A total of 2206 vacuum tubes are used in the computer. Over half of these tubes are high-vacuum diodes, types 6AL5, which were used in preference to crystal diodes for matrix circuits.

Design and construction of the SPEC started in July 1949, and the first problem was solved on February 1, 1950. It was in continuous operation at the NEPA Project from the spring of 1950 until February 1951 and is being installed at the Oak Ridge National Laboratory. The computer will be called the "ORACLE" which is derived from Oak Ridge Automatic Computer for Linear Equations.

During the operating period, the computer was in an operating condition for approximately 85% of the total available time, with 15% of the time devoted to testing and servicing. Most machine errors were detected during the solution of problems, although test problems were used for periodic checks to assure proper operation of the machine. Check circuits, with visual and audible indications, are employed to detect improper operation of the machine in the most critical places, but no complete checking system is employed.

Throughout the period of operation, the computer was exceedingly useful as a means of obtaining solutions to systems of simultaneous, linear equations up to the limit of its useful capacity. In addition, problems involving matrix products, Fourier analysis, numerical integration, and matrix inversion were undertaken with considerable savings in time and effort.

The author wishes to acknowledge the outstanding contribution of Mr. V. G. LEWIS and his staff of technicians at the NEPA facility in Oak Ridge in the construction and assembly of the computer. Special recognition is due Mr. L. C. OAKES for his valuable contributions in testing, servicing and operating the machine.

J. J. STONE

Oak Ridge National Laboratory
Oak Ridge, Tenn.

[1] Nuclear Energy for the Propulsion of Aircraft Project, conducted by Fairchild Engine and Airplane Corporation under special contract with the United States Air Force.

DISCUSSIONS

FLOATING OPERATIONS ON THE EDSAC

Summary. The difficulties which arise when programming calculations for large automatic calculating machines which have a fixed decimal point are discussed. This leads to a consideration of the possibility of using floating decimal arithmetic for certain kinds of calculations. A method by which floating decimal arithmetic can be carried out with any fixed decimal-point machine is outlined and the scheme adopted for use with the EDSAC is described in detail.

This scheme is based on a special kind of subroutine which we call *interpretive.* This enables the programmer to use a new order code of his own choice. The 'orders' of a programme drawn up with such a code are selected under the action of the interpretive subroutine and interpreted in terms of sequences of orders which perform the required operations. With the EDSAC

a single address 'order' code has been adopted, similar to the ordinary order code, in which the arithmetical 'orders' are interpreted in terms of floating decimal arithmetic. Special 'orders' for simplifying counting operations and the modification of other 'orders' are provided. The 'order' code is described in detail and an example is given of a programme drawn up using this code. Other topics discussed are the use of auxiliary subroutines with the interpretive subroutine, a method of facilitating programme assembly, and techniques for using the input tape as a form of auxiliary store. Finally the times of operation of the individual 'orders' of the code are given, together with an estimate of the factor by which the calculation time is increased as a result of using floating decimal arithmetic for an entire calculation.

Introduction. One of the most tedious tasks arising in the programming of calculations for solution by fixed decimal-point machinery is to arrange the calculation so that all quantities concerned remain within the limits of the machine and yet are expressed to an accuracy sufficient to ensure the desired precision in the results. Our experience with the EDSAC has shown that if this task can, in some way, be relegated to the machine then the time taken to prepare a programme after a problem has been understood is considerably reduced.

The problem does not arise with machines designed to operate directly with numbers expressed in the floating radix form. Numbers in this form are represented by $a \cdot r^p$. The first machine of this kind was the Bell Telephone Laboratories Relay Computer Model V[1]. This is a decimal machine (that is, $r = 10$) in which $1 > |a| > 0.1$, $19 \geq p \geq -19$ and a is expressed to an accuracy of seven significant figures. Since this was completed all important relay machines have been equipped with similar facilities. No electronic machine of this kind has yet been built but we would remark that in our opinion an electronic machine provided with a floating point arithmetical unit would be a powerful computing instrument even if it had a relatively slow store, a magnetic drum, for example. It would be particularly suitable for a laboratory which had to solve quickly a wide variety of problems as they are presented.

