PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

An Assessment of the System of Optimum Coding Used on the Pilot Automatic Computing Engine at the National Physical Laboratory

J. H. Wilkinson

Phil. Trans. R. Soc. Lond. A 1955 **248**, 253-281
doi: 10.1098/rsta.1955.0016

AN ASSESSMENT OF THE SYSTEM OF OPTIMUM CODING USED ON THE PILOT AUTOMATIC COMPUTING ENGINE AT THE NATIONAL PHYSICAL LABORATORY

By J. H. WILKINSON

Mathematics Division, National Physical Laboratory, Teddington, Middlesex

(*Communicated by Sir Edward Bullard, F.R.S.—Received* 10 *January* 1955)

CONTENTS

The paper describes the system of optimum coding which is used on the Pilot ACE, the electronic computer at the National Physical Laboratory. It includes a number of simple examples of programs prepared for the machine and gives an assessment of the gain in speed which results from the use of optimum coding in general. It concludes with a description of the design of the full-scale ACE which takes full advantage of the general principles embodied in the design of the Pilot ACE.

1. Introduction

During the years 1945–7 the design of a high-speed automatic digital computer, using a mercury delay-line storage system, was developed at the National Physical Laboratory, Teddington, by a small group of mathematicians working under the leadership of A. M. Turing. The machine included a number of special features which were intended to make it faster than other machines being designed at that time and using the same type of storage. The most important of these features were the provision of a small amount of short access storage in addition to the main store, and the simultaneous selection of the next instruction with the execution of the current instruction. To achieve the second of these objectives instructions were placed by the coder in such positions in the store that in general the next instruction emerged from it as the current instruction was completed. The machine used a variant of what is usually called a four-address code, instructions being in the form $A+B$ to C, with a fourth element D specifying the position of the next instruction. The four-address code was used not because of a preference for it from the point of view of coding but because, coupled with the other features of the machine, it made a higher speed possible than that attainable with a one-address code. The machine was to be called the Automatic Computing Engine (ACE), the term 'Engine' being used in recognition of the work of Charles Babbage, described by H. P. Babbage (1889).

In 1948, when the construction of a machine was started, it was decided that the ACE was too ambitious to be attempted at that stage and a smaller machine, the Pilot ACE, was designed, embodying some of the special features of the ACE. The Pilot ACE was regarded as a proving ground for the ideas involved in the ACE, and it was intended that the latter should be built after the Pilot ACE had been tried out. It was not envisaged that the Pilot ACE should be used for large-scale computation and for this reason economy of equipment was given overriding consideration. As a result many of the facilities provided on the machine are of a rudimentary nature, and sometimes several instructions are needed to achieve what would be done on the ACE (or many other full-scale computers) in one instruction. The extent of the economy in equipment can be gauged from the fact that the computer has only some 900 valves and 450 germanium diodes. The Pilot ACE was completed in May 1950, and early experience showed that it was a very fast and powerful computer. The only working machines in this country at that time were EDSAC at Cambridge University and the Manchester University machine. The rate of operation on both these machines is of the order of one instruction per millisecond. It is not possible to give a corresponding figure for the Pilot ACE since the rate at which instructions are obeyed varies from problem to problem, but the maximum rate is 16 instructions per millisecond, and there is a wide range of problems on which a rate of more than 5 instructions per millisecond is achieved. Accordingly, it was decided to bring the machine into regular operation, but in order to lessen the burden thrown on the user a small program of modifications was first undertaken.

Since these modifications were completed the machine has been used on a 13 h day basis on a large range of problems for government departments and industry. In the light of this experience, this paper attempts an assessment of the value of the special features included in the machine. It should be remembered, in comparing the speed of the Pilot ACE with that of other machines using a similar storage, that the Pilot ACE is seriously handicapped by the rudimentary nature of its facilities and that a considerable underestimate is obtained of what could be achieved on the ACE.

2. General considerations on access time

The mercury delay-line type of storage is the one which was most commonly used on the first group of electronic computers to be built. Although it is a most satisfactory store from the point of view of reliability and the pulse repetition rates which may be employed, it suffers from the weakness that in order to achieve a reasonable economy of equipment it is necessary to use delay lines which hold a moderate number of words, that is, numbers or instructions. This has the result that the average access time of any given word held in the store is disappointingly high having regard to the pulse repetition rate used. In fact the average access time is half the time taken for a sound wave to travel down one of the delay lines and is therefore independent of the pulse repetition rate except in so far as this effects what is regarded as the economical length of a delay line. The problem of access time is not peculiar to mercury delay lines, but is shared by other stores of a serial nature such as magnetostriction delay lines and magnetic drum storage. The Pilot ACE includes a number of special features in its design the effect of which is to overcome this

problem to some extent and the system of coding appropriate to such a design is now usually known as 'optimum coding'. The term is not particularly happily chosen since it would seem to imply that the time taken to perform a given program has been reduced to a true maximum though this is seldom, if ever, true.

There are two terms which are of importance in measuring the effective speed of delay-line machines and their definition is given here: *A minor cycle* is the period of time taken by a word in serial form to pass any point on a conducting line in the machine. Thus on the Pilot ACE, which use a word of 32 binary digits, and has a pulse repetition rate of 1 Mc/s, a minor cycle is 32 μs. *A major cycle* is the period of circulation of one of the main storage units. On the Pilot ACE these units each hold 32 words, so that a major cycle is 1024 μs or roughly 1 ms.

For purposes of comparison we shall refer to an 'orthodox' delay line machine which we shall take as a machine using a one-address code, in which consecutive instructions (often a half-word in length) occupy consecutive positions in the delay lines, except after special instructions such as discriminations which explicitly direct the machine to positions out of sequence. The EDSAC at the Mathematical Laboratory, Cambridge described by Wilkes (1948, 1949) is a well-known example of an orthodox machine in this sense. When computing, the sequence of events is as follows:

(i) An instruction is extracted from the store.

(ii) This instruction sets up gates appropriate to the transfer and operation which it involves.

(iii) The transfer and operation take place.

On an orthodox machine, by the time the operation is complete, the next instruction has passsed back into its delay line and is not available again until one major cycle after the first was extracted. For this reason alone, quite apart from the availability of the numbers which have to be transferred, instructions can take place at a maximum rate of only one per major cycle. The situation could be remedied by spacing instructions at a regular time interval, this interval being greater than the time taken to carry out any of the operations except for a few, such as multiplication, for which an extra period of an integral number of major cycles could be allowed. Alternatively, instructions could be spaced irregularly in such a way that the next emerged when the current one was completed. The second of these alternatives has been adopted on the Pilot ACE.

The problem of the availability of the numbers on which the arithmetic operations are to take place remains to be solved. It would not appear to be possible, in general, to place both instructions and numbers conveniently in the main store, but examination of programs shows that there are frequently extended sequences of operations in which only a few operands are involved. This is particularly true of basic subroutines. For example, the subroutine for the addition of two numbers $2^{\alpha}.a$ and $2^{\beta}.b$, expressed in floating form, to give an answer in the same form, contains a fair number of instructions, but has only the four operands a, α, b and β, and it produces during the course of the subroutine a number of simple arithmetic combinations of these operands. If a machine is equipped with a number of storage units which hold only one or two words each, then the access time of numbers in these stores will be one or two minor cycles, and these stores may be used to hold those numbers which are being operated upon most frequently at the current

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

stage of a computation. While doing floating addition, for example, they will hold the operands a, α, b and β at the beginning of the subroutine, and the intermediate values obtained, during the course of the subroutine. Although the bulk of all numbers stored will be in the main storage units, most of the instructions obeyed by the machine will concern numbers in short units.

A single example should suffice to show how this comes about. Suppose we wish to add two vectors of order n held in the main store, where the components of each vector are expressed as numbers in floating binary form. There will be a cycle of operations which will be performed n times, and in each repetition of that cycle, the only instructions which will be concerned with transferring numbers to or from the main store will be: one for extracting the component of one vector, one for extracting the component of the other vector and one for storing the component of the vector sum. All the instructions in the floating routine will perform operations on numbers in short units.

The situation is not unlike that which is common on machines using a magnetic drum auxiliary store. The majority of all subroutines and numbers are at any time held on the drum, but since each subroutine which is read into the high-speed store is used there, for a period which is considerably in excess of the time taken to read it from the drum (and the same applies to groups of numbers) the speed of the machine approximates to that of the high-speed store rather than that of the drum. The extent to which this is true will vary from one problem to another. Since the Pilot ACE is equipped with a magnetic drum there are three levels at which information may be stored, on the drum, in a long delay line or in a short delay line and the object of optimum coding is to make the effective speed of the machine as near as possible to that corresponding to the access time of the short delay lines.

3. Preliminary stage of coding on the Pilot ACE

It is convenient to consider the problem of coding on the Pilot ACE in two stages, usually referred to as the preliminary coding and the detailed coding respectively. The first stage has some points in common with the production of a flow diagram, though it is much more detailed than this operation as usually understood. In the preliminary coding no attention is paid to the position the instructions will occupy in the store. This is decided in the detailed coding, together with the precise form of the instruction words themselves. The code of the Pilot ACE is sometimes described as a 'three-address' type, but this form of classification is not particularly suitable for the machine. Each instruction calls for the transfer of words from one of 32 positions known as 'sources' to one of 32 positions known as 'destinations', and at the same time specifies which of 8 long storage units will provide the next instruction. The last address is necessary because instructions are not placed in a regular fashion in the store so that the position of the next instruction is not determined by a preassigned rule. On most machines instructions call for the transfer of single words; on some machines the transfer of half-words can also be effected. On the Pilot ACE transfers are of a much more flexible nature. The instruction word contains further information which specifies when a transfer starts and when it finishes, and a transfer lasting for any number of minor cycles from 1 up to 32 may be specified by an instruction in this way. This is done not only to achieve the transfer of multiple-length words in one instruc-

tion but also because, as will be seen below, it provides a number of useful arithmetic facilities such as left and right shifts at a very small cost in terms of equipment. It should be stressed that this feature of the code is not primarily occasioned by the fact that the machine employs optimum coding.

