Why Not Try a Plugboard?

REX RICE, JR.

IN recent years, a very large proportion of the man-hours expended in designing and constructing new digital computers has been devoted to machines that are "internally programmed." By "internally programmed" we mean a machine in which all of the instructions and operands are contained interchangeably in storage. In contrast, we may consider an "externally programmed" machine as one in which operands and a bare minimum of instructions for subroutine control are contained in storage and the bulk of the instructions are wired into plugboards. The net result of the emphasis on internally programmed computers has been that many of the computing fraternity seem to be accepting the belief that this is the only kind of machine to use for computing. To illustrate the extent to which this belief has gone, in the "First Glossary of Programming Terminology" issued for the Association for Computing Machinery, the word "plugboard" is not even listed. It is merely mentioned under the heading of "storage" as a device that holds information, but its use as a simple and very powerful means of replacing coded instructions certainly has not been emphasized. This is perhaps true because the only externally programmed machines available are combinations of accounting machines and cannot really be considered as computers. To date, with one exception, no large plugboard machine properly designed from the beginning as a computer has been available.

This paper will show, by the use of examples, how programming and logical control are easily accomplished on a properly designed plugboard machine. The abstractions such as relative coding, symbolic coding, and automatic coding, which are essential for programming ease in an internally programmed machine, have no parallel in plugboard machines since programming is direct and simple.

In the following discussion, let us hypothesize a plugboard computer that meets the fundamental objectives of an economical, well-balanced machine. As we discuss this machine and compare its features with those of an internally programmed machine, let us remember that our goal is not to develop abstraction after abstraction and not to find newer and more intriguing ways of how elaborate we can become, but, rather, our goal is to reduce each addition, subtraction, multiplication, division, logical test, and subroutine control to a minimum of programming effort.

In a paper of this length it is not possible to discuss in detail all of the features that make a plugboard machine easy to use, fast, and versatile. Consequently, the discussion is necessarily limited to only a few essential features to show that the technique will work. A description of the machine and then an example will demonstrate a few of the possibilities as well as illustrate programming a plugboard machine.

Description of the Machine

In the following discussion, reference should be made to the machine block diagram in Fig. 1.

For input, the plugboard machine has two punched card feeds, each of which is independently controlled by the plugboard instructions. Information from each feed goes directly into the main storage but is completely buffered so that computing may be done on data entered from the previous card while the next card is being fed. At this point, it should be emphasized that only operands, arguments, and parameters need be entered into the computer through the card readers. All instructions and logical control are "externally" wired into the plugboard by the operator. Additional input is obtained from an array of 10-position switches known as a "parameter board." This device is attached directly to the main channel and through it the operator may change parameters at any desired point during the computation. Each parameter value may be called for by the plugboard routine and if previously set up by the operator, is instantaneously available.

For output, a buffer of ten words connected directly to the printer will allow computing to continue and the next set of output data to be stored, while the previously computed results are being printed. Each word of storage in both the input and output buffers is a part of the main storage and may be used and addressed in the same manner as any other storage location. This input-output setup is second to none in actual usefulness. As higher speed input and output devices become generally available, they need only be attached to the buffers. It should be noted that tapes are generally considered to be a form of output, however, humans cannot read them and every tape must funnel through a printer somewhere.

A second and very important form of output is the "selectable diagnostic list" function. By the mere flip of a switch on the control panel, each number passing through the storage register is automatically listed on the output printer. The storage register is a central buffer tying the main storage to the arithmetic unit. A second switch may be used so that output occurs only at predetermined "break point" locations. No programming effort is necessary to use this feature. This is perhaps the easiest-to-use routine checking device yet devised.

Computer design form is the next major consideration. To be able to retain low pulse frequencies and yet obtain high computing rates, the machine must be highly parallel. The main channel has ten lines so that the nine decimal digits and sign are transferred in parallel. Additionally, the functions controlled from the plugboard are established so that a maximum number of operations may be accomplished in one plugboard program step. This highly parallel operation, or expressed in the nearest equivalent in internally programmed machine language, "multiple-address system," contributes greatly to effective speed. In internally programmed machines, even though several instructions may be packed in one word, the execution of the instructions is necessarily serial and may require much storage access time.

In some instances it is desirable to provide the computer with a "standard board." This means that a board will be wired with all the necessary standard functions such as add, subtract, sine, cosine, square root, etc., appropriate to a class of problems. With such a standard board this machine becomes an internally programmed computer with zero instruction time within subroutines. For this purpose an "interpretive routine decoder" is attached to the main digit transfer channel to allow us to decode a word in a single program step by placing individual digits directly into selected computer control elements. Thus, stored words are interpreted as instructions which function as connectives between wired subroutines. By the use of this

REX RICE, JR., is with Northrop Aircraft, Inc., Hawthorne, Calif.

technique, one-time problems may be run without requiring any special board planning or wiring. It appears that a combination of standard-board use, together with individual boards for repetitive problems, will optimize programming.

Word length in a computer is a much debated subject; however, for engineering and most scientific computations, operands of nine decimal digits seem to be sufficient. Double precision may be used in the rare case where greater accuracy is required. Additionally, nine digits lend themselves nicely to shift control and function control in the distributors, thus assisting in programming ease.

Storage size is also a debatable subject. However, experience indicates that the main high-speed storage may logically take either of two forms. A machine with 100 words plus the buffers in the main storage, and with a drum or similar device providing an optional larger storage addressable in blocks of ten words, will give a very powerful computer. In such a machine, any location in the main storage may be addressed by codes of two decimal digits. A more expensive and also more useful configuration would be a machine with a capacity of 1,000 words, plus the buffers in the main storage. This would require storage addressing registers of three digits. Both of these systems may seem small by comparison to the large computers. However, the experience at Northrop has demonstrated an important fact that should be emphasized: In a properly designed computer, the plugboard is at least equivalent to 1,000 words of zero-access instruction storage. A rack of plugboards represents all of the subroutine storage normally found in drums, tapes, etc. Additionally, the paralleling of many operations allows a small number of program steps to do the equivalent work of a large number of single-address instructions. These features, coupled with good input-output facilities, make comparison with internally programmed techniques and machine size requirements meaningless.