For a fixed decimal-point machine the usual arrangement is to associate scale factors with some or all of the quantities occurring in the calculation. These scale factors are of two kinds. First certain quantities can be associated with *fixed scale factors* which remain unaltered throughout the calculation. Their value must be chosen so that the quantities concerned do not exceed capacity and yet can be represented at all times with sufficient accuracy. This is the problem of providing 'elbow room.' It may not be possible to satisfy both requirements by the use of a single fixed scale factor and it is then necessary to introduce an *adjustable scale factor*, that is, a scale factor which is altered during the course of the calculation. It may be adjusted either continually or occasionally in accordance with some criterion or it may take a preassigned set of values. All these things the coder has to think about if the calculation is to be arranged to use the least possible machine time. The difficulties arising will vary from one calculation to another and may be trivial, moderately complex, or really hard. The individual nature of calculations and the necessity for a mathematical understanding of them, however, makes a general solution to the problem on these lines unattractive to attempt. The only way in which the difficulties may be avoided is to associate every number occurring in the calculation with its own ad-

justable scale factor. The scale factors can be stored most economically if they are powers of 2 or of 10. In the latter case the above scheme virtually amounts to programming a floating decimal point. This is the solution that we feel should be adopted for many kinds of calculation.

The floating decimal form of representation of numbers is especially useful when it is required to evaluate algebraic expressions. For example, to code the evaluation $(au^2 + bu + c)/(du + e)$ with a fixed decimal-point machine can be very troublesome if it is required to maintain accuracy over the entire range of numbers. If the reader has any doubts about this he is advised to try it. If floating decimal arithmetic is used, then the coding requires little thought and is quite direct.

It is the purpose of this paper to describe a method by which floating decimal arithmetic can be carried out on fixed decimal-point machines, and in particular to describe the scheme adopted for the EDSAC.

General considerations. The following remarks are fairly general and apply to almost any machine although certain features mentioned, for example, short and long locations, are made with the EDSAC in mind.

The most convenient way to programme operations on numbers expressed in the floating decimal form is by means of a special type of subroutine which we call *interpretive.* This enables words similar to those representing ordinary machine orders to be interpreted in terms of floating decimal operations. Such words will be referred to as 'orders' (with quotation marks). Each 'order' resembles an ordinary machine order but is never obeyed as such by the control circuits of the machine. Instead, 'orders' are selected in a definite sequence by the action of the interpretive subroutine and interpreted in accordance with a preassigned 'order' code, by means of sequences of orders which form part of the interpretive subroutine. The selective action of the interpretive subroutine will be referred to as the 'control' (in quotation marks). The advantage of such interpretive subroutines is that the 'order' code may be chosen to suit the convenience of the programmer. There need be no relation between the form of the 'order' code and the ordinary code of the machine. However, we have become so familiar with the ordinary EDSAC order code that a similar single address 'order' code has been adopted for the interpretive subroutines. Interpretive subroutines have also been placed in the EDSAC library for carrying out arithmetical operations on complex numbers and on double-precision numbers.

When representing floating decimal numbers in a fixed decimal-point machine it is most economical to pack both the numerical part and the exponent into a single storage location. This means, of course, that the numerical part of the number has fewer binary digits to represent it than would normally be available. However, they are all significant whereas in fixed decimal-point working digits are thrown away to provide 'elbow room.' In both cases accuracy may be lost owing to cancellation of leading digits at one end of a number and to rounding-off errors at the other.

