}$ | $84p_{11}$ | Sinecosine |
| 21 | (*S*)→$D_0$ | | (*S*)→$D_{15}$ | (*S*)→$D_{15}$ | 253 | $14p_{11}$ | $117p_{11}$ | Division |
| 22 | 0,0;0,7. | *E* | 0,0;0,7 | 158→*S* | 254 | $18p_{11}$ | $139p_{11}$ | Polynomial |
| 23 | 1*A* 0→*S* | | 0→*S* | 0→*S* | 255 | $n=5p_{11}$ | $n=5p_{11}$ | —— |
| 24–29 | $a_0$ to $a_5$ | | $a_0$ to $a_5$ | $a_0$ to $a_5$ | 256 | $h=24p_{11}$ | $h=24p_{11}$ | —— |
| 30 | (*M*)→0 | | blank space | working space | 257 | $22p_{11}$ | $151p_{11}$ | Print |
| 31 | (*M*)→0 | | | | 258 | $77p_{11}$ | — | [Already transferred] |
| 32 | (*M*)→0 | | | | | | | |
| | 110→*S D* | | | | | | | |
| | $p_{20}$→*T D* | | | | | | | |

* This programme is compiled by *I*2 and *T*2 of Table 3 and placed by them into the first magnetic store *M'* in the locations indicated. Control designations *nS*, *nA* are accepted by *I*2*T*2; in this example 1*S* stores zero.

than 16, otherwise to serial store locations, and full use of input control designations can be made in assigning these addresses.

In programmes involving fewer than 32 data elements, the tape punch pattern can be reduced to 35 per cent. of the original number of keyboard operations, by the codes,

$$x,y;z,m{:}E \text{ and } e=0,$$

where $x,y,z,m$ are five-digit codes and the three-address operation, assembled by the primary routine as a single command, is transformed by the compiler routine to the five commands of Table 1. In this system, use of input control operations is very restricted.

These reductions of keyboard operations require compiler routines with elaborate decoding operations, in which each three-address command is counted as five computer-code commands for the purpose of numbering programme commands. If such a scheme is used consistently, the programmer need not be familiar with normal programming methods but only with the special three-address methods.

The techniques used in transferring library routines can be extended to cater for modification by a parameter, of the process of transfer. This would be required to unroll a repeated loop, such as a polynomial routine, into a linear sequence of commands. The programmer would supply the parameters, $n$ (number of coefficients) and $h$ (address of the first coefficient), and mark the $e$ code with distinctive digits such as $p_{10}$, $p_9$. The schematic commands numbered 23 and 31 of Table 2 would have the form:

23. If $e$ has $p_{10}$ shift to 36, otherwise extract $(i+m)=h_P$
31. If $e$ has $p_9$ shift to 33, if $e$ is zero shift to 17

which transfers control to an extra sequence of commands. This sequence transfers coefficients from the library if $e$ has $p_9$ and then unrolls the library routine by transferring $l_1+1$ commands and transferring $l_2+1$ commands $n$ times, increasing the parameter $h$ by unity each time and finally transfers the terminating sequence of length $l_3+1$. The library data $l_1$, $l_2$, $l_3$ are stored in $a_L+1$, $a_L+2$ with the first command of the routine in $a_L+3$. It is convenient to use portion of compiler routine $T2$ by replacing $l$ and $a_L$ of Table 2 by appropriate values, as in the following schematic form of an additional sequence of compiler commands,

33. If $e$ has $p_9$ transfer $(h+r)$ to $a_P+r$ for $0\leqslant r<n$ and set $h'=a'_P\equiv a_P+n$
34. Extract $l_1$, $l_2$, $l_3$, set $l'=l_1$ and shift to 17.
35. Set $l'=l_2$, $a'_L=a_L-l_2-1$, $n'=n-1$, $h'=h+1$, if $n>0$ shift to 18.
36. Set $l'=l_3$, $e'=0$ shift to 18

after which command 23 detects $e=0$ and continues compilation normally.

Computer-code compiler techniques can be extended to cover quite a wide range of problems by extending the number of library routines. Less than 500 store locations would be required for routines for printing fractions or integers, reading decimally punched fractions or integers, division yielding fraction or integer results, square root, cube root, sinecosine, tangent, arc cos, arc sin,

exponential, logarithm to the base 2 or e, and hyperbolic functions. The rest of the 1024 store locations of one only of the magnetic drum stores of the C.S.I.R.O. computer may be devoted to the primary, control, and compiler routines and a battery of test routines for testing units of the computer, i.e. arithmetic unit test, store test, etc.

## VI. Conventions for Hyper-code Compiler Routines

Hyper-code commands are interpreted by a system of computer-coded routines, which provide an elaborate arithmetical or organizational operation corresponding to each hyper-code selected and interpreted by an interpretation system. As indicated in Part III (Pearcey and Hill 1954), standard operations are provided which include addition, subtraction, multiplication, division, and square root, as well as simple transfers, printing, and input of numbers according to different systems of representation. Useful systems of interpretive routines have been developed for elaborate arithmetics using floating decimal, double precision, and complex number representations, etc.

Additional hyper-operations of the form $f(\bar{A})\rightarrow\bar{A}$ are provided by hyper-commands of the ETC type,* coded as $n$;15,$Z$, which shift control to additional computer-coded routines starting at location $n$. Sequences of hyper-operations are obtained by hyper-routines which are invoked by hyper-commands of the CUE type, coded as $n$;10,$Z$, which shift hyper-control to location $n$. After the hyper-routine is performed, the hyper-routine returns hyper-control to its superior routine by a LINK-type hyper-command, coded as $n$;11,$Z$.

Interpretive techniques have the effect of providing the user with a " hyper-computer " operating as though the hyper-code were the new computer code. Compiler routines for hyper-code programmes must not only form the hyper-programme with its associated hyper-routines, but also, in effect, construct the hyper-computer by compiling the necessary function block and any additional routines of the ETC type. For this dual process the following conventions have been adopted for initial versions of hyper-code compiler routines for the C.S.I.R.O. Mark I computer.