The control of storage access for this plugboard machine is unique in the field of computing. The nearest approach in internally programmed machines is the *B* register concept. Access to all buffer storage may be controlled either directly on a program step by plugboard wires or from the numbers set up in any one of the "storage address registers." Access to the balance of the main high-speed storage on a given program step is controlled by the numbers in any one of the storage address control registers. As shown in the block diagram, the address registers are contained in a separate little computer, yet may be entered directly from the main computer channel. For example (see Fig. 1), on a previous program step, a number may either have been emitted directly into register *1* or it could have been inserted in *1* by bringing it over the main channel. On the program step now active a wire from a program step exit to the function "storage read-out per *1*" would transfer the word corresponding to the number standing in location *1* to the storage register where it becomes available for use.

In addition to the storage address register controls and controlling functions, there are available on the plugboard automatic comparisons of pairs of the registers. These comparisons are set up between registers *1* and *2*, *2* and *3*, and *3* and *4*. By the simple insertion of a single wire, one of the logical tests of the contents of register ($1<2$, $1=2$ or $1>2$) may be used to transfer control. Similar tests are available for comparisons of the other registers. These tests do not have to be programmed; they are automatically available. An example of one use of pairs of address registers is to have the count keeping track of the number of elements in a matrix row or column in register *1* and to count the computing cycles in a loop in register *2*. When these values are equal the $1=2$ impulse may be used to "pickup" a selector which will transfer control to another routine. The usefulness of the storage address registers and the comparison devices may be better appreciated if the basic machine cycle is discussed first.

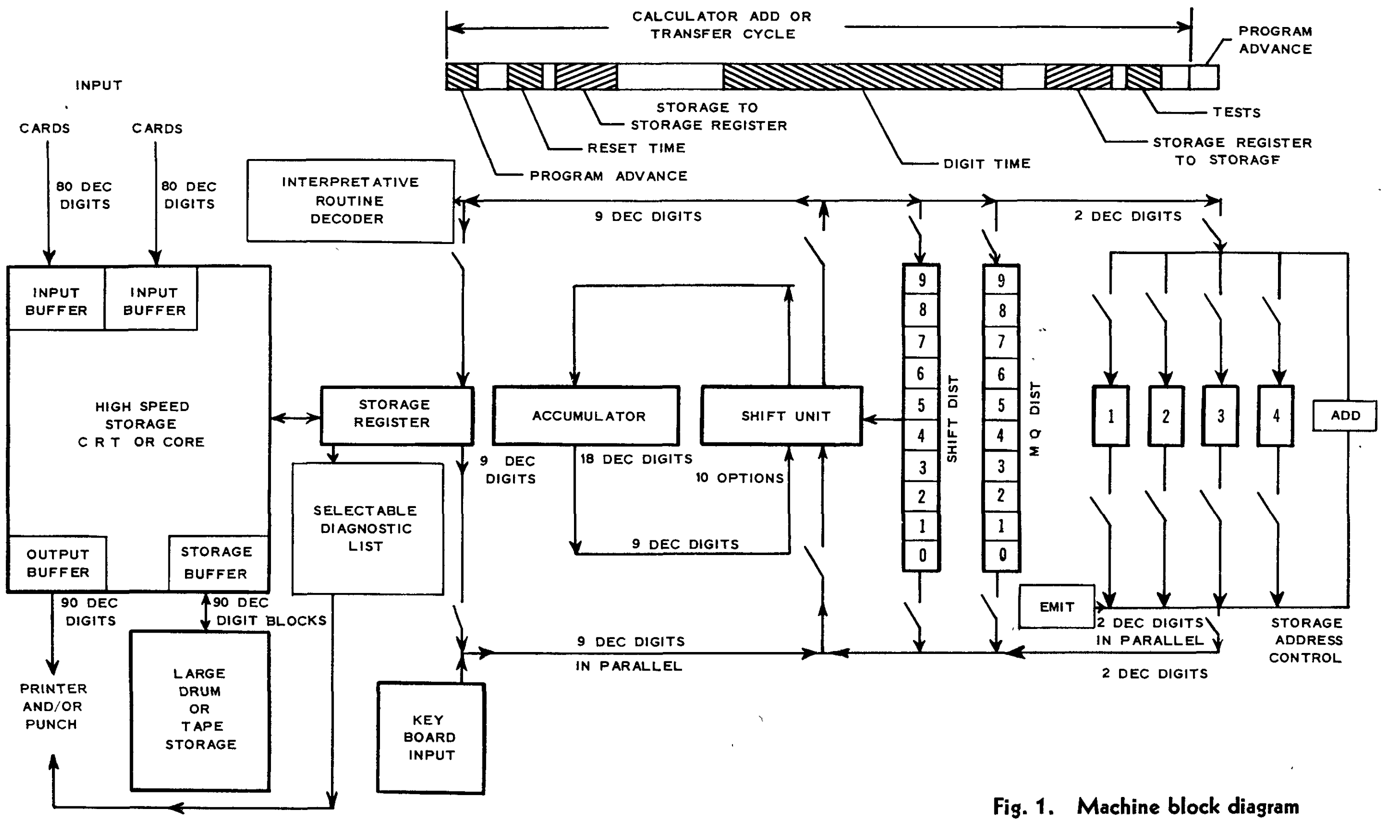
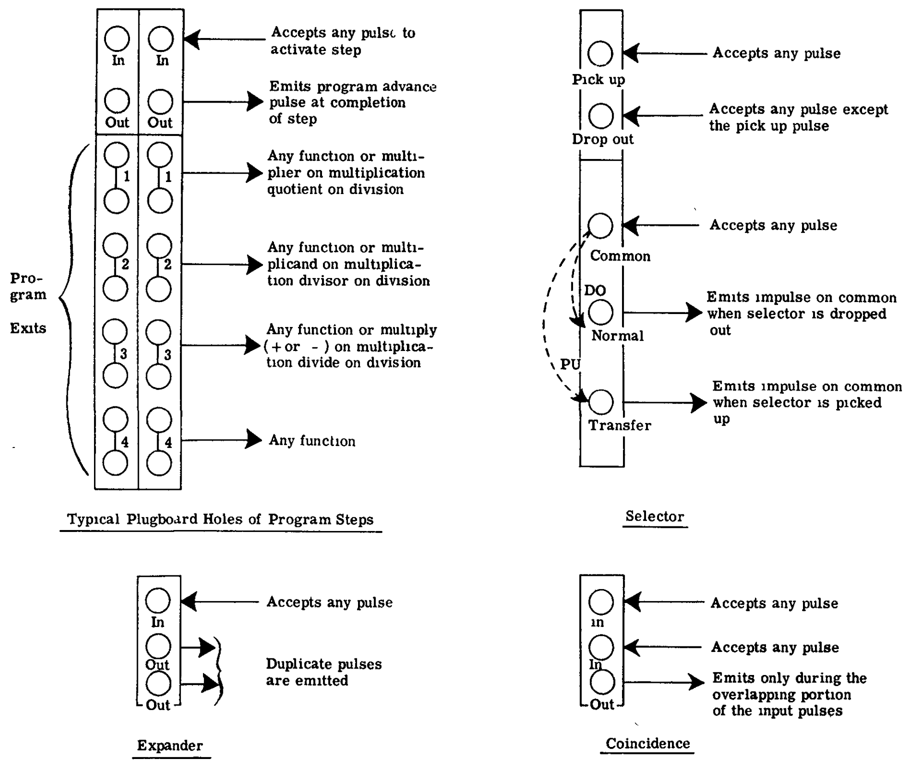