Two long and two short storage locations are set aside to form a kind of 'arithmetical unit.' One long location holds the numerical part of a number and one short location holds the exponent. Together they form the *floating decimal accumulator.* In a similar fashion the other long location and the other short location form the *floating decimal register.*

An arithmetical 'order' first causes a subroutine to select and unpack the operand and place the numerical part and the exponent into a further pair of storage locations—the 'multiplier register.' In the case of an add 'order' the operand is then added to the number held in the floating decimal accumulator. This is done as follows. Let a_0, a_a, and a_s denote the numerical parts of the operand, augend, and sum respectively, and let p_0, p_a, and p_s similarly denote their exponents. Then we have

$$\begin{aligned} a_s 10^{p_s} &= [a_0 + a_a 10^{-(p_0 - p_a)}]10^{p_0} \quad p_0 \geq p_a \\ &= [a_0 10^{-(p_a - p_0)} + a_a]10^{p_a} \quad p_0 < p_a. \end{aligned}$$

The *subtract* 'order' works in a similar fashion. A *register* 'order' places the operand in the floating decimal register. Other 'orders' enable the product of the operand with a number held in the floating decimal register to be added to or subtracted from the number held in the floating decimal accumulator. The *transfer* order causes the number held in the floating decimal accumulator to be converted to standard form, packed, and finally transferred to the store; the floating decimal accumulator is then 'cleared' by replacing the number held in it by zero, that is, by the special number 010^{-63}. The *input* 'order' causes the two parts of a number to be read from the input tape, packed, and transferred to the store. The *output* 'order' causes the numerical part and the exponent of the number held in the floating decimal accumulator to be printed on the same line in two adjacent columns on the page of the teleprinter.

The use of two separate storage locations for the floating decimal accumulator allows the range and accuracy of numbers held therein to be greater than those held in a single storage location elsewhere. This enables products to be accumulated without loss of accuracy due to intermediate rounding-off errors.

The scheme adopted for the EDSAC. The interpretive subroutine that has been developed to carry out floating decimal arithmetic with the EDSAC will now be described in more detail. Throughout this and the following section references to specific features or conventions used with the EDSAC have been avoided as far as possible but the following terms are used at least once: *initial orders*, *control combinations*, *preset parameters* (not to be confused with the *current parameter* defined below), *short* and *long* locations and the *E* order. Detailed descriptions of all these features can be found in reference 2.

The following abbreviations will be adopted:

$F(A)$	denotes the floating decimal number held in the floating decimal accumulator
$F(R)$	denotes the floating decimal number held in the floating decimal register
$S(mD)$	denotes the long storage location having address m
$S(mF)$	denotes the short storage location having address m
$F(mD)$	denotes the floating decimal number held in $S(mD)$
$F(mF)$	denotes the floating decimal number held in $S(mF)$
$C(mD)$	denotes the ordinary number held in $S(mD)$
$C(mF)$	denotes the ordinary number held in $S(mF)$

Number representation. The method described above of packing the numerical part a and the exponent p of a floating decimal number $a \cdot 10^p$ in a

single storage location has been adopted. With the EDSAC it amounts to representing the floating decimal number by the ordinary (fractional) number $p2^{-6} + a2^{-7}$, where a is expressed with an accuracy of 28 binary digits and lies in the range $(1 \geq |a| \geq 0.1)$ and p is an integer such that $63 \geq p \geq -63$. Zero has the special representation 010^{-63}. When numbers are transferred from the floating decimal accumulator to the store they are automatically transferred to the standard representation.

The 'order' code. Since the 'orders' are similar in form to the ordinary machine orders of the EDSAC they will be presented in the conventional form adopted for the latter. The 'orders' of the code fall into two classes, *arithmetical* and *organizational*. The arithmetical 'orders' can refer either to short or to long storage locations according as the 'order' is terminated with the code letter F or D. This is the usual EDSAC convention. If, however, these 'orders' are terminated by $\pi\Delta$ instead of D or Δ instead of F then a number, known as the *current parameter*, held in a certain storage location, will be added to the address of the order before it is obeyed. Among the organizational 'orders' are two—the P and the F orders—which enable the current parameter to be set to some initial value and subsequently adjusted after each cycle of a repetitive operation. In this way an arithmetical 'order' can refer to different numbers during different cycles of the calculation. These orders are described in more detail below. The scheme is similar in some respects to the way in which orders are modified in the Manchester University Electronic Computer Mk. 2. In this machine a built-in facility enables a number held in one of 8 registers—called the B registers—to be added to the address of an order immediately before that order is obeyed.