Simplest among the sources and destinations are those associated with the short storage units holding one word each, which are usually referred to as 'temporary stores' or *TS*'s because they are used to store temporarily those numbers which are in most frequent use at the current stage of a computation. These *TS*'s are five in number and, as a result of modifications which took place after the machine was first completed, they are not consecutively numbered. They are in fact, *TS*15, *TS*16, *TS*20, *TS*26 and *TS*27. Each *TS* has one source and one destination, the source belonging to *TSn* being source *n*, and the destination, destination *n*. When source *n* is selected a copy of the contents of *TSn* pass into a common line, 'highway', which connects all sources and destinations (see figure 1). When destination *n* is selected the number on highway passes into *TSn* and replaces its contents. In the preliminary coding a transfer from source 16 to destination 20 for example, is denoted by

16–20.

After this instruction has taken place both *TS*16 and *TS*20 contain the number originally in *TS*16. The period of transfer is not specified, because a transfer for more than one minor cycle achieves nothing more than a transfer for one minor cycle. Where the period is not specified it is assumed that it is for one minor cycle only, and this is the most common situation.

With these five sources and destinations we can only transfer numbers from one *TS* to another. In order to make computation possible there are a number of functional sources and destinations which are chiefly associated with the *TS*'s. *TS*16 for example, has, in addition to source 16 and destination 16, two functional destinations, destinations 17 and 18. If a transfer of a number to destination 17 is made, that number is added to the number previously in *TS*16. Numbers transferred to destination 18 are subtracted from the contents of *TS*16. When using these destinations the period of duration of the transfer is of great significance. If a transfer from a *TS* (say *TS*15) to destination 17 is made repeatedly over a period of *n* successive minor cycle, *n* copies of the content of *TS*15 are added to the original content of *TS*16. In this way small multiples (up to 32) of the content of any *TS* may be added to *TS*16. In particular, a transfer for 10 minor cycles is common in binary to decimal and decimal to binary conversion. In the preliminary coding the instruction for such a transfer continued for *n* minor cycles is written

15–17 (*n*).

A similar facility is provided by a prolonged transfer to destination 18. The transfer

16–17 (*n*)

is of particular importance. This adds the content of *TS*16 to itself for *n* consecutive minor cycles and hence multiplies the content by 2^n, or as it is more commonly expressed gives a left shift of *n* places. It will be seen that *TS*16 has a number of properties similar to those normally associated with the accumulator on an orthodox machine and it is usually referred to as the small accumulator.

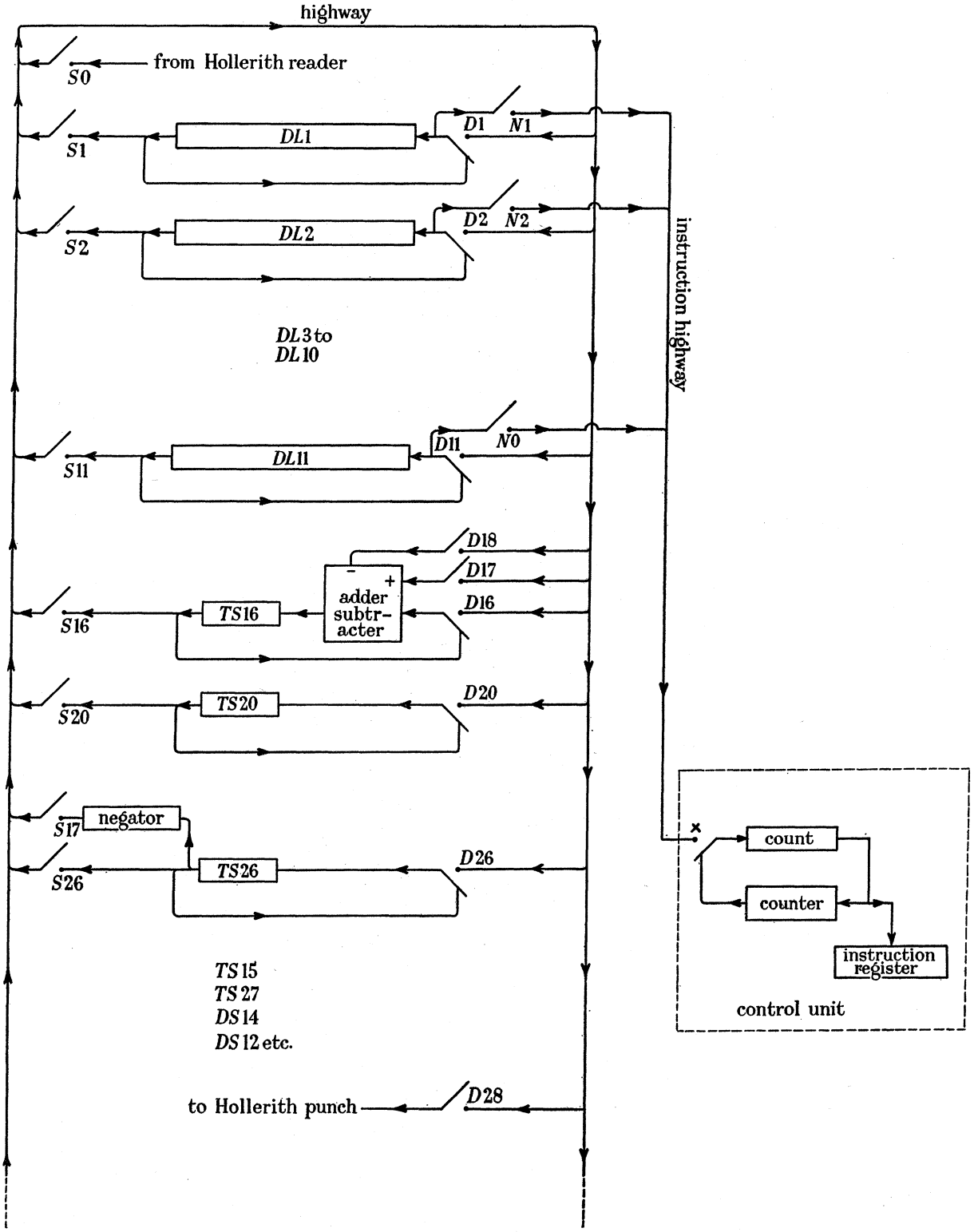


FIGURE 1. Simplified diagram showing some sources, destinations and next instruction sources.

*TS*26 and *TS*27 are associated with a number of functional sources. These are listed below.

Source 17 gives the 'ones complement' of the number in *TS*26 and is usually referred to as '$\sim TS26$'. This gives the number in *TS*26 with zeros and ones interchanged.

Source 18 gives the contents of *TS*26 divided by two, that is, with a right shift. The instruction

$$18-26\ (n)$$

therefore gives a right shift of n binary places. The number in *TS*26 is treated as a signed number, a negative number being represented by its complement with respect to 2^{32}.

Source 19 gives the contents of *TS*26 multiplied by two, that is, with a left shift. The instruction

$$19-26\ (n)$$

therefore gives a left shift of n binary places.

Source 21 gives the result of collating the number in *TS*26 with that in *TS*27, usually referred to as (*TS*26 & *TS*27). This derived number has a 1 in those digital positions in which the numbers in both *TS*26 and *TS*27 have a 1. Thus

if source 26 is	1	0	1	1	0	1	1
and source 27 is	1	1	0	1	0	1	0
then source 21 is	1	0	0	1	0	1	0

Source 22 gives a number which has a 1 in those digital positions in which the numbers in *TS*26 and *TS*27 disagree. This is usually referred to as ($TS26 \not\equiv TS27$). Thus

if source 26 is	1	0	1	1	0	1	1
and source 27 is	1	1	0	1	0	1	0
then source 22 is	0	1	1	0	0	0	1

If source 26 and source 27 have no 1's in common, then source 22 gives the sum of the two numbers, and the source is often used in this way when *TS*26 contains a single digit and *TS*27 contains a number which has not got a 1 in this position. This is a common situation when a number is being built up digit by digit. Another useful property of this source is that its most significant digit is a 1 only if the most significant digits of the numbers in *TS*26 and *TS*27 are different. Interpreting this result in the usual convention for positive and negative numbers, source 22 gives a negative number if the signs of the numbers in *TS*26 and *TS*27 are different and a positive number if the signs are the same.

There are a number of sources which provide useful constants. These are:

Source 23 which gives a single digit in the 17th position usually referred to as *P*17.

Source 24 which gives a single digit in the most significant position, referred to as *P*32.

Source 25 which gives a single digit in the least significant position, referred to as *P*1.

Source 28 which gives the number zero.

Source 29 which gives 32 consecutive 1's.

These five sources are not associated with delay lines, but are derived from gates on the basic pulse generators. Their great value lies in the fact that they provide numbers with

an access time of one minor cycle so that they supplement the *TS*'s to some extent while requiring very little equipment.

The above sources and destinations are adequate for the description of a trivial program which will serve to illustrate the preliminary coding stage. It has been mentioned earlier that each instruction selects the next. In the preliminary coding it is assumed that each instruction selects the one written immediately underneath, except at special points such as at the end of a repeated cycle where an indication is given of the next instruction. The way in which the next instruction is selected is dealt with in the detailed coding. The calculation to be programed is the derivation of the successive natural numbers, their squares and their cubes. The method of building up the squares and cubes is made as simple as possible since it is merely intended that the program should illustrate the process.