(1) *Notation.*—It is convenient to distinguish those symbols referring to store locations and operations associated with the interpreted hyper-programme by the superscript $H$ and those associated with the interpreting routines, including the function block and additional ETC routines, by the superscript $I$.

(2) Again, the complexity of symbolic addresses is avoided by including even redundant commands in the interconnexion pattern and by using transfer data stored during assembly of the input programme to organize transfer of library routines involved. The hyper-commands assembled from the tape and transferred hyper-routines are inserted in store space $S_P^H$, while the interpretation, function block, and ETC routines, are transferred to $S_P^I$, leaving space for data storage in $S_P^I$. When this process is complete the interpretive system is trans-

* Hyper-commands of types ETC, CUE, and LINK correspond to those correspondingly coded and concerned with programme organization as described in Part III (Pearcey and Hill 1954).

ferred from $S_P^I$ to $M$, the rapid-access store, all registers are cleared, and the computer started. Hyper-commands will be extracted from $S_P^H$ by the interpretation routine in $S_P^I$ and decoded to refer to data locations and routines for hyper-operations in $S_P^I$.

(3) The primary and control routines are used to input conventional computer-code and hyper-code commands and shift control to compiler routines whenever $E$-designated compiler codes are detected. Two types of compiler codes are $E$-designated; namely, CUE-type codes punched as $m,e;10,Z$, which refers to the hyper-operation provided by the $m$th hyper-routine in $S_L^H$, and ETC-type codes punched as $m,e;15,Z$, which refers to the hyper-operation provided by the $m$th special interpretive routine in $S_L^I$.

The process of punching data-address codes of hyper-commands is simplified by modifying the conventions outlined in Section XV of Part III (Pearcey and Hill 1954) which required post-punched $P$ and $Q$ designations to adjust data addresses to refer to $p$-word and $q$-word data groups respectively. The task of adjusting data addresses has been transferred to the interpretation procedure by including extra commands in interpretation routines to multiply stored data addresses by $p$ or $q$ and add the address of the head of the data groups. Consequently data addresses may be coded and stored as numbers of the set 0, 1, 2, 3, . . ., etc.

TABLE 5

HYPER-CODE CONVENTIONS FOR INTERCONNEXION COMMANDS FOR $z=f(x,y)$

| Location in $S_P^H$ | Programme Commands | | Library Data |
|---|---|---|---|
| | Input (Hyper-code) | Compiled | |
| $a_P^H-4$ | $(x)\rightarrow\bar{A}$ | $(x)\rightarrow\bar{A}$ | |
| $a_P^H-3$ | $(\bar{A})\rightarrow 0^*$ | $(\bar{A})\rightarrow R_x$ | $R_x$ |
| $a_P^H-2$ | $(y)\rightarrow\bar{A}$ | $(y)\rightarrow\bar{A}$ | |
| $a_P^H-1$ | $(\bar{A})\rightarrow 0$ | $(\bar{A})\rightarrow R_y$ | $R_y$ |
| $a_P^H$ | $m,e\rightarrow\bar{L}$ $E\dagger$ | $h+e\rightarrow\bar{L}$ | |
| $a_P^H+1$ | $(0)\rightarrow\bar{A}$ | $(R_z)\rightarrow\bar{A}$ | $R_z$ |
| $a_P^H+2$ | $(\bar{A})\rightarrow z$ | $(\bar{A})\rightarrow z$ | |

* The symbol " 0 " denotes a " blank " or zero address. $(\bar{A})\rightarrow 0$ is punched $8,Z$:
† This is punched $m,e;10,Z:7Y$.

(4) No modification of the input hyper-command pattern is required for ETC-type library references but in the case of CUE-type library references to library hyper-routines the interconnexion commands must be constructed from input commands and library data as is illustrated in Table 5. Similar patterns are obtained for operands requiring fewer addresses for operands or data by omitting unnecessary rows.

(5) When the interconnexion commands are first formed in $S_P^H$, the CUE-type compiler code, $m,e\rightarrow\bar{L}$, is stored unaltered in location $a_P^H$ and its address is stored in location $t_0^H+t_c^H$ of a transfer data block referring to hyper-routines.

Similarly each ETC-type compiler code, $m,e;15,Z$, is stored unaltered and its address $a_P^I$ is stored in location $t_0^H+t_c^I$ of another transfer data block, where $0 \leqslant t_c^H < t_0^I \leqslant t_c^I < 40$ say.

When the input programme is completely assembled, the transfer data of the type $a_P^H$ are used to select hyper-routines to be transferred from $S_L^H$ to $S_P^H$, and when this is complete the standard interpretive routines, together with additional interpretive routines determined by the transfer data $a_P^I$, are transferred from $S_L^I$ to $S_P^I$. When each hyper-routine or ETC routine is transferred, the address of its first command is known and used to form $h+e\rightarrow\bar{L}$ or $h+e\rightarrow\bar{S}$ to replace the compiler codes, $m,e\rightarrow\bar{L}$ or $m,e\rightarrow\bar{S}$ in locations $a_P^H$ or $a_P^I$ respectively.

The same compiler routine is used for this purpose with certain commands changed to alter the reference from " hyper " to " interpretive ".

(6) Hyper-routines involving either subordinate library hyper-routines or ETC routines are stored in the library $S_L^H$ with CUE commands and ETC commands replaced by appropriate compiler codes. During transfer the address in $S_P^H$ to which such a compiler code is transferred is included in the appropriate transfer block to ensure transfer of all subordinate hyper-routines or ETC routines involved.

(7) With each library is associated an index in $S_c$ consisting of a list of $a_L^H p_{11}+kp_9+np_2+tp_1$ in locations $i^H+m$ for hyper-routines, and for the non-parametric ETC routines a list of $a_L^I p_{11}+tp_1$ in location $i^I+c$, where $a_L^H$ and $a_L^I$ are the addresses of the $m$th routine in the libraries in $S_L^H$ and $S_L^I$ respectively. As previously, $n$ is the number of parameters and $t$ is a sign that the routine has been transferred. The extra symbol, $k$, denotes a number having a $p_{10}$ digit if the hyper-operation does not involve $R_y$ and a $p_9$ if $R_z$ is not involved.