The arithmetical 'orders'

$A\ m\ D$ Add $F(mD)$ to $F(A)$

$B\ m\ D$ Subtract $F(mD)$ from $F(A)$

$H\ m\ D$ Replace $F(R)$ by $F(mD)$

$V\ m\ D$ Add the product $F(R) \cdot F(mD)$ to $F(A)$

$N\ m\ D$ Subtract the product $F(R) \cdot F(mD)$ from $F(A)$

$D\ m\ D$ Replace $F(A)$ by $F(A)/F(mD)$

$\phi\ m\ D$ Replace $F(mD)$ by $F(A)$ and $F(A)$ by 0.10^{-63}

$L\ m\ D$ Similar to $\phi\ m\ D$ but in addition the non-standard content of the floating decimal accumulator is printed as negative sign or space; exponent (2 figs.); space; negative sign or space; 8 decimal digit fraction. For example, -98.283742 would be printed as 02 − 98283742.

$\Delta\ m\ D$ Input a sequence of numbers on the tape terminated by X into the locations mD, $(m-2)D$, etc. Each number is punched in the following way: Characters to represent the exponent; sign; numerical part. In the numerical part the decimal point is understood to be before the first digit punched. Any number of digits may be punched. For example, -98.283742 would be punched as 2 − 98283742. X is punched before the first number of the sequence and then the sequence is copied on to the data tape in the reverse direction. Numbers read into the machine must have exponents in the range -16 to $+15$, so that it can be represented by a single tape character.

The organizational 'orders'

$M\ m\ F$ Conditional order: if $F(A) > 0$ transfer 'control' to the 'order' which stands in $S(mF)$; otherwise replace $F(A)$ by $|F(A)|$ and proceed with the next 'order.'

$C\ m\ F$ Transfer control of the machine to the order which stands in $S(mF)$, that is, return to the machine order code.

$X\ m\ F$ Transfer 'control' to the X-auxiliary whose entry 'order' stands in $S(mF)$. When the X-auxiliary is completed it will return 'control' to the 'order' immediately following the X 'order.'

$G\ m\ F$ Transfer 'control' to the 'order' standing in $S(mF)$.

$P\ m\ F$, $F\ m\ F$ The function of these 'orders' is described below

The P and F 'orders.' These orders are provided to facilitate the coding of cycles of orders and of cycles within cycles, etc. Each cycle of 'orders' or 'loop' is started by a P 'order' and terminated by an F 'order.' The two are said to *correspond.* This correspondence is similar to that between left and right handed brackets in a lengthy algebraic formula if the direction left to right in the formula corresponds to the direction in which 'control' is normally advanced. Associated with each loop of 'orders' is a parameter whose value is set, initially, by the P 'order' and is subsequently adjusted by the F 'order.' The current parameter (see above) associated with any particular 'order' of a loop is the parameter set by the P 'order' which the 'control' encountered last. It is now possible to give a formal description of the function of these 'orders.'

$P\ m\ F$ Record a new parameter—the current parameter of the following 'orders'—and set it to the value $-m2^{-15}$

$F\ m\ F$ If the current parameter is $< -m2^{-15}$, increase its value by $m2^{-15}$, return 'control' to the 'order' immediately following the corresponding P 'order'; otherwise proceed with the following 'order' after restoring the current parameters to the value it had before the corresponding P 'order.'

The above description should be sufficient to enable a programmer to use the P and F orders in a programme. We now give a detailed description of the means by which the effects of these 'orders' are achieved.

The number of parameters has been limited to 3. This restriction, although not essential, is reasonable as problems involving loops within loops more than three 'deep' are likely to occur very rarely.