The successive natural numbers will be stored in *TS*15, the squares in *TS*20 and the cubes in *TS*26. The program consists of two parts, an initial part which puts zero into *TS*'s 15, 20 and 26 for the initial values of n, n^2 and n^3, and then a cycle of instructions which is such that it produces $(n+1)$, $(n+1)^2$, $(n+1)^3$ starting from, n, n^2, n^3:

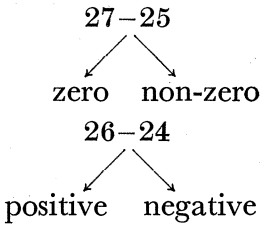
	28–15	zero to *TS*15
	28–20	zero to *TS*20
	28–26	zero to *TS*26
(*A*)	26–16	*TS*16 contains n^3
	20–17 (3)	*TS*16 contains n^3+3n^2
	15–17 (3)	*TS*16 contains n^3+3n^2+3n
	25–17	*TS*16 contains n^3+3n^2+3n+1
	16–26	*TS*26 contains $(n+1)^3$
	20–16	*TS*16 contains n^2
	15–17 (2)	*TS*16 contains n^2+2n
	25–17	*TS*16 contains n^2+2n+1
	16–20	*TS*20 contains $(n+1)^2$
	15–16	*TS*16 contains n
	25–17	*TS*16 contains $(n+1)$
	16–15	*TS*15 contains $(n+1)$
	(*A*)	

The last instruction is followed by (*A*) which indicates that the next instruction is that indicated by (*A*) on the left. All the instructions in this table are concerned with *TS*'s so that we would expect optimum coding to be very effective. The prolonged transfer is used in one or two of the instructions to add small multiples of numbers in *TS*'s to the contents of the small accumulator.

3·1. *Discriminating facilities*

Branching in programs is usually effected by the use of two special destinations. If a transfer is made from any source to destination 25 then the next instruction is one of two, according as the number transferred is zero or non-zero. If a transfer is made to destination 24 then the next instruction is one of two, according as the number transferred is

positive or negative. If a prolonged transfer is made to destination 24 then the two alternatives are selected according as all the numbers transferred are positive or at least one is negative. Branching is indicated in the preliminary coding as shown below.



(Discrimination of the zero, non-zero nature of the number in *TS*27)

(Discrimination on the sign of the number in *TS*26)

Suppose we wish to count the natural numbers in *TS*15 until we reach 2^{15}; then the program is

(*A*)	15–16	*n* in *TS*16
	25–17	(*n*+1) in *TS*16
	16–15	(*n*+1) in *TS*15
	16–17 (16)	Shift contents of *TS*16, 16 places to left
	16–24	

positive (*A*) negative (next program)

3·2. *Two-word stores and associated facilities*

There are two two-word stores in the machine which are referred to as *DS*'s or double-length temporary stores. These are *DS*12 and *DS*14. The two words in each of these delay lines are referred to as the even word and the odd word. The odd word, for example, energes from a *DS* during odd minor cycles and must be inserted during an odd minor cycle. In the preliminary coding we must specify which word is being transferred. The notation

$$14_o\text{–}16$$

represents the transfer of the odd word from *DS*14 to *TS*16, while

$$16\text{–}12_e$$

represents a transfer from *TS*16 to the even word position in 12, and

$$14\text{–}17\ (2)$$

represents the addition of both words in *DS*14 to the word in *TS*16. It is not possible to transfer the odd word from *DS*14 to the even position in *DS*12 without using a *TS* as an intermediate position. Two instructions of the types

$$14_o\text{–}TS$$
$$TS\text{–}12_e$$

are both necessary to effect such a transfer. The *DS*'s may be used to store two separate words or one double-length word. *DS*14 is associated with a number of functional sources and destinations. Normally these arithmetic functions treat the two words in *DS*14 as one number of 64 binary digits, the even word being the least significant 32 digits and the

odd word the most significant 32 digits. As will be seen later, the functions can also be modified by a single instruction so that they treat the numbers in *DS*14 as two separate single-length numbers.

Destination 13. When a number is transferred to destination 13 that number is added to the contents of *DS*14. The instruction

$$12-13\ (2)$$

adds the double-length number in *DS*12 to the double-length number in *DS*14. It does not matter if the first minor cycle of the transfer is even or odd. Similarly,

$$12-13\ (2n)$$

adds n times the double-length number in *DS*12 to *DS*14.

The transfer

$$14-13\ (2n)$$

is of special significance and is similar in effect to the instruction,

$$16-17\ (n).$$

The content of *DS*14 is added to itself repeatedly and therefore doubled after every two minor cycles of transfer. The instruction may therefore be used to give a left shift of up to 16 places of the double-length number in *DS*14. The transfer must, however, begin in an even minor cycle. The instruction

$$TS-13_o$$

has the effect of adding the number in the *TS* to the odd half of *DS*14. If there is a carry from the 32nd position it is lost and does not interfere with the even half of *DS*14. The odd half of 14 therefore behaves just like a single length accumulator in these circumstances. The instruction

$$TS-13_e,$$

on the other hand, adds the number in the *TS* to the even half of *DS*14, but a carry from the 32nd position adds into the first position of the odd half of 14.

Source 13 gives the contents of *DS*14 divided by 2, that is with a right shift. The instruction

$$13-14\ (2n)$$

may therefore be used to produce a right shift of up to 16 places, but as with the instruction

$$14-13\ (2n)$$

the transfer must begin in an even minor cycle.

3·3. *The long storage units*

The bulk of the high-speed storage is provided by 11 long storage units each of which holds 32 words of 32 binary digits. These units are usually referred to as 'delay lines' or *DL*'s and they are numbered *DL*1 to *DL*11. When extracting words from *TS*'s the minor cycle in which the transfer takes place is irrelevant since the same word is available in any minor cycle. For *DS*'s we need only specify whether the word is extracted in an even or an odd minor cycle. Each of the 32 words in a *DL* is identified by the time at which that word is available. We may regard successive minor cycles as numbered 0 to 31 with

minor cycle 0 following minor cycle 31 again. The word which emerges from delay line A in minor cycle number n is usually denoted by $[A_n]$ and the storage position itself by A_n. In orthodox delay-line machines it is not usual to pay attention to the particular delay line in which a number is stored but on the Pilot ACE the differentiation of each address into a spatial and temporal part is almost always observed. The prolonged transfer is of considerable importance in relation to the delay lines. For example by the instruction

$$9-10\ (32)$$

the whole of the contents of $DL9$ may be transferred to $DL10$. Similarly, by the instruction

$$10-17\ (32)$$

all 32 words in $DL10$ may be added to the original content of $TS16$. These two instructions are of great value in the use of the auxiliary magnetic store with which the machine is equipped. In the preliminary coding, transfers of specific words to or from delay lines are denoted by instructions of the type

$$2_7-16$$

which represents the transfer to $TS16$ of the word which is available from $DL2$ in the 7th minor cycle.

3·4. *Multiplication*

The multiplier on the Pilot ACE is of a very rudimentary nature and uses very little equipment. It is only partly automatic in that it produces the product of two numbers which it regards as positive integers. If the numbers stored are regarded as integers then a negative number 'a' is stored as $2^{32}+a$ on the Pilot ACE. In order to obtain the product of two numbers in this sign convention a small subroutine of instructions is used. The multiplier is associated with $DS14$ and $TS20$, and to multiply two numbers 'a' and 'b' together, 'a' must be sent to $TS20$, 'b' to $DS14$ odd and $DS14$ even must be cleared. If a transfer beginning in an odd minor cycle is made to destination 19 from any source, multiplication is initiated and is completed 65 minor cycles later, that is, just over 2 ms later. The product is given as a 64 digit number in $DS14$; 'b' is destroyed but 'a' remains in $TS20$.

The numbers 'a' and 'b' are treated as positive integers satisfying

$$0\leqslant a<2^{32} \quad 0\leqslant b<2^{32},$$

and to get the correct answer corresponding to an 'a' and a 'b' expressed in the sign convention $2^{32}(2^{32}-a)$ must be added to the result if b is negative and $2^{32}(2^{32}-b)$ must be added if a is negative, both corrections being necessary if both are negative. The multiplier works in parallel with the rest of the machine, and other instructions may be carried out while the product is being formed. If, however, destinations 13 or 14 are used in this period the product will be incorrect and there is no safety precaution against this. An interlock could have been provided which would have suspended operation for an integral number of major cycles until multiplication was complete if destinations 13 or 14 were used, but this was not done in the interests of economy. Because of the rather tiresome timing considerations, multiplication is usually performed by means of a subroutine in which these points are dealt with once and for all. The fact that the multiplier works in

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

parallel with the rest of the machine means that the corrections to be applied to the answer may be built up in *TS*16 (the small accumulator) during multiplication, so that no extra time is needed. Although it requires several instructions (12 in the simplest multiplication routine) the multiplication of two signed numbers still takes only a little over 2 ms.

3·5. *Alarm signal*

It is convenient to have some audible means of attracting the attention of the operator during the course of a computation. This is provided by a buzzer on the Pilot ACE which is controlled by destination 29. If a non-zero number is transferred to destination 29 then the buzzer sounds and continues until it is switched off. If a zero number is transferred then the buzzer does not sound. The use which is made of this facility is of course in the hands of the programer.

3·6. *Magnetic drum store*

Two years after it first came into regular use the Pilot ACE was equipped with a magnetic drum auxiliary store. Again the main considerations observed in its design were economy of equipment and a method of integration with the existing machine which required the minimum of modifications, so that it could remain continuously in use. It was also hoped to make the extra storage capacity provided as nearly as possible the equivalent of extra delay lines as regards its effective access time.