When a non-parametric routine, with $n=0$, is transferred, the index datum is replaced by $h_P p_{11}+0p_1$ and detection of zero $t$ for any subsequent reference to the same routine enables the compiler code to be replaced by $h_P+e\rightarrow\bar{L}$ or $h_P+e\rightarrow\bar{S}$ without a second transfer of the routine involved.

(8) Hyper-routines can be standardized to use data locations 0, 1, 2 as registers $R_x$, $R_y$, $R_z$, thus reducing the library data to the length specification $lp_{11}$ in location $a_L^H$ and the number corresponding to $k$ incorporated in the index data. If $k$ has no $p_{10}$, the compiler routine adds $R_y=1$ to the command in location $a_P^H-1$ and if $k$ has no $p_9$ the compiler routine adds $R_z=2$ to the command in location $a_P^H+1$, while if $R_x$ is involved no modification is required to the corresponding input command.

(9) ETC routines provide operations of the form $f(\bar{A})\rightarrow\bar{A}$ so that problem parameters are involved only by provision of various points of entry into the routine. Since ETC routines may require to use operations provided by routines of the interpretive function block, provision is made for incorporation of the head parameter of the standard interpretation block of routines as well as that of the ETC routine itself for $p=1$, 0 respectively. Since only the register $\bar{A}$ is involved, the library data are reduced to the length specification $lp_{11}$ stored in location $a_L^I$ where the first routine command is in location $a_L^I+1$.

(10) As in the case of computer-code compilers, the convention for incorporation of parameters requires that parameters be provided immediately preceding the compiler code in the case of input programmes, and in the case of library hyper-routines which refer to subordinate hyper-routines, any parameters are stored following the compiler code. In both cases, the parameters are to be transferred to the transfer block, whence they are incorporated into commands of the hyper-routine when it is subsequently transferred. To avoid redundant transfer for further references to the same parametric hyper-routine, the entry code is punched equal to the value $t_c^H$ had at the time of first reference to the routine.

(11) Since it is found that hyper-commands of library routines have the $p_{20}$ digit zero, it is convenient to distinguish those hyper-commands into which parameters are to be incorporated by storing them in the library with their $p_{20}$ digit unity. The $p_{19}$–$p_{17}$ digits of such commands are used to represent $p$, where the $p$th parameter associated with the hyper-routine is to be incorporated into the command.

## VII. Example of Hyper-code Compiler

A compiler routine for hyper-code programmes, compiler $H2$, is given in schematic form in Table 6 and in conventional computer coding in Table 7, together with library data for use in association with the problem programme outlined in Table 8. This hyper-programme is designed to print values of

$$y=(\sin x)/(\arccos x)-\sum_{r=0}^{8} a_r x^r$$

for values of $x<0\cdot5$ read from tape punched in decimal code. It is intended for use in association with an interpretive system for double precision arithmetic.

The library of hyper-routines is to be held in $S_L^H \equiv M''$ from location 300″ onwards, while the interpretive routines are to be held in $S_L^I \equiv M^{\text{iv}}$ in locations as listed in Table 7. The primary, control, and compiler routines are to be transferred from $S_L^H \equiv M''$ to $S_c \equiv M$ by switch operations of the console switchboard. The input tape is placed in the tape reader, all registers cleared, and the computer started. The problem hyper-programme will be compiled in locations 0′–21′ of $S_P^H \equiv M'$ and transfer data will be placed in locations 300–303 and 322–323 as indicated in Table 7.

The command $101 \rightarrow S,D$, punched following the problem programme (Table 8), shifts control to the group of commands 101–246, which are designed for transfer of hyper-routines if $D_8$ has no $p_{20}$ and for transfer of interpretive routines after $p_{20}$ is set in $D_8$ by command 249. In this case, $D_8$ holds $t_c^I$, which has no $p_{20}$, and the transfer data in 300–303 are used to transfer hyper-routines from 314″–325″ to 22′–33′ and from 327″–334″ to 34′–41′. In the latter case parameters in 302, 303 are incorporated into the hyper-commands transferred to 34′ and 37′. At the conclusion of this transfer, $t_e$ exceeds $t_c^H$ and control is returned to the primary routine.

The control designation $50T$, read from tape, sets $a_P^I$ equal to $50p_{11}$, thus leaving space for working data. The command $247 \rightarrow S,D$ shifts control to a