The parameters are stored together with the addresses of the locations of the corresponding P 'orders,' in 3 long storage locations $S(hD)$, $S(h+2)D$, and $S(h+4)D$. At any stage in the programme $S(hD)$ contains the current parameter, $S(h+2)D$ the previous parameter, and $S(h+4)D$ the previous parameter but one.

A P 'order,' for example, $P\ m\ F$ in $S(nF)$ causes $C(h+4)D$ to be replaced by $C(h+2)D$, $C(h+2)D$ to be replaced by $C(hD)$, and $C(hD)$ to be replaced by the number $-m2^{-15}+n2^{-33}$. When the corresponding F 'order,' $F\ q\ F$ is encountered the following operations take place. The size of the current parameter (the number $-m2^{-15}$ in the short storage location $S(h+1)F$) is tested. If this number is less than $-q\cdot 2^{-15}$ it is increased by the amount $q\cdot 2^{-15}$ and 'control' is transferred to the 'order' which stands in

$S(n+1)F$, where $n \cdot 2^{-15}$ is the number in the short location $S(hF)$. Otherwise $C(hD)$ is replaced by $C(h+2)D$, $C(h+2)D$ is replaced by $C(h+4)D$ and then 'control' proceeds with the next 'order.'

Example. The action of the 'orders' and the context in which they are meant to be used may be more clearly understood from a study of the following programme for the calculation of a root of an algebraic equation by the Newton-Raphson iterative process.

Location of data: let $f(x) \equiv \sum_{r=0}^{r=n} a_r x^{n-r} = 0$ be the equation to be solved.

It is assumed that the coefficients $a_n, a_{n-1}, \cdots, a_0$ are in $S(100\text{D}), S(98\text{D}), \cdots, S(100-2n)\text{D}$. The initial approximation x_0 stands in $S(6\text{D})$ and all subsequent approximations are placed there. A small quantity δ, used in the convergence criterion, stands in $S(8\text{D})$.

Formula used: the iterative formula

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

is used. The iteration is arranged to terminate when $|x_n - x_{n-1}| < \delta$. $f(x)$ is calculated from the recurrence relations

$$q_0 = a_0; \quad q_{r+1} = q_r \cdot x + a_{r+1}; \quad q_n = f(x)$$

and $f'(x)$ is calculated from the similar recurrence relations

$$q_0' = 0; \quad q_{r+1}' = q_r' \cdot x + q_r; \quad q_n' = f'(x).$$

The orders of the programme are listed below. For those readers who are not familiar with the EDSAC conventions it may be stated that the code letter θ provides for a system of numbering relative to the order immediately following a control combination G K. Thus, in the example below, the 'order' M 18 θ refers to the 'order' B 8 D.

	G		K	
0	ϕ		D	'clear' floating decimal accumulator
1	ϕ	10	D	'clear'; $S(10\text{D})$
2	H	6	D	place x_n in the floating decimal register
3	P	$2n+2$	F	
4	ϕ	12	D	
5	V	10	D	$p_{r+1}' = xp'_r + p_r$
6	A	12	D	
7	ϕ	10	D	
8	V	12	D	$p_{r+1} = xp_r + a_{r+1}$
9	A	102	$\pi\Delta$	cycle $n+1$ times
10	F	2	F	
11	D	10	D	$f(x_n)/f'(x_n)$ to $S(10\text{D})$
12	ϕ	10	D	
13	B	10	D	
14	A	6	D	$x_{n+1} = x_n - f(x_n)/f'(x_n)$
15	ϕ	6	D	
16	A	10	D	form modulus of $(x_{n+1} - x_n)$
17	M	18	θ	
18	B	8	D	subtract δ
19	*M*	1	θ	discriminate
20	ϕ		D	'clear' floating decimal accumulator
21	A	6	D	print the root x
22	*L*	6	D	

Auxiliary subroutines. To facilitate the coding of entire calculations in floating decimal arithmetic it is useful to extend the range of action of the interpretive subroutines by the addition of auxiliary subroutines for carrying out common numerical processes. The auxiliary subroutines, henceforth referred to simply as *auxiliaries*, are of two kinds. The first kind consists entirely of ordinary orders and will be referred to as *C auxiliaries.* Auxiliaries of the second kind consist largely of 'orders' and hence themselves use the interpretive subroutine. They will be referred to as *X* auxiliaries.