Fig. 1. Machine block diagram



In
In
Accepts any pulse to activate step
Out
Out
Emits program advance pulse at completion of step
Any function or multiplier on multiplication quotient on division
Any function or multiplicand on multiplication divisor on division
Any function or multiply (+or -) on multiplication divide on division
Any function
Program Exits
Typical Plugboard Holes of Program Steps
Pick up
Accepts any pulse
Drop out
Accepts any pulse except the pick up pulse
Common
Accepts any pulse
DO
Normal
Emits impulse on common when selector is dropped out
PU
Transfer
Emits impulse on common when selector is picked up
Selector
In
Accepts any pulse
Out
Out
Duplicate pulses are emitted
Expander
in
in
Out
Accepts any pulse
Accepts any pulse
Emits only during the overlapping portion of the input pulses
Coincidence

Fig. 2. Plugboard elements

The best way to understand machine operation is to study the timing diagram at the top of Fig. 1 where a simplified primary timer diagram is shown. In machine operation this represents the time of one add or transfer cycle and in the solution of a problem represents the sequence of events during one program step.

On the left is the program advance time. At this time in the cycle a pulse is emitted from the plugboard connection known as the OUT hub of the program step that was previously active. This pulse is available after that step has been completed. It may be used to activate the program step under consideration by putting a wire from the OUT hub to the IN of the step we wish to activate.

The next event is the automatic and/or plugboard-controlled reset of various machine registers. Following reset, any number called for, either by plugboard wire control or from address register control, depending on which is wired on this step, will be transferred from the main storage to the storage register. For example, if on a previous step we had emitted a 50 into address register *1* and on this step we called for "storage read-out per address register *1*," then the operand standing in storage location *50* would be transferred to the storage register.

Next in the primary cycle comes digit time. During this period numbers may be transferred around the machine on the main channel. Examples are as follows: The operand in the storage register may be moved through the shift unit matrix and into the accumulator. Note that shifting is accomplished as numbers are moved along the channel and does not require an extra program cycle. If desired, numbers can be moved from the storage register through the shift unit and into an address register or a digit distributor, whose functions are explained later.

After digit time it is possible to place the number in the main storage from the storage register. For example, "storage read-in per address register *3*" may be called for. At the end of the primary time cycle all automatic tests are performed and the appropriate board hubs are activated. Some of these tests are accumulator +, accumulator −, overflow +, overflow −, channel 0, improper divide, and address register comparison. The use of these hubs in logical operations will be illustrated in the example to follow. By comparison with most internally programmed machines where such tests must be programmed, and where much machine time is required, these automatic tests represent advances in both ease of programming and in reduced computing time.

An additional operation that may be programmed to parallel other operations during digit time is the changing of a number in a storage address register. An example of a parallel operation is perhaps the best way to show this. It is possible on one program step to move a number from the main memory into the storage register per address register *1;* then during digit time, to shift it and add or subtract it into the accumulator. Also, during digit time it is possible to use the digit "emitter" and increase the number standing in *1* by any desired amount. Following this it is possible to read the number still standing in the storage register into some new main storage location by calling for "storage read-in per *1*." In this step we have paralleled the following operations: 1. add a word in memory to the contents of the accumulator; 2. shift for proper decimal alignment; 3. change the number in an address register; and 4. make a transfer of the number originally called out from one main storage location to another.

The digit distributors previously mentioned perform several useful functions. The one on the left in the block diagram in Fig. 1 is called the shift control distributor. It may be used in many ways but in addition to its automatic controlling of shifts during multiplication and division, it is available to the programmer during add or transfer cycles for both shifting control and for logical control. A digit brought into the distributor on a previous program step may be used to control shifting on the active step by putting in a wire which calls for "shift per shift control distributor." Optionally, from outlets on the plugboard, a digit previously stored in the shift control distributor may be used to activate a corresponding hub for picking up or dropping out selectors used in logical control.

The distributor on the right in the block diagram in Fig. 1 is the "multi-quotient distributor." In addition to its automatic use in multiplication and division it is also available for logical control during add or transfer operations.

In a paper of this nature it is not practical to discuss all of the various combinations available to the programmer; consequently, discussion has been limited to a few illustrative uses. The ease of use and flexibility when all of the machine components are available to the programmer cannot be fully appreciated until one has run a problem on the machine.