TABLE 6

SCHEMATIC OUTLINE OF COMPILER $H2$ FOR HYPER-PROGRAMMES

1. Set $t_c^I = t_0^I - t_0^H$ (i.e. $20p_{11}$) and erase this command [40–42]
2. If $(A)$ is a cue command, shift to 4. [43–47]
3. Set $(t_0^I + t_c^I)' = a_P^I$, $t_c^{I\prime} = t_c^I + 1$ and shift to primary routine command 1. [48–54]
4. Extract $(i^H + m) = a_L p_{11} + k p_9 + n p_2 + t p_1$. If $n$ is zero shift to 8. [55–69]
5. Set $r = n - 1$, $t_c^{H\prime} = t_c^H + n$ [70–72]
6. Set $(t_0^H + t_c^H)' = (a_P^H)$, $t_c^{H\prime} = t_c^H - 1$, $a_P^{H\prime} = a_P^H - 1$, $r' = r - 1$ [73–79]
7. If $r$ not negative shift to 6. [80–81]
8. Set $(t_0^H + t_c^H)' = a_P^H$, $t_c^{H\prime} = t_c^H + n + 1$ [82–85]
9. Set $(a_P^H)'$ = cue command, $a_P^{H\prime} = a_P^H + 1$ [86–89]
10. If $k$ has no $p_{10}$, set $(a_P^H - 2)' = (a_P^H - 2) + p_{11}$ [90–95]
11. If $k$ has no $p_9$ set $2p_{11}$ in $A$ to be added to next command [96–98]
12. Shift to primary to continue assembly of input commands [99–100]
13. Entered by $101 \rightarrow S,D$ following input programme; sets $t_e = 0$ [101]
14. Extract $(t_0^H + t_e) = a_P^H$ {or $a_P^I$}, $(a_P^H) = m,e$; CUE {or $(a_P^I) = m,e$; ETC} [102–107]
15. Extract $(i^H + m) = a_L p_{11} + k p_9 + n p_2 + t p_1$ {or $(i^I + m) = a_L; t$} [108–120]
16. Store $a_L$ and $n$ and if $t = 0$ shift to 26. [121–127]
17. Extract $(a_L) = l$. If $n$ zero shift to 20. [128–133]
18. If $l$ is zero, shift to 21. [134–137]
19. Extract $(t_0^H + e) = h_P^H$, set $n' = 0$ and shift to 27. [138–141]
20. Set $(i^H + m)' = h_P^H$ {or $(i^I + m)' = h_P^I$} [142–143]
21. Set $(t_0^H + t_c^H)' = h_P^H$ {or $(t_0^I + t_c^I)' = h_P^I$, $(t_0^I + t_c^I + 1)' = h_0$, $n' = 1$} [144–151]
22. Set $a_L' = a_L + 1$, extract $(a_L) \equiv$ library command if hyper-command, shift to 28. [152–159]
23. If of the form $? \rightarrow M$, $(M) \rightarrow ?$, $K \rightarrow S$, shift to 31. [160–171]
24. Set $(a_P)' =$ library command and $a_P' = a_P + 1$. [172–175]
25. Set $l' = l - 1$. If $l > 0$ shift to 22. [176–178]
26. Extract $(i^H + m) = h_P^H$ {or $(i^I + m) = h_P^I$} [179–180]
27. Set $(a_P^H)' = h_P^H + e \rightarrow I$ {or $(a_P^I)' = h_P^I + e \rightarrow S$} [181–183]
28. Set $t_e' = t_e + n + 1$. If $t_e \geqslant t_c$, shift to primary, otherwise to 14. [184–192]
29. If CUE type shift to 33, if ETC type shift to 32. [193–200]
30. If non-parametric shift to 24, otherwise shift $p$ to $p_{20}$–$p_{18}$ digits [201–204]
31. Add $(t_0 + t_e + p)$ to library command, shift to 24. [205–218]
32. Set $(t_0^I + t_c^I)' = a_P^I$ and shift to 24. [219–222]
33. Set $(t_0^H + t_c^H)' = a_P^H$, extract $(i^H + m_s) = a_L p_{11} + k p_9 + n_s p_2 + t p_1$ [223–236]
34. Set $n_s' = n_s - 1$, if $n_s$ negative shift to 24. [237–239]
35. Set $(t_0^H + t_c^H)' = (a_L)$, $t_c^{H\prime} = t_c^H + 1$, $a_L' = a_L + 1$ and shift to 34. [240–246]
36. Entered by $nT$, $247 \rightarrow S,D$ from tape; set $t_c' = t_c^I$, $t_c^{I\prime} = p_{20}$, $t_e' = t_0^I - t_0^H$ [247–251]
37. Adjust references of 174 to $S_P^I$, 129,154,242 to $S_L^I$, set first command in $S_P^I$, shift to 15. [252–262]

Programme constants [263–272] Library index [273–275, 276–279]