C auxiliaries. These are called in by a *C* 'order' which transfers control to the entry order of the auxiliary. This then carries out the appropriate calculation by means of machine orders. When this has been completed control is transferred back to a point in the interpretive subroutine. This causes the 'control' to resume the obeying of 'orders' starting at the 'order' immediately following the *C* 'order.' Thus the *C* auxiliary can be considered as being an extension of the 'order' code and used as such.

A typical *C* auxiliary is one which replaces $F(A)$ by its square root. The way in which this is done using ordinary orders is as follows. Let p_a and n_a denote the exponent and numerical part of $F(A)$. Two cases arise: If p_a is even, $p_s = p_a/2$ and $n_s = \sqrt{n_a}$; if p_a is odd, then $p_s = (p_a + 1)/2$ and $n_s = \sqrt{n_a/10}$. In both cases the arithmetical operations can be done most simply by using the ordinary techniques of fixed decimal-point working.

C auxiliaries are almost as fast and as economical in storage space as the corresponding routines for fixed decimal-point working. The extra orders required to handle the exponent are largely offset by the orders that would otherwise be required to cater for a large range of the numerical part.

X auxiliaries. These are called in by the special 'order' *X m F*, where *m* is the location of the entry order of the auxiliary. When the *X* auxiliary has completed its part of the calculation 'control' is returned to the 'order' immediately following the *X* 'order' which called the auxiliary into use. This cannot be done by the same method as is used for the *C* auxiliary because the *X* auxiliary itself uses the interpretive subroutine. Instead the *X* 'order' causes the address of the location of the next 'order' to be stored in a certain location for future reference. This record is called the *link.* At the end of the auxiliary is a *C* 'order' which directs control to a set of orders within the interpretive subroutine which use the link to return 'control' to the main programme.

X auxiliaries can use other *X* auxiliaries. To enable this to be done the *X* 'order' causes a list of links to be kept. At any point in the programme the link at the head of the list refers to the 'order' to which 'control' must be returned after the current subroutine has been finished with. The number of links in the list measures the depth to which 'control' has passed within the auxiliaries. A reference number which is adjusted every time a link is added to or removed from the list records this depth. This enables the link at the head of the list to be selected by those orders of the interpretive subroutine which are called into use when the *X* auxiliary has been completed.

The Directory. Reference to the auxiliaries can be facilitated, if desired, by using the following scheme. The auxiliaries are first enumerated in the order in which they are to be read from the input tape into the store. Thus auxiliary no. 1 is the first auxiliary read from the tape, auxiliary no. 2 is the

second auxiliary read from the tape, and so on. When the auxiliaries have been ordered in this way the m-th auxiliary can be called in by the 'order' $X\ m\ L$ or $C\ m\ L$ (depending on whether it is an X or a C auxiliary).

This is achieved by means of a table of switching orders—called a *directory*—stored in consecutive storage locations beginning with $S(hF)$. Each entry directs 'control' (or control) to the first 'order' (or order) of one of the auxiliaries. If, for example, the m-th auxiliary is an X auxiliary whose first 'order' stands in $S(nF)$, then the m-th entry in the directory, that is, the entry standing in $S(h + m)F$, is the 'order' $g\ n\ F$. Thus to call in the auxiliary 'control' is first transferred, by the 'order' $X\ h + m\ F$, to the 'order' $G\ n\ F$ which in turn transfers 'control' to the first 'order' of the auxiliary. Similarly, if the m-th auxiliary is a C auxiliary, the m-th entry is the ordinary order $E\ n\ F$, and the auxiliary is called in by the 'order' $C\ h + m\ F$.