Plugboard Elements

During the previous discussion frequent reference has been made to the plugboard. At this point an examination of some of the elements on the plugboard is in order. A list of the many machine commands available on the plugboard is too long to be given here. However, complete control of all necessary machine elements and their functioning is available on the plugboard. See Fig. 2.

The program step is the fundamental element on the plugboard used to control machine functions. A balanced computer should have approximately 200 program steps available. Each program step will have hubs available with the functions shown in Fig. 2. The IN hub will accept any pulse to activate the step and start a primary timer cycle. During the cycle the four program exits will become active and wires plugged from them to any machine function will activate the function. As frequently happens in parallel operation, if four exits are not enough, one of them may be plugged to the IN of an expander which will then duplicate the pulse on its OUT hubs.

The coincidences and the selectors are available as board functions for logical control and in a limited sense for storage. The selectors have five hubs: a pickup, a drop-out, a common, a normal, and a transfer. The pickup hub will accept an impulse from a program exit, from the OUT of a coincidence, or from one of the digit outlets of the shift or multi-quotient distributors. It causes an internal connection to be made between the common hub and the transfer hub. This connection remains until the drop-out is impulsed, at which time the selector connects the common to the normal. These selectors are electronic and operate practically instantaneously. In conjunction with the selectors, coincidences allow complete and easy logical control.

A coincidence has two IN hubs that will accept any pulse. During the overlapping portion of these pulses the OUT hub emits a pulse. For example: If on program step 2 we wish to test the accumulator for a negative number and transfer control accordingly, we plug one of the program step 2 exits into an IN of the coincidence. The automatic accumulator negative test is plugged into the other IN. If both of these conditions coincide, then a pulse will be emitted at the OUT. This hub may be connected to the pickup of a selector. Through the selector we may control a jump to any other program step by wiring the program OUT to the common of the selector. The normal and the transfer of the selector may be wired to the appropriate IN hubs of any desired steps that start other routines. By proper wiring, program loops, conditional transfer, unconditional transfers, and many forms of logical control may easily be accomplished. It is interesting to note that a series of logical conditions controlling transfer are exceptionally easy to establish by merely using selectors in series.

Use of the Plugboard Machine

A good way to illustrate programming operations, machine functions, parallel operation, and simplicity of board wiring is to use an illustrative example. See Fig. 3. A square root routine similar to that used on desk calculators is a good example of what a parallel machine can do. The following is a version of this routine as worked out by M. L. Lesser and T. S. Eason.

The basis of the process is the following fact from number theory:

$$n^2 - \sum_{j=1}^{n}(2j-1) = 0$$

That is, the sum of the n successive odd integers from 1 through $2n-1$ is equal to n^2. This can easily be verified by examining a table of n, n^2, and the first difference of n^2.

Thus, given n^2, n can be determined by simply counting the terms in the summation of the successive odd integers when this summation is built up to equal n^2. This process can be shortened by performing the operation as a subtraction on radicand groups of two digits each to produce one digit of the root. These 2-digit groups of the radicand are formed by measuring in each direction from the decimal point as in the long-hand method of high school algebra. The operation of this shortcut method is best described by example. See Fig. 3.

The square root routine described follows the foregoing procedure with a single exception dictated by machine convenience only. By writing $\sum_{j=1}^{n}(2j-1)$ as $\sum_{j=1}^{n}(j+j-1)$, the subtracted terms count by successive integers, rather than by successive odd integers.

$$n^2-\sum_{j=1}^{n}(2j-1) = n^2-\sum_{j=1}^{n}(j+j-1)=0$$

EXAMPLE·

$$\sqrt{144} = 12$$

$$144 = \sum_{j=1}^{12}(j+j-1)$$
$$= \sum_{j=1}^{10}(j+j-1)+\sum_{j=11}^{12}(j+j-1)$$
$$= 10^2\sum_{j=1}^{1}(j+j-1)+\sum_{j=11}^{12}(j+j-1)$$

ACCUMULATOR	TRIAL ROOT	PROGRAM STEP NUMBER
144		
-000	000	2
144		
-100	100	4
44		
-100	100	2
- 56 OVERDRAW		
+100	100	4
44 RESTORE	10	5
- 10	10	2
34		
- 11	11	4
23		
- 11	11	2
12		
- 12	12	4
0		
- 12	12	2
- 12 OVERDRAW		
+ 12	12	4
0 RESTORE, FINISH	12	5

Fig. 3. Square root routine

Fig. 4 shows a planning sheet which corresponds almost exactly with the "diagram" and "flow chart" combination of the planning for an internally programmed machine. This is the first of three operations needed to get a program into a plugboard. It is the only operation which requires any original thinking; the other two merely are translations of information planned on this diagram.

In Fig. 4 the large rectangles each represent a program step with the functions that occur on that step shown inside. The IN of the step is at the top and OUT is at the bottom. Selectors are shown as branches that might be used to alter either machine functions or to transfer control. Coincidences are shown as smaller rectangles. The connections of their OUT hubs are noted. The following describes step by step the operations necessary to obtain the square root. The program step numbers are shown just above the upper-left-hand corner of the large rectangles.

Step 1

Setup Step

1. "Reset shift control distributor" which will be used to tell what column of the root we are developing.

2. "Reset multi-quotient distributor" which will be used to tell us when we have made nine tries. This is used to prevent automatically a tenth try that we know will always cause an overdraw.

3. "Read-in addressed storage (main storage location) per number in address register *1*." Since no numbers are on the channel this clears the storage where the root is to be developed.

4. "Advance shift control distributor" to *1*. This function adds one to whatever number is standing in the distributor (i.e., zero, since the shift control distributor was also reset on this step). The shift control distributor is used to locate the position in the accumulator from which the subtraction is made.

5. "Block reset" selectors *1* → *10*. This insures that a block containing all the selectors used are in the "dropped-out" condition.

Step 2

Reduction Step—first subtraction. Reduction occurs if all selectors are normal (dropped out) as shown in Fig. 4.