The drum has 128 tracks each of which stores 32 words, giving an auxiliary storage capacity equivalent to 128 additional long delay lines. Although there are 128 tracks there are only 16 reading and 16 writing heads. The tracks are divided into 8 groups of 16, and the reading and writing heads have 8 positions in each of which they have access to one group of 16 tracks. The reading and writing heads may be moved independently.

Information is read from the drum to the high-speed store or written from the high-speed store on the drum, one track at a time. The transfer always takes place to and from *DL*9.

To call for a transfer either way two instructions are needed, one to shift the reading or writing heads to the required group of 16 tracks and one to select a track within the group. If it is known that the heads are in the correct position then the first instruction may be omitted.

In order to describe the magnetic instructions it is necessary to introduce an element of the instruction word which does not otherwise concern us in the preliminary coding. This is a two-digit element taking the values 0, 1, 2 and 3 which is known as the characteristic. Normally the characteristic plays a role in determining the period of transfer, but in the magnetic instructions a characteristic of zero is used in instructions reading from the drum and a characteristic of one for writing on the drum. In the preliminary coding of the magnetic instructions a characteristic of zero is not mentioned, but a characteristic of one is denoted by an '*s*' after the instruction. The reason for this will be seen later.

The instructions for reading are

$(16+m)-22$ Select mth group of tracks where m takes the values 0 to 7;

followed by

$n-22$ Read the nth track in a group where n takes the values 0 to 15.

Similarly, the instructions for writing are

$(16+m)-22\ (s)$ Select mth group of tracks;

followed by $n-22\ (s)$ Write on the nth track in the selected group.

When an instruction refers to destination 22 then the source number and the first digit of the characteristic are set upon a register which determines the nature of the magnetic transfer.

The transfer of information is set in train by the magnetic instruction and approximately 10 ms are taken to transfer one track to or from $DL9$. The rest of the machine may be used normally while this transfer takes place, but if we reach an instruction which uses destination 22, source or destination 9, that instruction is not obeyed until the magnetic transfer is complete. There is a similar interlocking facility which waits for the completion of a head shift when necessary.

The time taken for the heads to shift is approximately 60 ms. If the instructions

$16-22\ (s)$
$5-22\ (s)$

are used for example and the writing heads are not already in the first position, then the second of these two instructions will not be obeyed until the head shift is completed. If, however, the heads are already in the first position then the second instruction will be obeyed immediately after the first. When a track of information is transferred from the drum to $DL9$ then, provided its contents are used for a period which is considerably in excess of 10 ms, the access time of data on the drum is unimportant. This is usually the case, but where it is not, then if speed is important this is achieved by making the transfers one move ahead of the time when they are required. One track of information is used while the next is being transferred. The fact that the prolonged transfer enables us to send the contents of $DL9$ to any other DL in one instruction is of great importance, because as we shall see later this takes only 1 ms. The prolonged transfer is also of great value in checking a magnetic transfer. If a track is used to store 31 words and minus their sum as the 32nd word, then the transfer may be checked by clearing $TS16$ and obeying the instruction

$9-17\ (32)$

which adds the 32 words in $DL9$ together and produces the sum in $TS16$. This sum should be zero.

A complete list of the sources and destinations is given in table 3 at the end of the paper.

4. Detailed coding

The second stage of coding a problem is usually referred to as the detailed coding. In order to understand this we must introduce the complete instruction word. Each instruction is allotted one complete word of 32 binary digits of which it uses 26 only, the remaining digits being spare. The elements of the instruction word in order of their position in the instruction are:

N (3 digits, 2 to 4). Selects from which of 8 permissible delay lines the next instruction is to be found.

S (5 digits, 5 to 9). Selects which of the 32 sources is to be used.

D (5 digits, 10 to 14). Selects which of the 32 destinations is to be used.

C (2 digits, 15 and 16). In general determines, in conjunction with the next two elements, the period of transfer. It is also used in a special manner in connexion with several destinations such as the magnetic destination 22.

W (5 digits, 17 to 21).
T (5 digits, 25 to 29). } Determine the period and timing of the transfer and the position of the next instruction in the DL selected by N.

G (1 digit, 32). This is a zero only if the instruction is a 'stop' instruction.

The element C is usually called the 'characteristic' and may take the values 0, 1, 2 or 3. Of the above elements S and D and the period of the transfer, which is a function of C, W and T, are mentioned in the preliminary coding. The meaning of the other elements may be defined as follows. If the instruction in question occupies position m in its delay line then the transfer is timed relative to this position. The instruction emerges from its delay line in minor cycle m (having been selected by the previous instruction). It is then registered during minor cycle $(m+1)$ and sets up the appropriate gates. The transfer begins in minor cycle $(m+W+2)$ and the next instruction is $[N_{m+T+2}]$ for all values of the characteristic. The latter determines when the transfer stops.

If the characteristic is 0 then the transfer lasts from minor cycle $(m+W+2)$ to minor cycle $(m+T+2)$, that is for $(T-W+1)$ minor cycles.

If the characteristic is 1, then the transfer lasts for one minor cycle only, namely minor cycle $(m+W+2)$ and the instruction is usually referred to as a 'single' transfer or an 's' transfer. If the characteristic is 3 then the transfer lasts for two minor cycles $(m+W+2)$ and $(m+W+3)$, and the instruction is usually referred to as a double transfer or a 'd' transfer. The value 2 of the characteristic is not used. When the characteristic is 1 the instruction may be expressed as follows. Transfer $[S_{m+W+2}]$ to D_{m+W+2} and take the next instruction from N_{m+T+2}. When the source or destination is a TS or a similar short access source then the subscript is irrelevant. For a characteristic of 3, two consecutive words beginning with $[S_{m+W+2}]$ are transferred and for a characteristic of 0 the transfer continues right up to the time when the next instruction emerges. In the above definitions all numbers are to be interpreted modulo 32. If $W=4$ and $T=2$ then for zero characteristic the transfer lasts for

$$(2-4+1)=-1 \text{ which is interpreted as } 32-1, \text{ i.e. } 31 \text{ minor cycles.}$$

Similarly, if the instruction in position 17 is

$$5.\quad 5-7\ s\ \ 15\ \ 16$$

then the next instruction is in $DL5$ position $(17+16+2)$, that is in $DL5_3$. The prolonged transfer may be regarded as an additional facility which happens to cost very little in terms of equipment. Its weakness is that the period of transfer terminates in the minor cycle position in which the next instruction is stored. For example if we wanted to transfer $[5_{6\text{ to }9}]$ to position $6_{6\text{ to }9}$ by an instruction occupying minor cycle 3 then the instruction needed would be $\quad N\quad 5-6\quad 1\ 4$

because this transfer starts in minor cycle 6 $(=3+1+2)$ and ends in minor cycle 9 $(=3+4+2)$. The next instruction is $[N_9]$. In this respect the prolonged transfer differs from the single and double transfers. For these the word or double word transferred may be in an arbitrary position and the next instruction also in an arbitrary position. The double transfer and the DS's are of great value when dealing with two-word aggregates

such as arise in complex, floating or double-length arithmetic. The decision to employ relative timing on the Pilot ACE was made in the interests of economy of equipment. In any future version of a delay line serial machine using optimum coding, it would almost certainly be replaced by an absolute timing system. If the numbers W and T were absolute timing numbers the instruction could be written

$$S_W - D_W \text{ next instruction } N_T,$$

and if this is interpreted in terms of addresses which are used on an orthodox machine, the instruction corresponds to $(32S+W)$ to $(32D+W)$ with the next instruction in $(32N+T)$, and it will be seen that the first two addresses are not arbitrary, but must have the same value modulo 32. Usually, of course, at least one of S and D is a short access store and then this interpretation is irrelevant for that address.

The element N may take the values 0 to 7. Corresponding to values 1 to 7 the next source of instruction is the corresponding $DL1$ to 7. For the value 0 however, the next source of instruction is $DL11$. This peculiarity is due to modifications which took place during the development of the machine.

A simplified schematic diagram of the Pilot ACE is shown in figure 1. The machine has two main lines of communication, one marked 'highway' which connects each of the 32 sources to each of the 32 destinations via gates, and a second marked 'instruction highway', which connects the 8 delay lines 1 to 7 and 11 via further gates to the main control unit. This control unit consists of a static register on which the N, S and D elements of an instruction are set up, and a counter which determines from the W, T and C elements the timing and duration of the transfer. The entry to this unit is controlled by a gate X and the time at which this is opened to let the next instruction in is also determined by the counter from the elements of the previous instruction. We may start at the stage when the gate X has been open for one minor cycle m, and has allowed an instruction to pass into the control unit. In minor cycle $(m+1)$ this instruction will be registered. The appropriate S and N gate will immediately become operative, but the selected D gate will not change over until a time determined by the counter. Instructions from the selected next instruction source pass down the instruction highway but do not enter the control unit because gate X is closed. The counter first determines the time at which the D gate will switch over and as soon as this occurs the transfer begins. It then determines when the D gate will switch back and so terminate the transfer and finally it determines when the X gate will open and allow the next instruction to pass into the control unit. It will be seen that the execution of the current instruction and the selection of the next instruction are carried out simultaneously.

The highest rate of operation is achieved when $W=T=0$. This gives a transfer lasting for one minor cycle (except for a 'd' instruction) and the next instruction is in position $m+2$. In subroutines, instructions of this type are very common and when a series of them is being obeyed the sequence of events is:

First minor cycle. Instruction 1 is registered.

Second minor cycle. Instruction 1 is obeyed and instruction 2 is extracted.

Third minor cycle. Instruction 2 is registered and so on.

The rate of operation is then 16 instructions per major cycle.