TABLE 7

COMPILER ROUTINE $H2$ IN CONVENTIONAL COMMAND FORM*

| | | 70 | 100 | 130 | 160 | 190 | 220 | 250 |
|---|---|---|---|---|---|---|---|---|
| 40 | $22 \rightarrow Hu$† | $(A) \rightarrow D_3$ | $13 \rightarrow S$ | $(A) \rightarrow D_7$ | $(267) \overset{\pm}{\rightarrow} A$ | $102 \rightarrow S$ | $(C) \rightarrow 300$ | $20 \rightarrow Hu$ |
| 1 | $(Hu) \rightarrow D_8$ | $p_{11} \overset{=}{\rightarrow} D_3$ | $(D_2) \overset{=}{\rightarrow} D_2$ | $(D_6) \rightarrow A$ | $z(A) \overset{c}{\rightarrow} S$ | $(C) \rightarrow D_3$ | $p_{11} \overset{\pm}{\rightarrow} D_8$ | $(Hu) \rightarrow D_2$ |
| 2 | $(Z) \rightarrow 41$ | $(A) \overset{\pm}{\rightarrow} D_1$ | $(D_2) \overset{\pm}{\rightarrow} K$ | $z(A) \overset{c}{\rightarrow} S$ | $205 \rightarrow S$ | $99 \rightarrow S$ | $172 \rightarrow S$ | $(261) \rightarrow A$ |
| 3 | $(A) \rightarrow D_4$ | $(C) \overset{\pm}{\rightarrow} K$ | $(300) \rightarrow A$ | $142 \rightarrow S$ | $(D_0) \rightarrow A$ | $(D_0) \rightarrow A$ | $(D_1) \overset{\pm}{\rightarrow} K$ | $c(A) \rightarrow 174$ |
| 4 | $(264) \overset{\cdot}{\rightarrow} A$ | $(-1') \rightarrow B$ | $c(A) \rightarrow D_3$ | $(D_4) \rightarrow A$ | $(264) \overset{\cdot}{\rightarrow} A$ | $(264) \overset{\cdot}{\rightarrow} A$ | $(C) \rightarrow 300$ | $(271) \rightarrow A$ |
| 5 | $(268) \overset{=}{\rightarrow} A$ | $(D_1) \overset{\pm}{\rightarrow} K$ | $(D_3) \overset{\pm}{\rightarrow} K$ | $0,31 \overset{\cdot}{\rightarrow} A$ | $z(A) \overset{c}{\rightarrow} S$ | $(268) \overset{=}{\rightarrow} A$ | $p_{11} \overset{\pm}{\rightarrow} D_1$ | $(A) \rightarrow 129$ |
| 6 | $z(A) \overset{c}{\rightarrow} S$ | $(B) \rightarrow 300$ | $(0') \rightarrow A$ | $z(A) \overset{c}{\rightarrow} S$ | $205 \rightarrow S$ | $z(A) \overset{c}{\rightarrow} S$ | $(D_0) \rightarrow A$ | $c(A) \rightarrow 154$ |
| 7 | $55 \rightarrow S$ | $p_{11} \overset{=}{\rightarrow} D_1$ | $(A) \rightarrow D_4$ | $144 \rightarrow S$ | $(D_0) \rightarrow A$ | $223 \rightarrow S$ | $\frac{1}{2}(A) \rightarrow A$ | $(272) \rightarrow A$ |
| 8 | $(D_8) \overset{\pm}{\rightarrow} K$ | $p_{11} \overset{=}{\rightarrow} C$ | $31,0 \overset{\cdot}{\rightarrow} A$ | $c(A) \overset{\pm}{\rightarrow} K$ | $(265) \overset{\cdot}{\rightarrow} A$ | $(269) \overset{=}{\rightarrow} A$ | $\frac{1}{2}(A) \rightarrow A$ | $c(A) \rightarrow 242$ |
| 9 | $(C) \rightarrow 300$‡ | $p_{11} \overset{=}{\rightarrow} D_3$ | $(A) \overset{=}{\rightarrow} D_4$ | $(300) \rightarrow A$ | $(266) \overset{=}{\rightarrow} A$ | $z(A) \overset{c}{\rightarrow} S$ | $\frac{1}{2}(A) \rightarrow A$ | $(C) \rightarrow A$ |
| 50 | $p_{11} \overset{\pm}{\rightarrow} D_8$ | $s(D_3) \overset{c}{\rightarrow} S$ | $\frac{1}{2}(A) \rightarrow A$ | $D(_6) \overset{=}{\rightarrow} D_6$ | $z(A) \overset{c}{\rightarrow} S$ | $219 \rightarrow S$ | $\frac{1}{2}(A) \rightarrow A$ | $(266) \overset{\pm}{\rightarrow} A$ |
| 1 | $p_{11} \overset{\pm}{\rightarrow} D_8$ | $73 \rightarrow S$ | $\frac{1}{2}(A) \rightarrow A$ | $181 \rightarrow S$ | $205 \rightarrow S$ | $(D_0) \rightarrow A$ | $\frac{1}{2}(A) \rightarrow A$ | $c(A) \rightarrow 0'''$ |
| 2 | $(D_4) \rightarrow A$ | $(D_1) \overset{\pm}{\rightarrow} K$ | $\frac{1}{2}(A) \rightarrow A$ | $(D_9) \overset{\pm}{\rightarrow} K$ | $(D_0) \rightarrow A$ | $s(D_0) \overset{c}{\rightarrow} S$ | $0,31 \overset{\cdot}{\rightarrow} A$ | $107 \rightarrow S$ |
| 3 | $(D_0) \overset{=}{\rightarrow} D_0$ | $(C) \rightarrow 300$ | $\frac{1}{2}(A) \rightarrow A$ | $(C) \rightarrow 276$ | $(C) \overset{\pm}{\rightarrow} K$ | $172 \rightarrow S$ | $c(A) \overset{\pm}{\rightarrow} K$ | $\langle 0,0,0,31 \rangle$ |
| 4 | $1 \rightarrow S$ | $p_{11} \overset{\pm}{\rightarrow} D_1$ | $\frac{1}{2}(A) \rightarrow A$ | $(D_2) \overset{\pm}{\rightarrow} K$ | $[c(A) \rightarrow 0']$ | $30,0 \overset{\cdot}{\rightarrow} A$ | $(273) \rightarrow Hl$ | $\langle 0,0,31,0 \rangle$ |
| 5 | $(D_4) \rightarrow A$ | $c(A) \overset{\pm}{\rightarrow} D_1$ | $0,31 \overset{\cdot}{\rightarrow} A$ | $(C) \rightarrow 300$ | $p_{11} \overset{\pm}{\rightarrow} C$ | $(D_6) \overset{\pm}{\rightarrow} A$ | $(Hu) \rightarrow A$ | $\langle 0,0,31,31 \rangle$ |
| 6 | $\frac{1}{2}(A) \rightarrow A$ | $(D_4) \rightarrow A$ | $s(D_8) \overset{c}{\rightarrow} S$§ | $s(D_8) \overset{c}{\rightarrow} S$ | $p_{11} \overset{=}{\rightarrow} D_7$ | $(A) \rightarrow D_0$ | $7,30 \overset{\cdot}{\rightarrow} A$ | $\langle 0 \rightarrow S \rangle$ |
| 7 | $\frac{1}{2}(A) \rightarrow A$ | $(C) \overset{\pm}{\rightarrow} K$ | $3 \overset{=}{\rightarrow} A$¶ | $152 \rightarrow S$ | $s(D_7) \overset{c}{\rightarrow} S$ | $28,0 \overset{\cdot}{\rightarrow} A$ | $2 \overset{=}{\rightarrow} A$ | $\langle (0) \rightarrow Z \rangle$ |
| 8 | $\frac{1}{2}(A) \rightarrow A$ | $c(A) \rightarrow 0'$ | $c(A) \rightarrow D_9$ | $(320) \rightarrow A$ | $152 \rightarrow S$ | $(A) \overset{=}{\rightarrow} D_0$ | $s(A) \overset{c}{\rightarrow} S$ | $\langle 0,0,10,0 \rangle$ |
| 9 | $\frac{1}{2}(A) \rightarrow A$ | $p_{11} \overset{\pm}{\rightarrow} C$ | $(D_9) \overset{\pm}{\rightarrow} K$ | $(D_2) \overset{\pm}{\rightarrow} K$ | $(D_9) \overset{\pm}{\rightarrow} K$ | $c(A) \rightarrow Hu$ | $172 \rightarrow S$ | $\langle 0,0,5,0 \rangle$ |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 60 | $\frac{1}{2}(A)\rightarrow A$ | $s(D_5)\overset{c}{\rightarrow}S$ | $(276)\rightarrow A$ | $c(A)\rightarrow 301$ | $(273)\rightarrow A$ | $(Hl)\rightarrow A$ | $p_{11}\overset{\pm}{\rightarrow}D_5$ | $\langle 0,0,7,31\rangle$ |
| 1 | $0,31\dot{\rightarrow}A$ | $p_{11}\rightarrow A$ | $(A)\rightarrow Hu$ | $p_{11}\rightarrow D_6$ | $(D_4)\overset{\pm}{\rightarrow}A$ | $(A)\overset{\pm}{\rightarrow}A$ | $(D_5)\overset{\pm}{\rightarrow}K$ | $\langle(0^{\mathrm{iv}})\rightarrow A\rangle$ |
| 2 | $c(A)\overset{\pm}{\rightarrow}K$ | $(C)\overset{\pm}{\rightarrow}K$ | $(Hu)\rightarrow D_5$ | $p_{11}\overset{\pm}{\rightarrow}D_5$ | $(D_3)\overset{\pm}{\rightarrow}K$ | $(A)\overset{\pm}{\rightarrow}A$ | $[(0'')\rightarrow B]$ | $\langle(0^{\mathrm{iv}})\rightarrow B\rangle$ |
| 3 | $(273)\rightarrow Hl$ | $(-2')\overset{\pm}{\rightarrow}A$ | $(270)\dot{\rightarrow}A$ | $(D_5)\overset{\pm}{\rightarrow}K$ | $c(A)\rightarrow 0'$ | $(A)\overset{\pm}{\rightarrow}A$ | $(D_1)\overset{\pm}{\rightarrow}K$ | $300p_{11}+513p_1$ |
| 4 | $(Hu)\rightarrow D_5$ | $(C)\overset{\pm}{\rightarrow}K$ | $\frac{1}{2}(A)\rightarrow Hl$ | $[(0'')\rightarrow A]$ | $(D_6)\rightarrow Hu$ | $(D_2)\overset{\pm}{\rightarrow}A$ | $(B)\rightarrow 300$ | $313p_{11}+513p_1$ |
| 5 | $(Hu)\rightarrow A$ | $c(A)\rightarrow -2'$ | $(Hu)\rightarrow D_6$ | $(A)\rightarrow D_0$* | $(Hu)\rightarrow D_2$ | $c(A)\overset{\pm}{\rightarrow}K$ | $p_{11}\overset{\pm}{\rightarrow}D_1$ | $326p_{11}+517p_1$ |
| 6 | $\frac{1}{2}(A)\rightarrow A$ | $(D_5)\overset{\pm}{\rightarrow}D_5$ | $p_1.(A)\overset{c}{\rightarrow}S$ | $(263)\dot{\rightarrow}A$ | $p_{11}\overset{\pm}{\rightarrow}D_2$ | $(300)\rightarrow A$ | $237\rightarrow S$ | $0p_{11}+1p_1$ |
| 7 | $3,31\dot{\rightarrow}A$ | $s(D_5)\overset{c}{\rightarrow}S$ | $179\rightarrow S$ | $(267)\overset{=}{\rightarrow}A$ | $(D_1)\rightarrow A$ | $c(A)\rightarrow D_0$ | $(D_8)\rightarrow Hu$ | $227p_{11}+1p_1$ |
| 8 | $z(A)\overset{c}{\rightarrow}S$ | $2\rightarrow A$ | $(D_5)\overset{\pm}{\rightarrow}K$ | $z(A)\overset{c}{\rightarrow}S$ | $(D_2)\overset{=}{\rightarrow}A$ | $172\rightarrow S$ | $(Hu)\rightarrow D_1$ | $268p_{11}+1p_1$ |
| 9 | $82\rightarrow S$ | $0\rightarrow B$ | $[(0'')\rightarrow A]$ | $193\dot{\rightarrow}S$ | $s(A)\overset{c}{\rightarrow}S$ | $(D_8)\overset{\pm}{\rightarrow}K$ | $p_{20}\rightarrow D_8$ | $307p_{11}+1p_1$ |