1. "Read-out addressed storage per *1*." This number will be zero on first cycle and will be the trial root on following cycles. Note that the trial root remains in the storage register.

2. "Out per shift control distributor." This sets up the shift unit in the proper fashion for subtraction of the trial root from the radicand in the accumulator.

3. "Accumulator subtract." The function is through selector *1* so that it becomes "add" if the selector is "picked up."

4. "Test accumulator for negative." A coincidence between a negative number at test time and this program step is used.

5. "Accumulator negative" is also wired directly to "selector *1* pickup." This will cause an addition on steps 2 and 4 for restoring after an overdraw.

Note: If an overdraw occurs during this first subtraction, selector *2* is "picked up." This, in turn, "picks up" selector *4* during program step 3, to change the loop. Selector *4* is the logical control and storage for the indication of completion of one series of subtractions which will develop one digit of the trial root. Since the accumulator is negative, selector *1* is also picked up. This causes an addition instead of a subtraction to occur during step 4, bringing the radicand back to the appropriate value for the shift and next trial reduction. Since selector *2* is picked up, an increase in the trial root is prevented so no subsequent subtraction in its tally position in the storage register is necessary.

Step 3

Modification and Storage of Trial Root

1. "Address channel read-out" connects the emitter to the main channel.

2. "Emit *1*." The emitter is activated to tally the subtraction on the previous step. This function is active when selectors *2* and *3* are "dropped out."

3. Shift "out per shift control distributor" places the digit 1 on the channel in the correct shift position.

4. "Read-in addressed storage per *1*." This stores the modified trial root in main storage.

Step 4

Reduction Step—second subtraction

1. "Read-out addressed storage per *1*." This brings the trial root back to the storage register.

2. "Out per shift control distributor" sets up the shift unit so the trial root may be placed in the proper position of the accumulator.

3. "Accumulator subtract" (add if selector *1* has been picked up). This subtracts the trial root a second time or adds if an overdraw was encountered on step 2.

4. "Multi-quotient distributor advance." This function adds a 1 to the multi-quotient distributor only after each double reduction. This distributor is set up so that a 9 in it indicates the third type of possible overdraw.

5. "Test multi-quotient distributor for 9." This function is set up through a coincidence of a digit 9 in the multi-quotient distributor and the accumulator being positive. When these occur simultaneously, selector *4* is picked up. When this condition occurs we know that another subtraction will always cause an overdraw. Since we have not yet subtracted, no reduction cycle is necessary; consequently, we may proceed with the setup for the next position of the trial root.

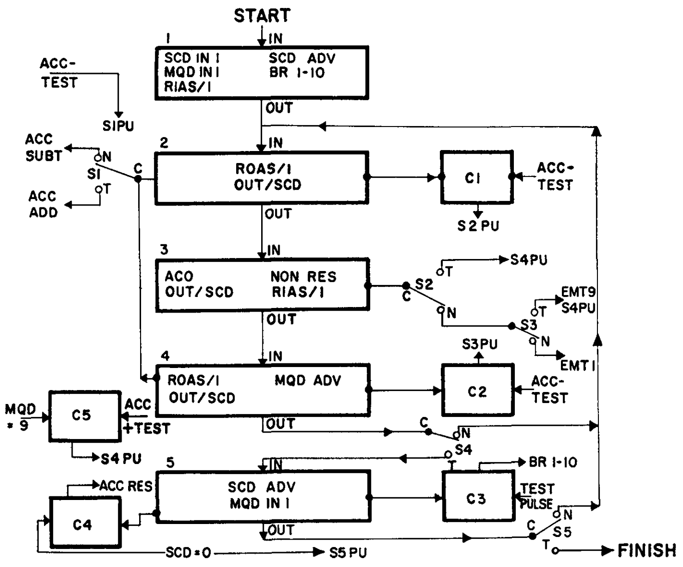


Fig. 4. Diagram and flow chart of square root routine

Setup:
Radicand in accumulator
Decimal at any even period from left address of root in 1
End:
9-digit root in AS/1
Accumulator reset
Negative radicand yields a zero root
ACC = accumulator
S = selector
C = coincidence

The equation in Fig. 3 indicates that for machine convenience we chose to increase the trial root by unity on each subtraction. If overdraw occurred after the first subtraction, it is only necessary to add the trial root back once; however, if overdraw occurs after the second subtraction, it is necessary to add the trial root back twice. This is accomplished by repeating steps 2, 3, and 4 with selectors *1* and *3* picked up.

Selector *1* is transferred automatically any time the accumulator is negative. Selector *3* is picked up by a coincidence of a negative accumulator occurring while program step 4 is active. Note that when selector *3* is transferred, it will cause selector *4* to be picked up on the next cycle through the loop. Also, the transferred function of selector *3* is to "emit 9," instead of "emit 1." Since we are adding into the storage register without carry between digits (using the "non-reset" function) and since the register provides for an end around carry within each digit position, the addition of a 9 has the effect of decreasing the digit contained in the register position by 1. This operation restores the trial root digit under consideration to the proper value if an overdraw occurred on the second subtraction.

Step 5

Second Setup and Shift

1. "Shift control distributor advance." This adds a 1 into the shift control distributor to control the shift so that a new position of trial root is developed on the following cycles. After we have developed a full nine digits of the root, an advance of 1 in this step brings the shift control distributor around to zero. This condition is used to pick selector *5* which signals completion of the routine and also is used through coincidence *4* to reset the accumulator after the last cycle.

2. "Block reset of selectors *1* through *10*." This is accomplished by the coincidence of a test pulse that occurs late in the primary timer cycle and the fact that program step 5 is active. The selectors are reset so the setup for the next trial root cycle starts properly. It should be noted that if "block reset" is impulsed while a pulse is on the pickup hub, that particular selector will remain picked up. This condition occurs with selector *5* after the final reduction cycle.

3. "Multi-quotient distributor in *1*." Since no numbers have been placed on the channel during this program step, this causes the multi-quotient distributor to read in a zero and set it up for the next reduction cycle.

The transferred point of selector *5* may be connected to the IN hub of the next step in the program so that succeeding computations may be started.