Transfers to the discriminating destinations are exceptions to the above rules. If the numbers transferred to destination 24 have no $P32$ digits, that is, are all positive, then the next instruction is determined in the usual way, otherwise it is one minor cycle later. Similarly, if the numbers transferred to destination 25 are all zero then the next instruction is in the normal position, otherwise it is one position later. This means that the two alternative instructions which follow a discrimination are always in consecutive positions.

A simple example of detailed coding, the formation in succession of the natural numbers from 1 to p and their squares, will illustrate the procedure. The detailed coding is given side by side with the preliminary coding (table 1). The numbers in brackets at the end of the instructions indicate the minor cycle position of the next instruction. All the instructions are stored in $DL1$ so that the N element is 1 in all of them. Programs are drawn up on special sheets which are divided into 3 columns and 32 lines numbered 0 to 31.

TABLE 1. CODING OF THE EVALUATION OF THE NATURAL NUMBERS AND THEIR SQUARES

detailed coding

minor cycle of instruction	N	S D	C	W	T	minor cycle of next instruction
0	1	28–15	.	0	0	(2)
1						
2	1	28–20	.	0	0	(4)
3			next program			
4	1	15–16	.	0	0	(6)
5						
6	1	20–17	d	0	1	(9)
7						
8						
9	1	25–17	s	0	0	(11)
10						
11	1	16–15	.	0	0	(13)
12						
13	1	20–16	.	0	0	(15)
14						
15	1	25–17	s	0	0	(17)
16						
17	1	16–20	.	0	0	(19)
18						
19	1	1_{21}–18	s	0	1	(22)
20						
21			contains p			
22	1	16–25	.	0	11	(3, 4)

preliminary coding

	1_0	28–15	(15 contains 0^2)
	1_2	28–20	(20 contains 0)
	.....		
$(A+1)$	1_4	15–16	(16 contains n^2)
	1_6	20–17 (2)	(16 contains n^2+2n)
	1_9	25–17	(16 contains n^2+2n+1)
	1_{11}	16–15	(15 contains n^2+2n+1)
	1_{13}	20–16	(16 contains n)
	1_{15}	25–17	(16 contains $n+1$)
	1_{17}	16–20	(20 contains $n+1$)
	1_{19}	p–18	(16 contains $n+1-p$)
	1_{21}	(p)	
	1_{22}	16–25	discriminate

non-zero 1_4 $(A+1)$ zero 1_3 (A) next program

Each column holds the content of one delay line. It will be seen that many of the instructions have $W=0$ and $T=0$ so that the arithmetic involved in the detailed coding is negligible. It should be appreciated that the W and T numbers are far from unique. For instance in the first instruction they could take any values whatever, and zero is chosen because this gives the highest rate of operation. In instructions like 1_9 where we have a transfer which *must* last for one minor cycle only, an 's' type of instruction is used although, since the W and T numbers are zero, a characteristic of zero would have been satisfactory. This is done as a safety measure. When modifications are made to a program it is frequently convenient to change the T number of an instruction. If an s type instruction is used then, even if we forget to change W, we still have a single transfer. The number

p may be said to be a parameter of the subroutine, and it is stored in such a position that it is immediately available when it is required. It is usually possible to store a few isolated numbers of this type so as to achieve a low access time, but when we are storing a whole array of numbers, such as a matrix or the coefficients of a power series, it is not practicable. The instruction $[1_{22}]$ is of special interest. It is a discriminating instruction and its timing number must be chosen so that if the number in $TS16$ is not zero, the next instruction is the first instruction of the repeated cycle. The alternative instruction which will be obeyed after forming p squares, is one position earlier than the beginning of the cycle. In the preliminary coding it is convenient to denote alternative instructions by (A) and $(A+1)$ so that when the detailed coding of the first of the pair is done one will be reminded that two consecutive positions must be available. In this example $(A+1)$ is coded first and placed in 1_4 and at this stage 1_3 should be marked with a cross to avoid using it before reaching (A). The main cycle of the program takes 1 major cycle only, and in fact the T number of the last instruction has been specifically chosen to return to the beginning of the cycle. The 9 instructions of the cycle are therefore obeyed in 1 major cycle instead of the 9 major cycles which would be needed on an orthodox machine. Here, as always in this paper, we mean by an orthodox machine an orthodox delay line machine. If the cycle had contained more instructions then the gain in speed could have been even more marked, though if there had been too many, two major cycles would have been taken. If the instructions in the main cycle could be done in less than 16 minor cycles, then by repeating them 2 complete cycles could be done in one major cycle. Similarly, if the cycle takes less than 11 minor cycles 3 cycles could be performed in a major cycle by having 3 copies of the instructions. This point is relevant in fundamental subroutines such as division or square root extraction where speed is very important.

5. Special facilities associated with the characteristic

There are two facilities of a rather special nature on the Pilot ACE which are more easily explained now the detailed coding has been described. They are associated with two trigger circuits known as TCA and TCB. Each of these triggers has two states, an 'off' and an 'on' state. The normal state of these triggers may be regarded as off, and the facilities so far described correspond to this state.

The trigger circuit TCA may be put in the 'on' state by a transfer from any source to destination 21 using an 's' type instruction. It then remains on until a transfer is made to destination 21 with characteristic zero. During the period when TCA is on, $TS20$ ceases to have an independent existence and is fed instead from $DL10$. Source 20 then supplies the contents of $DL10$ delayed by one minor cycle so that, for example, $[10_7]$ is obtained from $TS20$ in minor cycle 8. The instruction

$$20-10\ (32)$$

has the effect of delaying the whole contents of $DL10$ by one minor cycle.

The following example will illustrate the value of this facility. Suppose

$$\begin{aligned} 10_{0\text{ to }2} &\quad \text{contains} \quad x_n x_{n+1} x_{n+2} \\ 10_{6\text{ to }8} &\quad \text{contains} \quad y_n y_{n+1} y_{n+2} \\ 10_{12\text{ to }14} &\quad \text{contains} \quad z_n z_{n+1} z_{n+2} \end{aligned}$$

and we have a set of instructions which computes x_{n+3} and stores it in 10_3, y_{n+3} and stores it in 10_9 and z_{n+3} and stores it in 10_{15}. If the instruction

$$20-10\ (32)$$

is obeyed, then on its completion

$$\begin{aligned} 10_{31\text{ to }2} &\quad \text{contains} \quad x_n x_{n+1} z_{n+2} x_{n+3} \\ 10_{5\text{ to }8} &\quad \text{contains} \quad y_n y_{n+1} y_{n+2} y_{n+3} \\ 10_{11\text{ to }14} &\quad \text{contains} \quad z_n z_{n+1} z_{n+2} z_{n+3} \end{aligned}$$

It will be seen that $10_{0\text{ to }2}$ now contains

$$x_{n+1} x_{n+2} x_{n+3} \quad \text{instead of} \quad x_n x_{n+1} x_{n+2}$$

and, similarly, for $10_{6\text{ to }8}$ and $10_{12\text{ to }14}$. If x_{n+4}, y_{n+4}, z_{n+4} are now determined by the same rules as their predecessors, then the instructions used in their determination can refer to numbers in fixed positions. There are numerous occasions when this facility is of great value, the most common being in some methods of solving ordinary differential equations by step-by-step processes. The single instruction affects the multiple substitution

$$\begin{aligned} &x_{n+1} \text{ replaces } x_n \quad x_{n+2} \text{ replaces } x_{n+1} \quad x_{n+3} \text{ replaces } x_{n+2} \\ &y_{n+1} \text{ replaces } y_n \quad y_{n+2} \text{ replaces } y_{n+1} \quad y_{n+3} \text{ replaces } y_{n+2} \\ &z_{n+1} \text{ replaces } z_n \quad z_{n+2} \text{ replaces } z_{n+1} \quad z_{n+3} \text{ replaces } z_{n+2} \end{aligned}$$

in one major cycle. If there had been further variables, these too would have been operated upon at the same time.

The trigger circuit *TCB* may be put 'on' by a transfer from any source to destination 23 using an '*s*' type instruction. When *TCB* is on then the arithmetic facilities associated with *DS* 14 become those which are appropriate if the two words stored in it are entirely separate. Normally the arithmetic facilities, source 13 and destination 13, treat the two words as one double-length number with the most significant part in the odd position. *TCB* may be put off again by a transfer from any source to destination 23 with zero characteristic. *TCB* must be off during multiplication since in this process a double length product is being produced in *DS* 14.

The convention is adopted that both triggers are normally off. If a subroutine puts either trigger on then it must put it off again before returning to the main routine.

6. Input and output

The main input and output facilities on the Pilot ACE are provided by a Hollerith reader and a Hollerith punch; but there are simpler alternative means of input and output and these will be considered first.

The simple input is provided by a set of 32 manual switches on which a binary number may be set up. The number on the switches is available from source 0 once per minor cycle. The simple output is of a similar nature and consists of a set of 32 lights. Destination 28 is associated with this set of lights, and when a transfer is made from any source to destination 28 the number transferred is set up in binary form on the lights. The lights can only be cleared, that is the number registered on them returned to zero, by operating a manual switch.

Before describing the punched card input and output it is necessary to introduce another facility provided on the machine. So far it has been assumed that all instructions are carried out at high speed. In fact there is a digital position in the instruction word which is such that when it is a one, the instruction is carried out immediately, but when it is a zero the machine halts before obeying the instruction and will not continue until a manual switch is operated. This digit, the 'go' digit, serves two purposes. First as a useful facility when testing programs, and secondly for synchronizing the operation of the punched card input and output with the high-speed computer.