* The primary and control routines in locations 0–29 have been omitted from this table to avoid duplicating part of Table 3, and the $D$ registers are used for much the same purposes, as in Table 3.

† The number 22 represents $t_0^I-t_0^H+2$, where the two spaces are for parameters involved in transfer of the function block.

‡ The programme parameters have been given the values $t_0^H=300$, $t_0^I=320$, $i^H=273$, $i^I=276$.

§ This and other sign tests on $(D_8)$ have no effect for $(D_8)=22$ during transfer of hyper-routines, but command 249 sets $p_{20}$ in $D_8$ to introduce other operations during transfer of interpretive routines.

¶ The number 3 represents $i^I-i^H$.

sequence of commands 247–262 designed to prepare for transfer of interpretive routines. Commands 247–251 replace $t_c^H$ by $t_c^I$, $t_c^I$ by $p_{20}$, and $t_e$ by $t_0^I - t_0^H$

252–258 change 174 to refer to $S_P^I$, and 129, 154, 242 to refer to $S_L^I$
259–261 sets the first command in $S_P^I$ as $50 \rightharpoonup S$
262 shifts to 107 with $D_3$ previously set at 42 by command 191
107–125 with $(A)$ zero set $c$, $a_L$, $n$ zero corresponding to a pseudo-command of the ETC type in 42′.