The previous discussion on the plan-~~ning of a routine~~ completes the first operation in the setup of a job. It should be emphasized that the planning of this routine is exactly comparable to planning any subroutine on an internally programmed machine. Once the routine is worked out, it may be used in any portion of a program and placed anywhere on the board by merely changing the numbers associated with the program steps, coincidences, and selectors.

The second operation involves the transfer of this information onto a wiring chart. Fig. 5 shows the chart filled in for the square root routine. On the left we find the hubs for the program steps enumerated. Reading from left to right we find the IN hub, the program exist *1*, *2*, *3*, and *4* and, finally, the OUT hub. The selectors and their hubs follow. On the left we find the pickup hub followed by the drop-out, the common, normal, and, finally, the transfer hubs. Next, the expanders are shown. From left to right we find an IN hub and then the two OUT hubs. Finally, the coincidences are shown. The two IN hubs precede the OUT hub. Above and to the right the common hubs and the outlets for the multi-quotient distributor and the shift control distributor are shown.

In practice this wiring chart is used to establish both ends of each wire required. For example, in program step 1, outlet hub *3* shows the designation *E7*. This is the abbreviated code for expander *7*. By looking in the column for expanders we find that the IN hub of *7* is marked *P1-3*. This tells us that the other end of the wire comes from program step 1, hub *3*. Similar conventions are used elsewhere.

The third operation in establishing a program is to wire a board from the wiring chart. This is usually done by placing a check mark in each box as the wire is inserted. Operations 2 and 3 may be performed by coders or machine operators. Contrary to common belief, the time necessary to wire and check a control panel is insignificant compared to the over-all time spent in the analysis and programming of a problem. Control panels for a machine of this type may be wired in 6 man-hours for problems of a difficult nature such as missile launching trajectories, which include all the various aerodynamic and inertial complexities. This, no doubt, compares favorably with checkout time on an internally programmed machine for such a complex problem. Machine time for board check-out may be as low as 1/2 hour for any person who has wired two or three boards previously.

The discussion to this point has been devoted to a hypothetical plugboard computer; however, the techniques involved and the various machine functions discussed are not pure theoretical speculation. Northrop Aircraft has been using a new type of experimental International Business Machines plugboard computer for over a year. This machine closely approximates the one discussed in this paper. Our experience using this computer has borne out our belief that a properly designed plugboard machine can be easier to program, less expensive to rent or purchase, and, most certainly, be as versatile as the largest available computer. It is interesting to note that on this experimental machine the square root takes a maximum of 25 milliseconds, which is equivalent to two "divide-times."

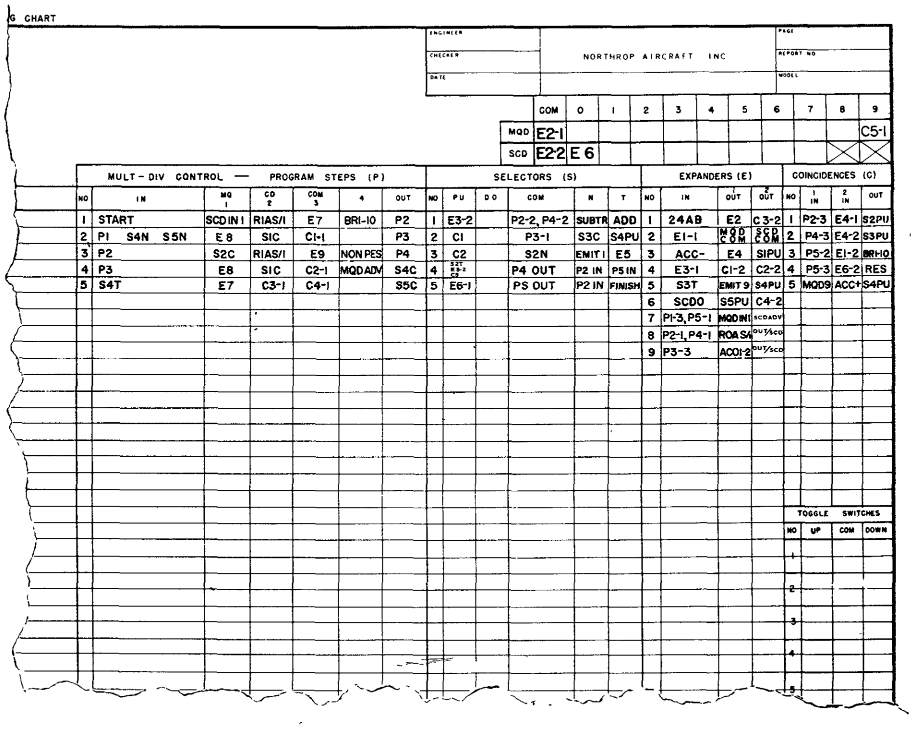
Advantages

The first consideration in the evaluation of computers should be to obtain the maximum computing per dollar expended. The costs of a computing installation, in general, may be divided into two sections. The first section covers the cost of the machine or its rental, as the case may be, plus installation costs. If we consider that the rental of a large, internally programmed machine, or the equivalent amortized cost of a purchased machine, might approximate $25,000 a month, then its yearly cost will be $300,000. By comparison, a production version of the plugboard machine type described would rent for less than $8,000 per month and two of these machines would have more net output than one large, internally programmed machine. The yearly rental for two of these plugboard machines would be $192,000, as compared to the rental of $300,000 for the large, internally programmed computer. Judgment on machine capabilities must take into consideration the total time spent on a computer and not merely be concerned with compute speed alone. The best information available shows that the time on the larger, internally programmed machines is as follows: 50 per cent of the time the machines are input- or output-limited; 25 per cent of the time they are limited by humans in making setups, check-outs, and in trouble-shooting; 25 per cent of the time they are compute-speed-limited. These figures are based on engineering computing requirements and, most certainly, will vary in other applications and for specific problems; however, they seem to represent a rough average.