A standard punched card has 12 rows and 80 columns, of which only the first 32 columns are used by the Pilot ACE. Cards are read through the Hollerith reader row by row, and the 12 rows pass successively under a set of 32 reading brushes in the order $YX01\ldots9$. When a row of a card is in position under the reading brushes the number of that row is sensed by them and is available as a 32 digit binary number. Cards are read through the reader at the rate of 200 per minute and each row is capable of being read, while it is under the reading brushes, for a period roughly equal to 3 ms. The Pilot ACE can order a card to pass through the reader by making a transfer from any source to destination 31, the source being irrelevant. Usually source 0 is used for convenience when punching the instruction. The instruction

$$0-31$$

merely calls for a card to pass through the reader. Without further instructions nothing will be read from the card. As soon as a card starts to pass through the reader, source 0, which is normally controlled by the 32 input switches, ceases to be controlled by them and is supplied instead from the 32 reading brushes. When a row of a card is under the brushes, source 0 gives the number punched on that row. Between rows of the card source 0 gives zero. In order to ensure that a transfer from the cards takes place when a row is in position, an instruction calling for such a transfer has no 'go' digit and it is arranged that when a row of a card passes under the reading brushes the reader provides the equivalent of a single operation of the manual switch. An instruction without a 'go' digit is denoted by a cross

$$0-16\times.$$

If we wish to pass a card through the reader and to read the numbers on the Y row, the 0 row and the 1 row, ignoring the number on the X row, the following instructions are needed.

$0-31$	feed a card through reader
$0-15\times$	read Y row into $TS15$
$28-28\times$	a dummy instruction which is obeyed when the X row comes into position
$0-16\times$	read 0 row into $TS16$
$0-20\times$	read 1 row into $TS20$

The third instruction is necessary in order to make certain that the X row has passed before trying to read the 0 row. It has become customary to use the instruction 28–28 as a dummy instruction on the Pilot ACE. If there had been any other instructions to be obeyed between the second and fourth of the above group, then the 'go' digit could have been omitted from one of these and the dummy instruction would not have been necessary.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

The Hollerith punch output is operated and synchronized in a similar manner. The passage of a card is ordered by a transfer from any source to destination 30. As before, the source is irrelevant and consequently source 0 is used. As a card passes through the punch a 32-digit binary number may be punched on each of the 12 rows Y, X, 0, ..., 9 as it comes into position. The punch magnets are controlled by destination 28 which ceases to control the output lights as soon as a card starts to pass through the punch. Synchronization is achieved as before by omitting the 'go' digit in the instruction calling for a transfer to destination 28. Thus to punch the contents of *TS*16 on the Y row, *TS*20 on the X row and the sum of *TS*16 and *TS*20 on the 1 row the following instructions are needed.

0–30	feed a card through the punch
16–28×	punch from *TS*16 on to Y row
20–28×	punch from *TS*20 on to X row
20–17×	add number in *TS*20 to that in *TS*16
16–28×	punch sum on 1 row

Here the fourth instruction has its 'go' digit omitted and this ensures that the 0 row is passed before proceeding to the fifth instruction. Cards may be punched at the rate of 100 per min, that is, at half the speed at which they may be read. The speeds of reading and punching may be expressed as 1280 binary digits per sec and 640 binary digits per sec respectively.

Once a card passage has been initiated the whole of the card will pass; that is, we cannot stop it at an intermediate row. It is convenient to have some means of recognizing when the last row of a card has passed. This is achieved by means of a special source, source 30, which gives a zero output at all times except for a period which starts some time before the 12th row of a card arrives in position and persists until some time after it has passed. The signal is provided by both the read and the punch mechanisms, and the exact period for which it lasts has been carefully determined in the light of programming experience to be as useful as possible. The signal is referred to as T.I.L. (13th impulse line) because the Hollerith gives 12 other signals, one for each line.

The maximum period which may separate two instructions calling for transfers from two consecutive rows of a card when reading is 15 ms. The corresponding figure for transfers to a card when punching is 35 ms. Since the transfer instructions have no 'go' digit there is no danger of reading from (or punching on) the same row twice. As a result of optimum coding the number of instructions which may be obeyed in these periods have maximum values of 240 and 560 respectively. Although it is seldom possible to achieve rates of operation as high as these, the ability to do a considerable volume of computation between rows of cards has been of fundamental importance on the Pilot ACE. Initial input of instructions into the machine is achieved in the following manner. The machine has a manual switch which clears the store, and starts the card reader. The effect of clearing the store is to make the instruction consisting entirely of zeros, which is equivalent to

0 0–0 0 0×

the next to be obeyed. This zero instruction has no 'go' digit, so that it is not obeyed until the first row of the first card in the Hollerith reader is in the reading position. Source 0

has been assigned specially to work from the input. Destination 0 is also assigned in a special manner. It is such that when a word is sent to it, that word becomes the next instruction to be obeyed and the N element in the current instruction is ignored. The zero instruction therefore has the effect of reading in the first word on the first card and making it the next instruction to be obeyed. After we have managed to input one instruction the rest of the input procedure is comparatively simple, but a discussion of the details of the method adopted on the Pilot ACE would be out of place in this paper.

7. Influence of the optimum coding facility on the form of fundamental subroutines

For obvious reasons it is important that the fundamental subroutines should be as fast as possible, and it is here that optimum coding is most effective. The subroutines are produced in a form in which the operands occupy fixed positions which are nearly always short access stores. The detailed specification of a subroutine will suffice to illustrate the principle.

Extraction of the square root of a double-length number

'Given a double-length number "a" which is treated as an integer in $DS14$, it produces $\sqrt{a}$ (also as an integer) in $TS16$. The subroutine uses $TS26$ and $TS27$ in addition to $DS14$ and $TS16$.'

The subroutines are designed in general to make the best possible use of the facilities available on the TS's and DS's, but it is an advantage to have some means of choosing which TS to use when several are equally convenient. The order chosen is $TS16$, $DS14$, $TS26$, $TS27$, $TS20$, $DS12$, $TS15$ in order of decreasing use. The advantage of having such a scheme is that if a main program uses several subroutines there is still quite a chance that $TS15$, for example, will not be used by any of them and may therefore be used as a temporary store by the main program.

An important feature of the Pilot ACE is that the speed of the simpler operations such as addition and subtraction is very much higher relative to multiplication than is true on an orthodox machine. If optimum coding were not used on the Pilot ACE, addition would take one major cycle and multiplication (if made completely automatic) would take two major cycles so that the incentive to use the multiplier would be very considerable. With optimum coding, however, addition and subtraction frequently take only 64 μs each, including the set up time of the instruction, so that the balance is quite different. A result of this is that whereas on an orthodox machine the division or the extraction of a square root will usually be coded using an iterative process which employs the multiplier, such as forming the sequence

$$x_{n+1} = x_n(2 - ax_n)$$

for the reciprocal of a, on the Pilot ACE division is programmed by a process which finds the quotient digit by digit by the long-division process, in which additions and subtractions only are used. Moreover, the adoption of digit-by-digit processes often makes it possible to devise subroutines which are much more satisfactory.

For example, experience with programming reveals that the most satisfactory form of division is that which divides a single-length divisor into a double-length dividend to give

a single-length answer, and similarly in extracting a square root it is convenient to take the square root of a double-length number, particularly if that number has been formed as the sum of squares or products of other numbers. Their superiority is due to their freedom from tiresome scaling factor considerations. The subroutines which are cast in this form are just as fast on the Pilot ACE as those which use a single-length dividend or take the square root of a single-length number. As an example of such a routine, that for dividing a double-length number by a single-length number is given. The program

TABLE 2*a*. PRELIMINARY CODING

	(2_0)	14_0–16		send more significant half of dividend to *TS*16
	(2_3)	2_6–17		add *P*31 to this half and then test if resulting number has a *P*32 in it, i.e. if it is negative. If so the dividend is too large to give a single-length answer for any single-length divisor
	(2_6)	(*P*31)		
	(2_8)	16–24	(*A*) positive (*A*+1) negative	
(*A*)	(2_{14})	25–16		send *P*1 to *TS*16 which will hold a marker digit which will move one position to the left for each repetition of the main cycle
			(the next 8 instructions form the cycle)	
(*B*)	(2_{11})	14_0–27		send 32 most significant digits of remainder to *TS*27
	(2_{13})	14–13	(2)	shift remainder and partial quotient one place to the left
	(2_{17})	16–25	(*C*) zero (*C*+1) non-zero	check if marker digit has become zero, i.e. passed *P*32 position
(*C*+1)	(2_2)	22–24	(*D*) positive (*D*+1) negative	test if divisor and remainder have same or different signs
(*D*)	(2_4)	25–13	(2)	if the same sign then add a *P*1 to the partial quotient and subtract the divisor from the partial remainder. N.B. Source 17 is used to do subtraction
	(2_7)	17–13_0	next instruction is *E*	
(*D*+1)	(2_5)	26–13_0		if different sign, then add divisor to remainder. Quotient unchanged
(*E*)	(2_9)	16–17	next instruction is *B*	move marker digit one place to the left and repeat cycle
(*C*)	(2_1)	14_e–16		cycle has been repeated 32 times; send quotient to *TS*16 from 14_e
	(2_{10})	24–13_e		add a *P*32 to the quotient
	(2_{12})	13_e–24	(*A*+1) positive (*A*+2) negative	this instruction together with 2_{12} tests if the first two digits of the quotient are the same or different. If they are the same then the quotient is too large for single-length representation
(*A*+1)	(2_{15})	29–29	self repeating	if this is not so then round off the quotient by adding a *P*1 for the least significant digit
(*A*+2)	(2_{16})	25–17	on to link to next program	

gives a failure indication and the machine stops if the quotient is larger than a single-length number. As is the case with many of the most fundamental subroutines, a very sophisticated use of the code is made. The subroutine may be specified as follows. Given a double-length number '*a*' in *DS*14 and a single-length number '*b*' in *TS*26 it produces the quotient $c=a\div b$ in *TS*16, the round-off rule being that the least significant digit in '*c*' is to be a 1. The arithmetic basis of the method used is the so-called non-restoring method of division.