The interpretive system of routines is transferred from $1^{iv}$–$227^{iv}$ to $50'''$–$276'''$ and (42′) is adjusted as though it were an ETC-type compiler code. Commands 184–186 increase $t_e$ from 20 to 22, which is then used to organize the transfer of the arc cosine routine from $269^{iv}$–$306^{iv}$ to $277'''$–$314'''$. It will be noted that commands 148–151 set $h_0 = 50$ in location 323 so that during transfer all commands with $p = 1$ will be adjusted to refer to address $h_0 = 50$, the head address of the interpretive system of routines. When this transfer is complete, $t_e$ exceeds $t_c^I$ and control returns to the primary routine which reads $p_{20} \rightharpoonup T, D$ from tape and the computer stops.

By use of switchboard operations, the contents of the store $S_P^I \equiv M'''$ are transferred to $M$, all registers are then cleared, and the data tape is inserted in the tape reader. When the computer is started, command 0 shifts control to 50, the first command of the interpretive system of routines, and this system proceeds to extract and interpret hyper-commands. The hyper-commands 0′–4′ read in nine two-word coefficients into locations 8–25 and then, for each $x$-value read by hyper-command 5′, the value of $y$ is calculated by hyper-commands 6′–19′ and printed by hyper-command 20′.

This example illustrates those aspects of hyper-compiler techniques, which differ from the techniques previously described for computer-code compilers.

## VIII. Extension of Hyper-compiler Techniques

The volume of punching required in forming the programme tape may be reduced by minimizing the data put on the tape for library references, as indicated previously. Further reduction of punching of hyper-codes can be obtained by including an additional routine linked into the primary routine designed to expand 10 binary digit codes, consisting of five address digits, $a$, and five operation code digits, $f$, into the conventional 20-digit hyper-code form, $ap_{11} + fp_6 + 20p_1$.

A further economy in store space would be obtained if only those interpretive routines actually required by the programme were transferred to form the interpretive programme finally used. This would require an additional routine to examine each hyper-code of the input programme and transferred hyper-routines, and store a third set of transfer data. These transfer data would be used following transfer of hyper-routines to select standard interpretive routines for transfer, and, when this is completed, the ETC-type routines would be transferred in the manner already considered.

In certain cases, the library of routines used may require extension of the hyper-compiler routine to cover the case of parameters modifying the mode of

transfer as in unrolling a polynomial, and the case of ETC-type interpretive routines which require other ETC-type routines as subordinates. These extensions correspond to those discussed in the case of computer-code compiler routines, and were omitted in compiler $H2$ for the sake of simplicity.

## IX. Processes for Use with Compiler Routines

While most programmers will be concerned solely with construction of problem programmes, some will be concerned with construction of libraries for use with compiler routines.

### (a) *Process for Construction of Libraries*

Having selected the routines to be formed into a library, adjust the coding of parametric commands to include $p.p_{18}$ in the $p_{20}$–$p_{18}$ digits of commands of the form $(?)\rightarrow M$, $(M)\rightarrow ?$, $K\rightarrow S$, or $(8+p)p_{17}$ in the $p_{20}$–$p_{17}$ digits of hyper-commands. Allocate library code numbers to the routines and then replace all references to subordinate routines by $mp_{11}+ep_1$ in the case of computer-code compilation or by $m,e\rightarrow S$, $m,e\rightarrow \bar{L}$ in the case of hyper-code references to ETC routines or hyper-routines respectively. If parametric subordinate routines are involved, insert the parameters immediately following the compiler code in the superior routine. Finally insert $lp_{11}$ preceding the routine and in the case of computer-code programmes include library data $Lp_{16}+R_yp_{11}+R_zp_6+R_xp_1$ preceding the length datum.

These may be assembled together with the primary, control, and compiler routine into $M''$ using control designation $mS$ to store the $a_L$ of the $m$th library routine and the library index is simply formed for each $m$ by the control operation $mA$, $mp_2+p_1$ and the result inserted into the appropriate location $i^H+m$ of $M''$. The resultant sequence of commands can then be punched out on tape using a standard routine and is thereafter available for subsequent use.

It remains only to form a " library record " for use by other programmers intending to use this library. The library record lists the routine operations provided by the library together with a list of corresponding values of $m,e,n$ and a description of any parameters involved. Also included are the terminating control designations to be punched following input programmes as in Table 8.

### (b) *Processes for Users of Libraries*

The problem programme is designed in the conventional manner with the following exceptions. In the case of hyper-programmes data locations 0,1,2 are left free for use as registers. All references to library operations are encoded as $m,e,E$; $m,e\rightarrow \bar{L},E$; or $m,e\rightarrow S,E$ as the case may be, where $m$ and $e$ are obtained by reference to the library record. Should $n$ be non-zero the parameters indicated in the library record are provided immediately preceding the compiler code.

The terminating control operations specified in the library record are punched on tape immediately following the punching of the input programme. After the library is inserted from tape and checked, the input programme is

inserted as has been described. The compiled programme may then be transferred to appropriate stores by switchboard operations and performed in the standard fashion.

TABLE 8

HYPER-PROGRAMME COMPILATION FOR $y=(\sin x)/(\arccos x)-\sum_{r=0}^{8} a_r x^r$ USING INTERPRETIVE ROUTINES FOR DOUBLE PRECISION ARITHMETIC