The second section in the cost of running a computing installation covers the salaries paid to the programming personnel. As shown by experience, between 30 to 35 people are required to operate a large, internally programmed computer. If it is assumed that their hourly rate, including overhead, approximates $6 per hour, then the total salary paid per year will be $360,000. It should be noted here that the manpower costs exceed the equipment costs. Due to the ease of programming, 20 people could produce the same results with two properly designed plugboard machines. At the same assumed labor rate, the total expenditure for these people would be $240,000 per year.

Summarizing these figures, the total assumed cost of operating a large, internally programmed machine is $660,000 and for the equivalent output in a properly designed plugboard machine, the cost would be $432,000.

A few remarks regarding the type of personnel required to operate a plugboard machine are in order since it has been said that extraordinary people are needed to program a plugboard machine. Experience has shown that this is not



G CHART

ENGINEER	NORTHROP AIRCRAFT INC	PAGE
CHECKER		REPORT NO
DATE		MODEL

	COM	0	1	2	3	4	5	6	7	8	9
MQD	E2-1										C5-1
SCD	E2-2	E 6								╳	╳

MULT-DIV CONTROL — PROGRAM STEPS (P)							SELECTORS (S)						EXPANDERS (E)				COINCIDENCES (C)			
NO	IN	MQ 1	CD 2	COM 3	4	OUT	NO	PU	DO	COM	N	T	NO	IN	1 OUT	2 OUT	NO	1 IN	2 IN	OUT
1	START	SCD IN 1	RIAS/1	E7	BR1-10	P2	1	E3-2		P2-2, P4-2	SUBTR	ADD	1	24AB	E2	C3-2	1	P2-3	E4-1	S2PU
2	P1 S4N S5N	E8	S1C	C1-1		P3	2	C1		P3-1	S3C	S4PU	2	E1-1	MQD COM	SCD COM	2	P4-3	E4-2	S3PU
3	P2	S2C	RIAS/1	E9	NON PES	P4	3	C2		S2N	EMIT 1	E5	3	ACC-	E4	S1PU	3	P5-2	E1-2	BRHO
4	P3	E8	S1C	C2-1	MQD ADV	S4C	4	S2T E3-2 C3		P4 OUT	P2 IN	P5 IN	4	E3-1	C1-2	C2-2	4	P5-3	E6-2	RES
5	S4T	E7	C3-1	C4-1		S5C	5	E6-1		P5 OUT	P2 IN	FINISH	5	S3T	EMIT 9	S4PU	5	MQD9	ACC+	S4PU
													6	SCD0	S5PU	C4-2				
													7	P1-3, P5-1	MQD IN1	SCD ADV				
													8	P2-1, P4-1	ROA S4	OUT/SCD				
													9	P3-3	AC01-2	OUT/SCD				

TOGGLE SWITCHES			
NO	UP	COM	DOWN
1			
2			
3			
4			
5			

Fig. 5. Wiring chart

true. The people who program the equipment at Northrop have the same background, experience, and abilities, on the average, as the persons programming internally programmed computers in other installations. However, there is one difference in our operation. The programmer is shown that he has available the fundamental building blocks from which he can construct any desired computer. As a result, the programmer is unhampered by the limitations of a built-in operational code. The flexibility and potentialities of a plugboard computer are limited only by the ingenuity of the programmer. In effect, he is told, "Here are the components; design a special-purpose computer which most efficiently does your problem." This approach stimulates original thinking and results in increased interest, initiative, and productivity on the part of the programmer.

Table I. Record of Average Learning Hours for Programmers

Programmers' Background	Time Required to Learn Machine and Program Simple Problem, Hours	Time Required to Learn Sophisticated Use, Hours
IBM *604*, etc.	4	20
Engineers; no computing experience	8	24
Engineers; previous CPC computing experience	2	16
701 programming experience	2	8

Table I shows an actual record of the average learning hours used for people programming the machine now at Northrop.

In the comparison of computer types, the second and final consideration is the ability of the computer to do the problems experienced in a given installation. Table II lists most of the important features for both types of machines.

Table II. Comparative Features of Large Internally Programmed and Plugboard Machines

	Large Internally Programmed Machine	Plugboard Machine
Operand storage	Electrostatic or core, drum, tape, cards	Electrostatic or core, punched cards
Instruction storage	Anywhere in storage	Plugboard wires, machine components, main storage
Operand access	Variable, program for optimum	Automatically optimum
Instruction access	Variable, program for optimum, abstractions increase net access	Effective zero-access
Rapid access storage	Crowded by stored instructions	All available for operands
Logical control	Always programmed, every command serial	1. Tests fully automatic 2. Performed in parallel with computations 3. Series of conditions easy to construct 4. Simple restoration 5. Many types of control more easily available
Operation	Serial	Highly parallel
Computing rate	Should be high because of serial operation	May be order of magnitude lower
Programmer's "feel"	Usually abstract	Direct, physical
Input	Cards, tape, console keys	Two card feeds, parameter board, plugboard
Output	Line printer, cards, tape	Line printer, cards, tape
Physical size of machine	Assume *704* or *1103* as unity	Estimated one third

This paper has described a computer which is small in size, large in ability, and unsurpassed in versatility. After considering the flexibility, the ease of programming, and the reduced operating cost, the obvious question is, "Why not try a plugboard?"

Discussion

H. R. J. Grosch (Chairman): I would like to begin the discussion by asking Mr. Rice if he has had any experience in teaching this type of equipment to someone who started in the stored program area, or something similar.

My criticism is that it seems to me a very difficult machine to learn in comparison with the stored program.

Rex Rice, Jr: One of the questions that has been asked about this machine by people who have viewed it has been, "Well, isn't it difficult to program?"

We have had people with varied backgrounds actually put problems on the machine and we consider that they are ready to try a problem when we have described the machine's functions to them and put a few wires on the board with their help (but essentially with their watching the operation). We then turn a simple problem over to them.

This simple problem is shown in the middle column of Table I. People with IBM *604* experience of the type where they are familiar with a plugboard take about 4 hours to be able to start thinking about this machine. Engineers with no computing experience take about 8 hours. Essentially we have to show them what the program steps, selectors, coincidences, expanders, and things of that nature mean and give them a physical feeling for what is happening in the machine.

Engineers with previous CPC experience are done in about 2 hours. They merely have to find where the holes are in the board. The same applies to people who know how to set up logical operations, who know what routines mean, and things of that nature. It takes us about 2 hours to give, for example, *701* programmers the location on the board of the hubs. This varies with individuals, but I have given you an average figure.

On the right, people with *604* experience take about 20 hours to become sophisticated, meaning that they can code a problem of their own, get it to run without undue difficulty, and they begin to see the potentials of parallel operation.

For engineers with no experience, since they have to learn loops and programming, it takes about 24 hours; for engineers with previous computing experience, about 16 hours; and for internally programmed experience, about 8 hours.

I believe that this record should stand against any internally programmed machines.

V. M. Wolontis (Bell Telephone Laboratories): Opposition to plugboard machines is not universal in the data-processing area. There are many problems—for instance, calculation of means and variances of tabular data—solved more economically by a *604* than by a *650*.

J. Belzer (Battelle Memorial Institute): Are you discussing a specific computer? If so, which one?

Rex Rice, Jr.: The specific computer that I have had available for this experience is the experimental machine in operation at Northrop Aircraft. This machine was produced by IBM.

As far as the hypothetical machine is concerned, it is not as yet announced by any computer company. I hope that my remarks here will prod them in this direction.

H. Robbins (Hughes Aircraft): Have you any features comparable to the machine aids to coding possible with internal storage? That is, a program library, automatic assembly of routines, etc.?

Rex Rice, Jr.: This is a difficult question to answer in a short time. However, let it suffice to say that the library of routines will represent, as we illustrated in the square-root routine, a worked-out wiring chart which some programmer has made and has boiled down to the most efficient subroutine possible. This then gives us a complete library, as they are developed, of subroutines.

The "splatter function" (effectively the interpretive routine decoding device on the channel) was purposely placed there so that we could decode one word in the memory, and this allows us logical control of the transfer between subroutines.

I would like to emphasize that fact, because if you have not worked with a plug-board machine it is not obvious. All of the instructions within the subroutine are contained in zero access storage on the plug-board. The word necessary to transfer control to any other routine desired at the end of that subroutine may or may not, as desired, be contained in the memory. There is nothing in the internally programmed machines that cannot be duplicated easily on this machine. Abstractions become useless because after all of the abstractions are developed on an internally programmed machine you merely approach what we are already doing.

K. F. Powell (Babcock and Wilcox): Despite that fact that this machine is quite attractive, it is not apparent why it should reduce the size of the programming staff.

Rex Rice, Jr.: I will have to base my remarks on experience. The programmers after the first few problems do not have to spend time worrying about the resetting of memory registers which represent the storage for logical control. They begin to get a feel of the fact that a selector is the logical control. They merely insert a selector in the routine, then at any convenient time much later, days later, in their problem or planning if they wish they stick a drop-out wire in one of the hubs of the program step to be sure the selector is dropped out. There is no modification necessary. They do not need subroutine control to make these changes.

This is one example. It is a very difficult thing to define if a person has not actually worked one of these machines. Effectively, it is what we meant in the slide by showing the programmer's feel for his routine. The best answer I can give is that this feel is direct and physical as compared to a relatively abstract feel in the case of an internally programmed machine, particularly when abstractions are used for coding purposes.

R. W. Bemer (Lockheed Missile Systems): Is it true that a certain large user of computers with five IBM *650*'s on order would dearly love to have one of these machines?

Rex Rice, Jr.: I pass.

Characteristics of Currently Available Small Digital Computers

A. J. PERLIS

THE purpose of this paper is to survey a rather well-defined group of computing machines. No attempt will be made to place these machines in order with respect to certain applications. However, a listing of their pertinent characteristics will be given.

The last time the Joint Conference was held in Philadelphia, the proceedings were concerned with "large" general-purpose digital computers, e.g., Whirlwind, Mark III, ERA *1101*, the Princeton series, etc. As a class, they may be roughly characterized by satisfying (*a*) minimum cost in excess of $200,000 and (*b*) a maximum access time to main memory of less than 1 millisecond.

The present conference is concerned with "small" digital computers. This paper will further limit the conference subject by restricting its discussion to the class of "small," general-purpose electronic digital computers, which may roughly be characterized by:

1. A maximum cost less than $150,000.
2. An internal storage of at least 1,000 words.
3. Stored programs.

Thus, such well-known (types of) computers as the IBM CPC, the Remington Rand *409*, the whole class of digital differential analyzers, and other special-purpose computers are not included. Specifically, the following computers, with their respective manufacturers, are to be surveyed.

1. The *650*, International Business Machines Corporation.
2. The *102d*, National Cash Register Corporation.
3. The *30-203*, ElectroData Corporation.
4. The Miniac, Marchant Research Corporation.
5. The Elecom *120*, Underwood Corporation.
6. The Alwac, Logistics Research, Inc.
7. The Circle, Hogan Laboratories.
8. The Monrobot, Monroe Calculating Corporation.
9. The *E 101*, Burroughs, Inc.

All of the computers listed are decimal with the exception of numbers 6 and 7, which are binary. Several of the firms, numbers 2 and 5 in particular, also manufacture binary machines. These machines are not generally intended to be used in on-line control applications but rather as scientific and engineering calculators and as units of data-processing systems in commerce. Hence, it is no accident that almost all use the decimal system.

The extreme pertinence of the conference subject at this time indicates the value placed on these machines. Industry and commerce, the real source of sales in this business, are definitely awakened to the value of these machines. Perhaps this awakening has not always been accompanied by understanding but it exists and is widespread. The questions have changed from, "What are they?" and, "How will it save money and fit the organization?" to, "Should a digital computer be rented or purchased now?" and, "Should it be large or small?"

For many of these computer firms, a day of prosperity seems about to arrive. Considering some of their checkered histories

A. J. PERLIS is with Purdue University, West Lafayette, Ind.