The program consists of an initial part in which a test is first made to see if the dividend is so great that the quotient cannot possibly be single-length, whatever the value of the

single-length divisor. This will occur if the modulus of the dividend is greater than 2^{62} (regarded as an integer). This part is followed by a cycle which is repeated 32 times, and in each repetition a digit of the quotient is obtained. The original dividend and the successive remainders are stored in *DS*14. The quotient is also built up in *DS*14, and in each repetition of the cycle the remainder and current quotient are moved one place to the left thus making room for the next digit of the quotient. After 32 repetitions of the cycle the full quotient is in 14_e and is sent to *TS*16 to be rounded off. There is a second test to determine whether the result of the divisor is a single-length number. The first test may be

TABLE 2*b*. DETAILED CODING OF DIVISION SUBROUTINE

	single speed					double speed				
0	2	14_o–16	*s*	1	1 (3)	2	14_o–16	*s*	1	2 (4)
1	2	14_e–16	*s*	1	7 (10)	2	14_o–27	*s*	0	0 (3)
2	2	22–24	.	0	0 (4, 5)					
3	2	2_6–17	*s*	1	3 (8)	2	14–13	.	1	2 (7)
4	2	25–13	.	0	1 (7)	2	2_6–17	*s*	0	2 (8)
5	2	26–13_o	*s*	0	2 (9)					
6			(*P*31)					(*P*31)		
7	2	17–13_o	*s*	0	0 (9)	2	16–25	.	0	0 (9, 10)
8	2	16–24	.	0	4 (14, 15)	2	16–24	.	0	10 (20, 21)
9	2	16–17	*s*	0	0 (11)	2	14_e–16	*s*	1	3 (14)
10	2	24–13_e	*s*	0	0 (12)	2	22–24	.	0	0 (12, 13)
11	2	14_o–27	*s*	0	0 (13)					
12	2	13_e–24	*s*	0	1 (15, 16)	2	25–13	.	0	1 (15)
13	2	14–13	.	1	2 (17)	2	26–13_o	*s*	0	2 (17)
14	2	25–16	.	0	27 (11)	2	24–13_e	*s*	0	0 (16)
15	2	29–29	.	0	30 (15)	2	17–13_o	*s*	0	0 (17)
16	*N*	25–17	*s*	0	*m* (to link)	2	13_e–24	*s*	0	3 (21, 22)
17	2	16–25	.	0	14 (1, 2)	2	14_o–27	*s*	0	0 (19)
18										
19						2	14–13	.	1	2 (23)
20						2	25–16	.	0	11 (1)
21						2	29–29	.	0	30 (21)
22						*N*	25–17	*s*	0	*m* (to link)
23						2	22–24	.	0	0 (25, 26)
24										
25						2	17–13_o	*s*	0	0 (27)
26						2	26–13_o	*s*	1	2 (30)
27						2	25–13	.	0	1 (30)
28										
29										
30						2	16–17	.	0	1 (1)
31										

said to deal with the possibility of a 'gross' failure and the second to deal with all other cases of failure. The second test merely detects whether the first two digits given by the repeated cycle are the same or different. If they are the same the answer is not a single-length number. The failure instruction 2_{15} may be reached either as a result of failing the first test, that is from instruction 2_8, or the second test, that is from instruction 2_{12}.

The repeated cycle of instructions is quite short, and by duplication two digits of the quotient may be obtained per major cycle. It is not necessary to duplicate all 8 instructions in the repeated cycle. Counting is performed by means of a marker digit in *TS*16 which is moved one position to the left per cycle until it runs off the end. The instruction 2_9 provides this left shift and 2_{17} is used to detect when the digit has run off the end. We

need not duplicate these instructions; instead we may shift the marker digit two positions to the left and then discriminate after obtaining each pair of digits in the quotient.

With this small amount of duplication the time of division is reduced to 16 ms in spite of the fact that a double-length dividend has been used. Before the Pilot ACE was equipped with a magnetic drum, the shortage of storage space was an overriding consideration and the above method of doubling speed was seldom employed, but is now becoming much more common. The detailed coding is given of both the single- and double-speed programs and the preliminary coding of the single-speed version only. In the first example of the detailed coding (table 2*a*) the instructions are compact in the sense that no gaps are left. In the speeded up version (table 2*b*) it is not possible to achieve this. For the sake of tidiness programs are coded in a compact form whenever possible, but if gaps are left it is not particularly inconvenient since they may easily be filled by instructions from the main program. The fact that each instruction selects the position of the next, means that no 'unconditional transfer' instructions are needed in order to fill the gaps.

8. Assessment of the value of optimum coding

It will be realized from what has gone before that it is not possible to give a precise figure for the gain in speed of operation due to optimum coding, since this varies considerably from problem to problem. In the author's experience the gain over an orthodox machine with a major cycle of the same length is hardly ever less than two and is usually considerably greater. On a large range of important problems it is of the order of ten. However, in practice on the Pilot ACE perhaps the most important advantage that the use of optimum coding has given, is that it has made possible a mode of operation in which cards are used as an intermediate store for numbers in binary form, and in which all computation is performed between the rows of the cards while they are being read or punched. The 15 ms available between rows of cards when reading would seldom be long enough to do anything useful on an orthodox machine but on the Pilot ACE a large number of operations are possible. This facility has been particularly valuable in the field of matrix algebra and even before the Pilot ACE was equipped with a magnetic drum, programs existed and were regularly used for finding the latent roots of matrices, symmetric and unsymmetric, up to order 63, inverting matrices up to order 190 and multiplying together matrices up to order 190 without partitioning. Many of these programs were considerably faster than would have been possible on an orthodox machine even with a high-speed store sufficiently large to accommodate all the data. In fact, whenever the number of instructions carried out between rows of cards is greater than about 18 the program is necessarily faster than is possible on an orthodox machine, because there is less than 18 ms between the rows. The following are examples of the speeds of some typical programs, counterparts of which will exist on most machines which are in regular operation. They include examples where all the data can be held on the drum and some where cards are used as an intermediate store.

(i) Solution of a set of linear equations of order 30, with all the data stored on the drum, by the method of elimination. The program includes arithmetic checks on the reduction and an automatic means for avoiding difficulties due to a principal minor being very small or equal to zero. 90 s computing time.

(ii) Calculation of the dominant latent root of a matrix of order 30 by iteration. Each step consists of forming the vector $y_{n+1}=Ax_n$ from the vector x_n, followed by the normalization of y_{n+1} to give x_{n+1}. An arithmetic check on the computation is included. All the data are on the drum. Time per iteration 5 s.

(iii) Integration of system of n ordinary differential equations by the Gill (1951) modification of the Runge-Kutta process. Time $55n$ ms per step plus the time taken to calculate the auxiliary functions.

(iv) Subroutines for floating arithmetic carried out on two-word aggregates with one word for the radix and one for the index. 5 ms for floating multiplication, $(5+\frac{1}{2}q)$ ms for floating addition and subtraction where q is the number of significant zeros arising due to cancellation. 35 or 19 ms for floating division according as the single-speed or double-speed division process is used.

(v) Solution of a set of linear equations of order 100 with no zero coefficients by the method of triangular decomposition (sometimes known as the unsymmetrical Choleski method), using cards as an intermediate store. Time 160 min plus some card handling time which is of the order of 30 min. The program has the advantage that in the event of a breakdown the process may be continued from the point immediately prior to breakdown since all results to date are on cards. There are arithmetic checks on the matrix operations involved and a reading and punching sum check on all the results produced.

The above set of programs are fairly representative of the variation in the gain from optimum coding. Thus (i) and (ii) compare quite well with the speed achieved on the fastest parallel machines now in existence especially considering the fairly complete nature of the arithmetic checks they include.

Program (iii), on the other hand, is only about four times as fast as the corresponding program on EDSAC which has the same major cycle. The poor gain here is partly due to shortcomings in the facilities on the machine and to the importance of multiplication.

In program (iv) addition, subtraction and multiplication are about ten times as fast as similar programs on an orthodox machine, while floating division in its faster form is at least three times as fast.

Program (v) has, of course, no true counterpart on an orthodox machine, but comparison with routines dealing with smaller orders on other machines suggests that it is at least twice as fast as would be achieved on an orthodox machine even if it had sufficient high-speed store to hold all the data, which would require 312 delay lines.

Optimum coding has also been of great value in speeding up the rates of input and output of decimal data. The Hollerith decimal code is peculiarly unsuited to the process of binary to decimal or decimal to binary conversion unless the cards are fed endwise. Suitable input and output equipment using an endwise feed was not available, and as a result the problem of converting a decimal number which was read row by row had to be faced. By very careful optimum coding it has proved possible to read cards at the rate of 200 per min, each punched with three 9-digit decimal numbers with signs, and to do the conversion while the cards are feeding. This is the maximum amount of information that can be punched on the 32 columns. If the input were modified so that more than 32 columns could be read, refinements in the coding would make it possible to achieve higher rates of conversion. In the same way three 9-digit decimal numbers may be punched

per card (or the equivalent number of smaller numbers), though here since the punch operates at 100 cards per min the requirement is not so severe. In problems where the output of final results is considerable, as is the case in solving ordinary differential equations when the values of the dependent variables and their derivatives are needed at each step of integration, the speed of punching is frequently the limiting factor, so it is important that this should be as high as possible. The speeds of reading and punching may be expressed as 90 and 45 decimal digits per sec respectively. If data are in binary decimal form then 300 decimal digits per sec may be read and 150 per sec punched.