The directory is assembled by the initial orders as the auxiliaries are read from the tape into the store. At the same time the address of the locations, $S(hF)$ of the first entry in the table is placed in the preset parameter location L so that $X\ m + h\ F$ and $C\ m + h\ F$ can be punched as $X\ m\ L$ and $C\ m\ L$ respectively.

The advantage of the scheme is that the master routine can be drawn up in the final form once the set of auxiliaries have been ordered. It is not necessary for the coder to keep a record of the locations in the store of individual auxiliaries. In effect, the scheme shifts the burden of 'book-keeping' from the programmer to the machine. Similar principles are used by the assembly subroutines (for programme assembly) which already exist in the EDSAC library of subroutines (see reference 2).

The use of the input tape as a form of auxiliary store. The EDSAC is not provided with an auxiliary store so that for many problems the shortage of storage space is a real difficulty. This difficulty may be partly overcome by using the paper tape input medium as a form of auxiliary store. There are two ways in which this may be done.

The first way, used when the storage requirements grossly exceed those available, is to carry out the calculation in stages, using the input tape to store numbers and 'orders' not required throughout the calculation. This method will be referred to as *piecewise control* of the calculation.

In the second method the 'orders' of the master routine are placed on the tape and read into the store one at a time, each being obeyed immediately after it has been read. This method is used when the master routine consists of a lengthy sequence of 'orders' not many of which are repeated. If none of these are repeated the scheme takes no longer than putting the complete master routine into the store and then entering it. The principal advantage is that no storage space need be allocated to the master routine. A further advantage is that the progress of the calculation is apparent from the progress of the tape. This method of control will be referred to as *input control.* In the Manchester University Computer Group, tapes drawn up on these lines are referred to as *job-steering tapes.*

Piecewise control. To facilitate this mode of working the Δ 'order' and the initial orders can be used as follows.

The Δ 'order' enables sequences of numbers of any length to be read from the tape into the store when required, overwriting, if necessary, information no longer wanted.

If the initial orders are retained intact during the course of the calculation they can be recalled into use by a *C* 'order'. In this way 'orders' (or orders) can be read from the input tape into the store in the usual way at any time during the calculation. The reading of 'orders' can be halted and control transferred back to the interpretive subroutine by a suitable control combination.

The above scheme together with the use of the Δ 'order' enables quite complicated problems to be tackled.

Input control. In this scheme a special *C* auxiliary—the *input control auxiliary*—is used. This causes 'orders' to be read from the tape and interpreted immediately after they are read. Normally they will be of an arithmetical character or will call in auxiliary subroutines. The *C* auxiliary will continue to read and obey such 'orders' until this mode of 'control' is terminated by a suitable *C* 'order' on the tape itself. This directs control to a point within the interpretive subroutine and in this way causes 'control' to be returned to the 'order' immediately following the *C* 'order' which called the input control auxiliary into use.

Times of operations of the 'orders.' The times of execution of the individual 'orders' are as follows:

'orders'	Times of execution
A,*B*	90 ms.
V,*N*	105 ms.
H, *C*, *G*, *X*, *M*, *P*, *F*	50 ms.
D	140 ms.
ϕ	80 ms.

The times of operation of the *L* and Δ 'orders' are largely determined by the speed of the input and output units. The teleprinter can print six characters per second and the tape-reader can read about 25 characters per second. These rates allow for the time taken for binary to decimal conversion and vice versa.

For comparison it should be stated that each ordinary order of the EDSAC takes $1\frac{1}{2}$ ms. with the exception of the multiplication orders, *V* and *N*, which take 6 ms. each.

From a direct comparison it would seem that the floating 'orders,' other than those used for reading and writing, are about 60 times as slow as the machine orders and hence that a programme using the interpretive subroutine would be slower by the same factor. This is not altogether true because in such a programme fewer 'orders' are needed than would otherwise be necessary as there are no scale factors to deal with and the techniques for counting and for the modification of 'orders' have been streamlined. Moreover, the time taken by the *C* auxiliaries is about the same as that taken by the corresponding subroutine in fixed decimal-point working.

These factors vary from problem to problem but our experience has shown that the reduction in speed varies from about 20 to 1 to about 4 to 1. The reduction of the time taken to code a problem has to be experienced to be believed!

R. A. BROOKER
D. J. WHEELER

University Mathematical Laboratory
Cambridge, England

The authors wish to thank Dr. M. V. WILKES for his encouragement and advice in preparing this paper.

[1] F. ALT, "A Bell Telephone Laboratories 'Computing Machine—1 and II,'" *MTAC*, v. 3, 1948, p. 1–13 and 69–84.

[2] M. V. WILKES, D. J. WHEELER, & S. GILL, *The preparation of programmes for an electronic digital computer, with special reference to the* EDSAC *and the use of a library of subroutines.* Addison-Wesley Press, Inc., Cambridge, Mass., 1951.

BIBLIOGRAPHY OF CODING PROCEDURE

The National Bureau of Standards is forming a central library of notes, reports and technical publications concerned with programming and coding for electronic digital computers.

The collection of material has in view the possibility of increasing the efficiency in the use of high speed digital computers. The material will include for example coding manuals, computing routines, supervisory routines and codes for generating other codes. Those interested in research and instruction in coding practices are invited to avail themselves of this material.

Computation laboratories are invited to submit material to this library. A list of such acquisitions together with a short descriptive review will appear in the future issues of *MTAC*; they will be numbered serially for reference. Material should be sent to the attention of J. H. WEGSTEIN, Computation Laboratory, National Bureau of Standards, Washington 25, D. C.

Some material of this kind has already been included or reviewed in *MTAC*. This bibliography begins with references to eleven such items. The last four items are new.

1. S. LUBKIN, "Decimal point location in computing machines," *MTAC*, v. 3, p. 44–50.
2. HERMAN H. GOLDSTINE & JOHN VON NEUMANN, *Planning and Coding of Problems for an Electronic Computing Instrument.* Institute for Advanced Study, Princeton, New Jersey.
   Part II, v. 1, 1947, 69 p., 21.6 × 27.9 cm. [*MTAC*, v. 3, p. 54–56]
   Part II, v. 3, 1948, iii + 23 p., 21.6 × 27.9 cm. [*MTAC*, v. 3, p. 541–542]
   Part II, v. 2, 1948, 68 p., 7 figs., 21.6 × 27.9 cm. [*MTAC*, v. 4, p. 44–46]
3. FLORENCE KOONS & S. LUBKIN, "Conversion of numbers from decimal to binary form in the EDVAC," [*MTAC*, v. 3, p. 427–431]
4. M. V. WILKES, "Programme design for a high-speed automatic calculating machine," *Jn. Sci. Inst. and Phys. in Industry*, v. 26, 1949, p. 217–220. [*MTAC*, v. 4, p. 116]
5. ANON., *Description and Use of the ENIAC Converter Code.* Ballistic Research Laboratories, *Technical Note* no. 141, Aberdeen Proving Ground, Maryland, November 1949, 23 pages, mimeographed. [*MTAC*, v. 4, p. 170–171]
6. R. W. HAMMING, "Error detecting and error correcting codes," *Bell System Technical Jn.*, v. 29, 1950, p. 147–160. [*MTAC*, v. 5, p. 40–41]
7. M. V. WILKES, "Automatic computing," *Nature*, v. 166, December 2, 1950, p. 942–944. [*MTAC*, v. 5, p. 171]
8. S. GILL, "The diagnosis of mistakes in programmes on the EDSAC," R. Soc. London, *Proc.*, v. 206A, 1951, p. 538–554. [*MTAC*, v. 6, p. 49–50]