| Programme Commands | | | | Library Contents | | |
|---|---|---|---|---|---|---|
| No. | As Punched | | As Compiled | As in Table 7 | $a_L$ | $l$ |
| | $1S,\ hT,\ 2S,\ 8nT,$ $3S,\ 1AT$ | | $[h=5,\ n=8]$ | Primary routine | 0″ | 16 |
| | | | | Control routine | 16″ | 14 |
| 0 | $3A$ $0\rightarrow D_0$ | | $8\rightarrow D_0$ | Input parameters | 30″ | 10 |
| 1 | $(D_0)\overset{+}{\rightarrow}\bar{K}$ | | $(D_0)\overset{+}{\rightarrow}K$ | Compiler $H2$ | 40″ | 233 |
| 2 | $2A$ $(\bar{I})\rightarrow 0$ | | $(\bar{I})\rightarrow 5$ | Index of hyper-routines | 273″ | 3 |
| 3 | $1\overline{\rightarrow}D_0$ | | $1\overline{\rightarrow}D_0$ | Index of ETC routines | 276″ | 4 |
| 4 | $1A$ $1\rightarrow\bar{S}$ | | $1\rightarrow\bar{S}$ | | | |
| 5 | $(\bar{I})\rightarrow 4$ | | $(\bar{I})\rightarrow 4$ | *C* Library of Hyper-routines | | |
| 6 | $(4)\rightarrow\bar{A}$ | | $(4)\rightarrow\bar{A}$ | | | |
| 7 | $2,0\rightarrow S.$ | $E$ | $277\rightarrow S$ | 0 Exponential | 300″ | 12 |
| 8 | $(\bar{A})\rightarrow 3$ | | $(\bar{A})\rightarrow 3$ | 1 Sinecosine | 313″ | 12 |
| 9 | $(4)\rightarrow\bar{A}$ | | $(4)\rightarrow\bar{A}$ | 2 Polynomial | 326″ | 8 |
| 10 | $(\bar{A})\rightarrow 0$ | | $(\bar{A})\rightarrow 0$ | | | |
| 11 | $1,0\rightarrow\bar{L}.$ | $E$ | $22\rightarrow\bar{L}$ | Library of Interpretive Routines | | |
| 12 | $(0)\rightarrow\bar{A}$ | | $(2)\rightarrow\bar{A}$ | | | |
| 13 | $(3)\overset{\div}{\rightarrow}\bar{A}$ | | $(3)\overset{\div}{\rightarrow}\bar{A}$ | 0 Basic interp. system | $0^{iv}$ | 226 |
| 14 | $(\bar{A})\rightarrow 3$ | | $(\bar{A})\rightarrow 3$ | 1 Cube root | $227^{iv}$ | 40 |
| 15 | $(4)\rightarrow\bar{A}$ | | $(4)\rightarrow\bar{A}$ | 2 Arc cos | $268^{iv}$ | 38 |
| 16 | $(\bar{A})\rightarrow 0$ | | $(\bar{A})\rightarrow 0$ | 3 Logarithm | $307^{iv}$ | 52 |
| | $3A$ 0 | | — | | | |
| | $2A$ 0 | | — | Transfer Data | | |
| 17 | $2,0\rightarrow\bar{L}.$ | $E$ | $34\rightarrow\bar{L}$ | Loc. Hyper-routines | Loc. | Interp. Routines |
| 18 | $(0)\rightarrow\bar{A}$ | | $(2)\rightarrow\bar{A}$ | 300 11 ; 22 | 320 | — ; 50 |
| 19 | $(3)\overline{\rightarrow}\bar{A}$ | | $(3)\overline{\rightarrow}\bar{A}$ | 301 17 ; 34 | 321 | — ; 50 |
| 20 | $(\bar{A})\rightarrow 0t$ | | $(\bar{A})\rightarrow 0t$ | 302 64;0 i.e. $8n$ | 322 | 7 ; 277 |
| 21 | $1A$ $5\rightarrow\bar{S}$ | | $5\rightarrow\bar{S}$ | 303 5;0 i.e. $h$ | 323 | — ; 50 |
| | $101\rightarrow S.$ | $D$ | — | | | |
| | $50T,247\rightarrow S.$ | $D$ | — | Hyper-programme in $M'$ | Interp. Programme in $M'''$ | |
| | $p_{20}\rightarrow T.$ | $D$ | — | 0′–21′ Input programme | 0‴– 49‴ Data space | |
| | | | | 22′–33′ Sinecosine | 50‴–276‴ Interp. syst. | |
| | | | | 34′–41′ Polynomial | 277‴–314‴ Arc cos | |

## X. USE OF COMPILER ROUTINES

Efficient use of a group of libraries requires a rapid means of checking that any library is correctly inserted from tape. An extra pseudo-code included

at the end of each library may be so chosen that the sum of this and preceding library command codes is zero. This enables a checking routine, using a repeated loop of commands to sum the command codes of any library, to check the accuracy of insertion in a few seconds.

Checking of compiled programmes is speeded by use of a special routine for printing the resultant programme in symbolic form. This routine prints the input programme in conventional form and then lists the name and store location of the head of all transferred routines. This process is quicker than printing the complete programme, including library routines, in the conventional manner. It is necessary, however, to design a group of codes associated with each library routine which is used by the programme-print routine to print the names of library routines transferred. These print codes can be stored in locations $a_L-1$ and $a_L-2$ for eight-letter mnemonics.

Design of compiler routines and associated libraries of routines requires little more programming time than is required for design of the library routines themselves. The time required for inserting and checking programmes using compiler techniques is less than that required previously, because of the considerable reduction in checking time due to the reduction of human errors in coding and punching.

A further advantage of compiler techniques is the simplification of programming techniques, which results in faster preparation of programmes and training of programmers. The " Programmers' Manual " for use with compiler techniques is much simpler than that required for previous techniques and reduces the need for specialized knowledge of instrumental techniques foreign to the field of research of the user.

The compiler techniques adopted for use with the C.S.I.R.O. Mark I computer have been designed so far to simplify the task of programming in conventional " command " form for both computer-code commands and hyper-code commands. Further simplification requires the input symbolism to be freed from the restrictive form of " command " notation and brought closer to the form of conventional algebra, to which mental idiom a far larger number of users is more accustomed.

## XI. References

Laning, J. H., and Zierler, N. (1954).—A program for translation of mathematical equations for Whirlwind 1. Engineering Memorandum E-364, M.I.T. Instrumentation Laboratory.

Office of Naval Research (1954).—Symposium on automatic programming for digital computers. (O.N.R.: Washington, D.C.)

Pearcey, T., and Hill, G. W. (1953*a*).—*Aust. J. Phys.* **6**: 316–34.

Pearcey, T., and Hill, G. W. (1953*b*).—*Aust. J. Phys.* **6**: 335–56.

Pearcey, T., and Hill, G. W. (1954).—*Aust. J. Phys.* **7**: 485–504.