The price which has to be paid for optimum coding in terms of extra time taken to prepare programs is even more difficult to assess than the gain in speed. Again it is complicated by the fact that many of the more tiresome features of coding on the Pilot ACE are not related to the use of optimum coding but rather to economy of equipment. Moreover, in most problems the extra time consumed will depend on how much importance is attached to reducing machine time to a minimum, but there are two important considerations which should be stressed.

The first is that it is found in practice that there are usually quite large sections of programs in which high speed is relatively unimportant because these sections are non-repetitive. Here there is no need to make any attempt to gain more from optimum coding than is achieved without special effort. Optimum coding is important mainly in the repeated cycles and particularly in the subroutines, and here fortunately it is usually simplest because the operands involved are stored in *TS*'s. The time taken to prepare fundamental subroutines and important basic major routines such as the inversion of matrices and the calculation of their latent roots may be twice that which would be necessary without optimum coding, but here it seems that the gain in speed makes the effort well worth while.

The second consideration is that for most programs it is found that the main labour is in the preparation of the preliminary coding since it is here that the general plan of attack is developed. The detailed coding is never started until the preliminary coding has taken on its final form. Its production is then a routine task and much of it may be carried out by relatively unskilled labour.

With the development of fast and reliable methods of storage which are suitable for parallel operation it is improbable that optimum coding will be important on the most advanced machines to be built in the next decade. However, experience with the Pilot ACE has shown that it is possible to build a very fast machine which is surprisingly inexpensive in terms of equipment, and it is likely that such machines have a future for groups of users such as exist for example in the aircraft industry. Such groups are likely to use a machine for a considerably narrower range of problems than are treated by a general computing organization such as exists at the National Physical Laboratory. They could, therefore, afford to expend extra effort on making the more important routines as fast as possible. Much of the work of the aircraft industry for example lies in the field of matrix algebra and involves operations on matrices which are frequently of high order. The Pilot ACE probably deals more effectively with this type of work at a high speed than could any other existing machine with a comparable amount of equipment.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

9. Possible improvements in optimum coding

The Pilot Model has proved so effective a computer that in spite of its limitations, copies are in production under the name DEUCE. A few minor modifications have been made in these machines such as rationalizing the numbering of the sources and destinations so that, for example, the numbers of the short stores are consecutive. In addition, an automatic divider and a subtractive input on the long accumulator have been added. A certain amount of experimental programming has been done in order to determine whether 5 *TS*'s and 2 *DS*'s are the best combination of short access stores. This has shown that for many purposes a store holding four words is almost as valuable as four *TS*'s, and accordingly the DEUCE will have 4 *TS*'s, 3 *DS*'s and 2 stores holding four words each, which will be called *QS*'s. The limited experience with this distribution suggests that in many problems the gain will be considerable. For example, the program for calculating the latent roots of symmetric matrices by Jacobi's (1846) method as described in Taussky & Todd (1952) proved twice as fast with the new distribution and needed far less effort to achieve this measure of optimum coding. This is because in Jacobi's process there are rather more numbers than usual to which frequent access is needed. Even in examples where little or no further gain in speed was achieved it was frequently true that less attention to detail was needed to achieve the same effect.

However, a few minor modifications cannot be expected to remove the glaring deficiencies of the Pilot ACE. The design has now been drawn up of a full-scale machine which is not unduly large and which is essentially the original ACE with improvements which have been incorporated as a result of experience with the Pilot machine. The instruction word is of the four-address type in order to make it more powerful. This seems important on a machine in which the availability of instructions is of primary importance. The instruction is of the form

$$N \quad S_1 \quad F \quad S_2 \quad D \quad C \quad W \quad T \quad G$$

where N is again the *DL* which holds the next instruction and C the characteristic. Permissible instructions are of the type

$$S_1 + S_2 \text{ to } D$$
$$S_1 - S_2 \text{ to } D$$
$$S_1 \text{ and } S_2 \text{ to } D, \text{ etc.},$$

where the addition and subtraction may be with carry suppression appropriate to either single-length or double-length numbers. The timing of the instruction will be absolute and not relative as on the Pilot ACE. As was true on the Pilot ACE the addresses S_1, S_2 and D will not be complete addresses, and we shall only be able to add together two words in corresponding positions in two long delay lines. In general, it will be found that at least one of the addresses will be that of a short access store, though the prolonged transfer may be used, for example, to achieve the addition of two vectors whose corresponding elements are stored in the same minor cycle positions in two *DL*'s. It is intended to eliminate the minor cycle which was wasted on the Pilot ACE in setting up the source and destination gates, so that instructions may be carried out at a maximum rate of 32 per major cycle. A serious weakness of the Pilot Model is that when a group of numbers

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

is stored in consecutive positions, then the instruction needed to extract these numbers one by one in a repeated cycle, must be modified by unity in its W element for each repetition of the cycle. Unfortunately, when we reach the end of one delay line we do not pass automatically to the beginning of the next. To meet this situation there are a number of functions on the full-scale machine called 'modified' addition such that when they are used to add into the W position in an instruction then any carry over past the fifth digit of W is added into any, all, or none of the S_1, S_2 and D elements. For many subroutines the number of instructions needed with this code is only about half the number needed on the Pilot ACE and this, together with the increased rate at which instructions may be carried out and the improved facilities available, is expected to give an average gain in speed of about three or four to one over the Pilot ACE, besides a considerable simplification in programming.

The term four-address code has been used to describe this code, but it should be realized that as on the Pilot ACE the addresses are incomplete. If we agree to denote an address

TABLE 3. PILOT MODEL OF THE ACE: SOURCES, DESTINATIONS AND NEXT INSTRUCTION SOURCES

	sources		destinations		next instruction sources	
0	I.D. or card reader	0	instruction	0	$DL11$	
1	$DL1$	1	$DL1$	1	$DL1$	
2	$DL2$	2	$DL2$	2	$DL2$	
3	$DL3$	3	$DL3$	3	$DL3$	
4	$DL4$	4	$DL4$	4	$DL4$	
5	$DL5$	5	$DL5$	5	$DL5$	
6	$DL6$	6	$DL6$	6	$DL6$	
7	$DL7$	7	$DL7$	7	$DL7$	
8	$DL8$	8	$DL8$			
9*	$DL9$	9*	$DL9$			
10	$DL10$	10	$DL10$			
11	$DL11$	11	$DL11$			
12	$DS12$	12	$DS12$			
13†	$(DS14) \div 2$	13†	$DS14$ add			
14	$DS14$	14	$DS14$			
15	$TS15$	15	$TS15$			
16	$TS16$	16	$TS16$			
17	$\sim(TS26)$	17	$TS16$ add			
18	$(TS26) \div 2$	18	$TS16$ subtract			
19	$(TS26) \times 2$	19§	multiply			
20‡	$TS20$	20	$TS20$			
					characteristic	
					zero	one
21	$(TS26)$ & $(TS27)$	21§	TCA		off	on
22	$(TS26) \not\equiv (TS27)$	22	magnetics		read	write
23	$P17$	23§	TCB		off	on
24	$P32$	24	discriminate on sign			
25	$P1$	25	discriminate on zero			
26	$TS26$	26	$TS26$			
27	$TS27$	27	$TS27$			
28	zero	28	O.P.S. or card punch			
29	ones	29	buzzer			
30	T.I.L.	30§	punch			
31	—	31§	read			

* Associated with magnetics.
† Modified by TCB.
‡ Modified by TCA.
§ Independent of source used.

by a pair of numbers (A/B) the first, A, denoting the number of the delay line and the second, B, the position in the delay line then the instruction word on the ACE may be brought more in line with an orthodox four-address code by writing it in the form $(S_1/W)+(S_2/W)$ to (D/W) (N/T). When written in this form the instruction is far less formidable than the expanded form given above. The length of the instruction as given there is mainly due to the fact that addresses have been split up into delay line numbers and minor cycle portions in delay lines. It will be seen that there are four complete addresses of which the first is that of the first operand, the second that of the second operand, the third the destination of the result and the fourth the address of the next instruction. The first three addresses are not independent since we can only operate on numbers in the same positions in the long delay lines. When one of the addresses applies to a short access store the W number is irrelevant. The characteristic merely determines the period of the transfer. The division of the address into two parts is not particularly inconvenient and in fact has the advantage of keeping the numbers small and in a form in which they are easily recognized in the machine. On those machines in which addresses are written on a hexadecimal scale a similar separation is adopted in any case. On true four-address code machines the four addresses are all complete and independent, but the time taken to obey instructions is usually quite long and little advantage in speed arises from the multi-address nature of the instructions. On this full-scale machine the numbers from the two sources S_1 and S_2 will be extracted simultaneously together with the next instruction.

This paper has been written as part of the research program of the National Physical Laboratory, and is published with the permission of the Director of the Laboratory.

References

Alway, G. G. 1954 *Proceedings of a Symposium held at the National Physical Laboratory*, p. 65. (London: H.M.S.O.)

Babbage, H. P. 1889 *Babbage's calculating engines.* (Spon and Co., London, 1889.)

Freedman, A. L. 1954 *Proc. Camb. Phil. Soc.* **50**, 426.

Gill, S. 1951 A process for the step-by-step integration of differential equations in an automatic digital computing machine. *Proc. Camb. Phil. Soc.* **47**, 96.

Jacobi, C. G. J. 1846 *J. reine angew. Math.* **30**, 51.

Taussky, O. & Todd, J. 1952 *Math. Mag., Pacoima*, **26**, 71.

Wilkes, M. V. 1948 *Proc. Roy. Soc.* A, **195**, 265.

Wilkes, M. V. 1949 *J. Sci. Instrum.* **26**, 385.

Wilkinson, J. H. 1954 *Proceedings of a Symposium held at the National Physical Laboratory*, p. 5. (London: H.M.S.O.)

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES